Technical Manual
Project 6: Laser Communication

20/20 Vision (Group 24):

Denzel Delnawaz: 490424744

James Young: 490402708

Rocco Liu: 490503016

Bhavesh Balaji: 490400667

MTRX3700 Mechatronics 3

School of Aerospace, Mechanical and Mechatronic Engineering

The University of Sydney

November 2021

Declaration

We hereby declare that this submission is our own work and that, to the best of our knowledge and belief, it contains no material previously published or written by another person.

20/20 Vision (Group 24):
Denzel Delnawaz: 490424744
James Young: 490402708
Rocco Liu: 490503016
Bhavesh Balaji: 490400667

21 November 2021

Contents

List of Figures

List of Source Code

Nomenclature

List of Symbols

A	Large Signal Gain (V/mV)
d	Distance (m)
L	Free-space path loss (dB)
λ	Wavelength (m)
V_{out}	Output Voltage of Op-Amp (V)
V_+	Positive Input Terminal Voltage of Op-Amp (V)
V_-	Negative Input Terminal Voltage of Op-Amp (V)

List of Acronyms

COTS	Commercial Off-The-Shelf
COVID-19	Global Pandemic Disease of 2020-2021
GUI	Graphical User Interface
IR	InfraRed Light (see Electromagnetic Spectrum)
LED	Light Emitting Diode
MTRX	Mechatronics
NEC	Nippon Electric Company InfraRed Protocol
PC	Personal Computer
PCM	Pulse Code Modulation
PIC	PIC18F452 micro-controller
PWM	Pulse Width Modulation
RS-232	Serial Communication Standard
TTL	Transistor-Transistor Logic Family
TV	Television
USART	Universal Synchronous and Asynchronous Receiver-Transmitter

Chapter 1

Introduction

1.1 Document Identification

This document describes the specification and design of a proof-of-concept free-space laser communication mechanism. This document is prepared by 20/20 Vision (Group 24) for assessment in MTRX3700 in 2021.

1.2 System Overview

Project 6, Long Distance Optical Communication was chosen from a list of possible project ideas. The problem statement adhered to is: test the viability of optical free-space data communication.

The project centres around the idea of using a laser for free-space communication, sent and received by different devices and allowing the transmission of various forms of data.

The transmission side of the system will offer a user interface to select and input data which is converted into pulses of a laser signal over a chosen protocol.

The receiver side will receive these pulses through air and convert them back into digital logic which is then output in an appropriate form. See Fig. 1.1.

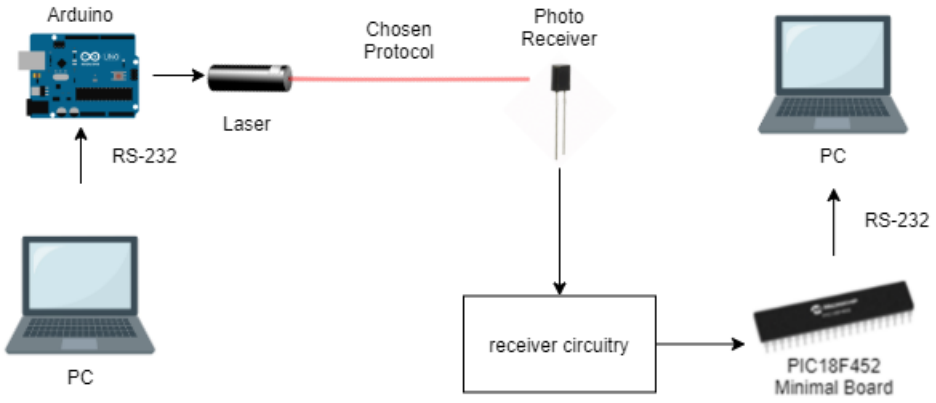


Figure 1.1 – System Overview

1.3 Document Overview

This document provides full engineering documentation for the proof-of-concept laser communication system that has been developed and how to use it. It contains detailed technical explanations sufficient for an engineer to assemble, test, calibrate, or repair the proof-of-concept system.

Included are sections on the overall system and user interface design, the hardware and software components of the system, system tests and performance as well as safety warnings when using the system and a conclusion.

If a more simplified non-technical explanation of components or the system is required, please see the User Manual in the Reference Documents, Section 1.4.

1.4 Reference Documents

The present document is prepared on the basis of the following reference documents, and should be read in conjunction with them. Other references included are more specific and only for detailed technical reference.

- “User Manual”. 20/20 Vision, November 2021.
- “Principles of Optics”. E. Born and M. Wolf, 1999.

Chapter 2

System Description

2.1 Introduction

This project involves testing the viability of free-space communication using line of sight laser and infrared light by sending a variety of data types, through different data protocols. Free space communication is currently being used almost exclusively in Space due to the lack of atmospheric interference, minimal attenuation and ability to use higher power lasers. However, this project attempts to identify plausible methods that the same free-space communication can be applied on the ground, understanding reasons why it is currently **not**, and how these problems can possibly be fixed.

Laser communication offers some different properties compared to traditional radio or wired communication, making it suitable to various different applications. These advantages include:

- Very high bandwidth (dense wave digital multiplexing)
- Can be used in regions with interference, due to EMI and RFI immunity
- Lower transmission loss compared to wire
- Long transmission distances in right circumstances

- Lower power consumption and infrastructure required
- No injury due to electric shock
- Secure transmission due to difficulty of interception and tapping

To explore these benefits, the system as described in Section 1.2 will be split into primary transmitter and receiver modules. The transmitter will take data from a user input, and modulate it using a chosen protocol and a chosen type of electromagnetic wave physically towards the receiver. The receiver will detect the measured pulses with a sensor, demodulating and decoding it back into the original data transmitted.

To analyse effectiveness of a laser communication system in different scenarios, various protocols and types of light were used. The following Fig. 2.1 shows the optional configurations explored.

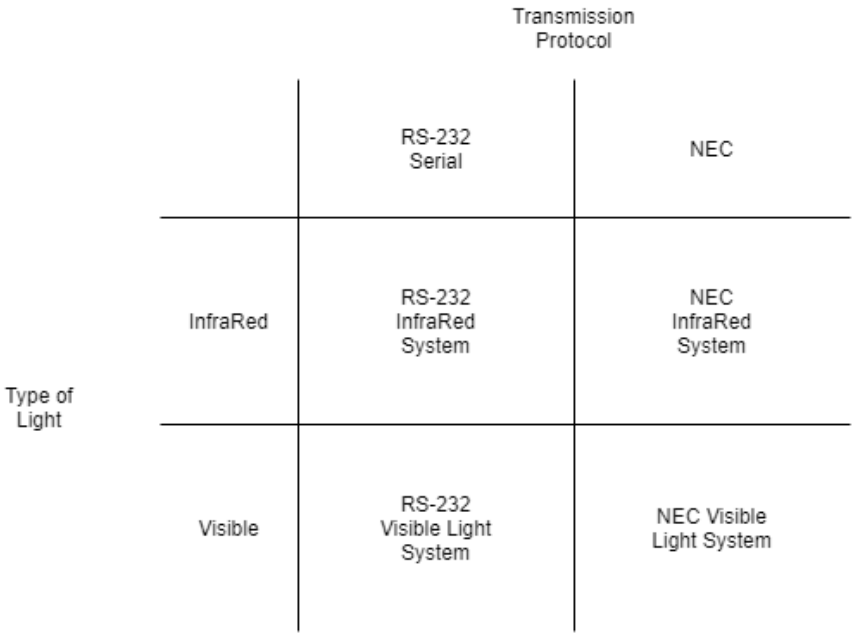


Figure 2.1 – Types of System Configurations

These were each chosen due to various distinct perceived benefits. For protocol, NEC is a slow and more reliable method of data communication already used in infrared remotes commercially, whereas RS-232 Serial Protocol is a widely understood and utilised high speed data communication protocol that is being phased out due to

inaccuracy. Further, for light type, visible light is obvious and easy to test with, but can be damaging to the eye, whereas InfraRed is not visible and not damaging to the eye, can travel further theoretical distance but being invisible can mean people are unaware.

2.2 Operational Scenarios

This system has the potential for both long ranged and short range use-cases.

Short Range:

- COVID Check-In
- Braille Replacement

Medium/Long Range:

- Military Communication
- Inter/Intra Satellite Communication
- Underwater ROVs
- Optical Fibre Data Network Replacement

However for the scope of this project and the available equipment, it was more justifiable to implement for short-range use cases, hence focusing more on the current issue that is *plaguing* us all with COVID and the rather clunky QR check-in system.

It's use as a COVID check-in device would be to replace the whole scanning, clicking and data-input process that is currently done. Rather it would be more efficient by a simple click of a button on the user's mobile device, using the inbuilt flashlight or IR diode, it will beam all the relevant information across to the client side for

record-keeping. This whole process should take less than 2 seconds if done properly (based on experimental testing).

Below is a Use-Case Diagram for the **COVID Check-in** example:

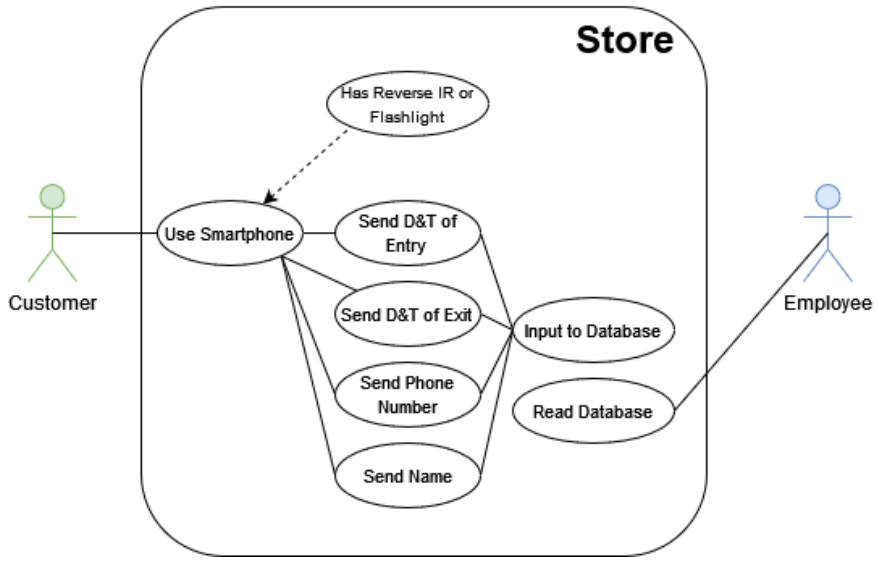


Figure 2.2 – Use Case Diagram

However there are factors that need to be accounted for prior to rolling out this system. It is very much an exposed process whereby, if a third-party actor was to have their own variation of the infrared receiver and decoder, they would be able to phish this information by standing in close proximity to the customer. This can be prevented by adding and sending an *authentication string* to the transmission cycle which is generated through RSA.

From a hardware side, this is very much an open-loop process where the customer has no indication that the data has been sent across and securely stored into the database without corruption due to the lack of an IR receiver on the customer's phone. However this can be circumvented by using an LED or LCD display on the employee's side to notify the user if their data has been transmitted, and they are allowed to enter the shop.

Another hardware point of failure is incomplete or incorrect transmission of data. For this use case specifically, there is not any *significantly private* information being shared, but it is still important to ensure that the data can be transmitted and reconstructed without errors as it is important information for the Australian Government. From our testing, we found that to avoid interference from the environment, modulating the signal is very effective in minimising any data corruption. When compared to un-modulated signals, the transmission rate is slower, but the unmodulated signals present significantly higher variance and error in their reconstructed strings.

Finally another point of failure is the impatience of the customer. This process involves holding up the phone with close proximity (1 to 5 m) of a receiver until the data is transmitted. For less signal attenuation and a lower chance of third party involvement, using a directional diode rather than an open LED would be ideal. However, this presents the problem that the process is entirely dependent on the user to maintain Line-Of-Sight with the receiver. If they drop their phone or move it beyond the receiver by accident, it will cut off communications and could introduce potential corruptions into the data being sent.

2.3 System Requirements

In order to judge the efficacy of the system, several requirements were set out early in development. These are overall system requirements rather than modular system requirements. **The core system requirements are the ones directly related to the COVID Check-In use case and are the primary measures of success.** The additional system requirements are to test the viability to expand the process to other use-cases as well, however they carried little to no weighting in the judgement of success of the system.

Core System Requirements

- CSR 1: The transmission of data *shall* be instantiated from a GUI rather than

console input.

- CSR 2: The transmission of data *shall* be possible using both IR and Laser.
- CSR 3: The transmission of data *shall* be 100% accurate in daylight, artificial light and night-time.
- CSR 4: The reception of data *shall* be 100% accurate for up to 10 seconds of constant transmitting.
- CSR 5: The system *shall* function up to a distance of 50 m.
- CSR 6: The system *shall* be able to transmit Strings.
- CSR 7: The system *shall* be able to transmit Numerics.
- CSR 8: The system *shall* be technically simple enough so as to design and construct in less than six weeks.
- CSR 9: The system *shall* meet requirements of the defined Project 6 for MTRX3700 in 2021.

Additional System Requirements

- ASR 1: The system *should* be able to transmit Images.
  - ASR 1.1: The system *should* be able to transmit a 240 x 240 image in less than 5 minutes.
- ASR 2: The system *should* be able to transmit Sound.
- ASR 3: The transmission of data *should* be tested using at least 2 different methods.
- ASR 4: The system *should* cost less than AUD$200.

2.4 Module Design

The system developed for this project involves using two separate computers attached to their own respective microcontroller, which communicate to each other by specifically designed circuits that can support free-space communication. **These can be fissioned into two main modules: “Transmitter” and “Receiver” (with also, a basic "User Interface")**, see Fig. 1.1. Within these main modules, each has 3 sub-modules that are common to both sides: “Hardware”, “Software” and “Electrical”.

The ‘Transmission’ module’s hardware component consists of an Arduino connected to a computer which contains the software capable of generating modulated and unmodulated pulses to output strings, numerics, images, and sound. The electrical component of this module involves the powering of Laser and IR Diodes.

The ‘Receiver’ module’s hardware consists of a PIC18F452 micro-controller to adhere to project specifications. The PIC contains software capable of reading the demodulated data and reconstruct them into meaningful data types for presentation. The hardware and software are less coupled here due to the PIC being significantly more accessible to program pin by pin and register by register, allowing for greater reception ability using Timer and Interrupt facilities. The electrical component of this module involves creating amplifier circuits to boost weak signals at long range, convert from analog to digital TTL logic, and decoding the modulated signals.

The operational scenarios considered place certain requirements on the laser communication system, and on the modules that comprise it. Some requirements here are dependant on the decision to implement Additional System Requirements mentioned in Section 2.3.

2.5 Module Requirements: Receiver

2.5.1 Functional Requirements

This section describes the functional requirements of the receiver module – those requirements that must be met if the module (and system) is to function correctly.

Inputs

- The receiver *shall* detect/sense the presence of particular electromagnetic waves (IR and visible light) to fulfil CSR 2.
- The receiver *shall* detect signals of both the NEC protocol (see Section 2.9) and RS-232 serial protocol.

Process

- The receiver *shall* convert the analog waves received into a digital signal, according to TTL logic standards.
- The receiver *shall* collate the digital signals into meaningful bytes of data according to the input protocol.

Outputs

- The receiver *shall* output the bytes in forms that are meaningful according to the input, including:
  - Parsing data to a PC using RS-232 serial protocol.
  - Outputting data as a signal to a speaker, using PCM (see Section 5.3.3).

Timing

- The receiver software *shall* be interrupt driven.
- The receiver *shall* meet timing requirements of protocol standards mentioned.

2.5.2 Non-Functional (Quality of Service) Requirements

Performance

- The receiver *should* have an accuracy of 100% correct data communication to fulfil CSR 4.

Design Constraints

- The receiver *shall* use the PIC micro-controller as the primary processor, interfacing other peripheral components to fulfil CSR 9.
- The receiver software *shall* be written in C18 to fulfil CSR 9.
- The receiver *should* primarily use COTS components to fulfil ASR 4.

2.6 Module Requirements: Transmitter

2.6.1 Functional Requirements

This section describes the functional requirements of the transmission module, and only if these requirements are met will the module (and overall system) function as intended.

Inputs

- The transmitter *shall* be interfaced with a GUI to input what data type is being transmitted
- The transmitter *shall* have appropriate power circuitry to maintain continuous operation of laser diode for at least 10 seconds to fulfill CSR
- The transmitter *shall* be able to transmit un-modulated, modulated and framed data types through Binary, NEC and Serial respectively.

Process

- The transmitter *shall* parse data bytes into individual bits and transmit them across.
- The transmitter *shall* be able to take dynamic user input mid transmission to change the message being sent.
- The transmitter *shall* be able to take in multiple types of input ranging from Strings, Numerics, Images and Sound.

Outputs

- The transmitter *shall* be capable of emitting both visible (red) and IR light
- The transmitter *shall* be capable of emitting a beam of up to 50 m in accordance with CSR 5.
- The transmitter *shall* be able to output consistent pulses of 100% accuracy for up to 10 seconds.

Timing

With regards to timing, the transmission side is purely event driven every time a byte is sent across from the connected computer via RS232 cable. Event driven functions on Arduino are very similar to interrupt in the sense that it will only trigger the laser transmission asynchronously, unlike polling on the PIC.

2.6.2 Non Functional Requirements

Performance

The transmitter *should* have an accuracy of 100% accurate data transmission.

Design Constraints

- The transmitter *shall* use an Arduino as the primary processor and interface with the laser peripherals.
- The transmitter software shall be written in Arduino, Processing 4.0 and Python
- The transmitter software shall use COTS components.

2.7 Module Requirements: User Interface

2.7.1 Functional Requirements

This section describes the functional requirements of the user interface module – those requirements that must be met if the module (and system) is to function correctly.

Inputs

- The user interface *shall* receive inputs from a human.
- The user interface *shall* provide four distinct options for interaction/input.

Process

- The user interface *shall* coordinate user inputs to corresponding data outputs.
- The user interface *shall* access files on a computer.
- The user interface *shall* manage serial communication to the transmitter module.

Outputs

- The user interface *shall* output a serial communication data stream to the transmitter module.

Timing

- The user interface *shall* be event driven, event being byte read.
- The user interface *shall* meet timing requirements of selected baud rate.

Failure Modes

- The user interface *shall* limit the user to accepted inputs only, exiting gracefully in the event of an input error.

2.7.2 Non-Functional (Quality of Service) Requirements

Performance

- The user interface *should* have a 100% accuracy in outputted data.

Interfaces

- The user interface *shall* utilise a connected mouse and keyboard to allow interaction for testing purposes.
- The user interface *should* allow for touch screen interaction in a mobile application.
- The user interface *shall* pass data using an RS-232 cable to join interface to transmitter module.

Design Constraints

- The user interface *shall* be written in Python.
- The user interface *should* utilise existing packages and features of user interfaces in Python.
- The user interface *shall* require access to local storage.

2.8 Conceptual Design: Receiver

For the receiver to satisfy defined requirements, a sensor is connected to the PIC micro-controller and output to peripheral devices. A simple block diagram is shown in Fig. 1.1.

The photodiode sensor is used to convert a light intensity to electronic signal such as current or voltage. This is connected to electronic circuitry, which processes and prepares the electronic signal to be received by a micro-controller in hardware. This circuitry will likely involve an electronic filter and op-amp for signal amplification. Then, the appropriate digital signal is received by the micro-controller and collated in software as bytes ready to output to an appropriate medium. See Fig. 2.3 for a visual explanation.

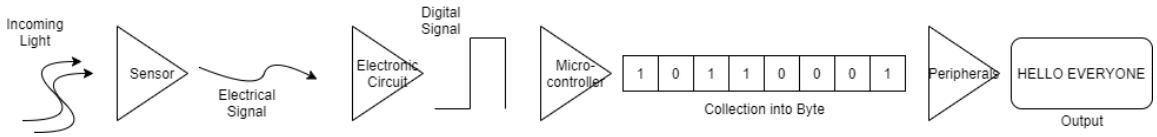


Figure 2.3 – Receiver Concept Block Diagram

It is likely that the PIC USART module will be used for serial communication on the receiver to the PC, and timer interrupts to determine protocol timing.

It is important also to study the attenuation of electromagnetic waves [3] as a limit to the distance the free-space laser communication system can transmit data. Generally, the attenuation of a wave is defined as

$$L = 20 \log \left(\frac{4\pi d}{\lambda} \right), \tag{2.1}$$

where L is free-space path loss in decibels (dB), d is distance in metres, and λ is wavelength in metres. Generally, this implies that IR signals can travel further, and that at a distance on 50m, the signal will have a loss of approximately 180dB. This calculation, though, does not consider other factors such as light pollution or weather.

With this information, full link budgets could be calculated. But, as we do not have the gain (dBi) of the transmitter or receivers, it is not calculated here.

2.8.1 Assumptions Made

Assumptions include:

- Commercial electronic components are available that are able to fulfil the above requirements.
- The transmitter and receiver can be aligned and focused as desired.
- Light pollution will not have an extremely significant affect on the signal.

2.8.2 Constraints on Receiver Performance

Constraints include:

- Micro-controller clock frequency, likely to be a limiting factor on the receiver speed.
- Size of the receiver, when small it is much more difficult to align the beams and test properly, especially at larger distances.

2.9 Conceptual Design: Transmitter

The transmitter module for this project should be capable of emitting both modulated and non-modulated visible and IR light. But in order to understand how the transmission module works, we have to delve a bit into the communication protocols being used in this module.

A computer will be connected to the Arduino micro-controller via Serial RS232. This is a common wired communication protocol which uses framed data bytes and a software handshake to initiate and engage communication. Typically RS232 has a 9 pin connector (before USB A) and transmits at voltage levels ranging from either $\pm$15V or $\pm$3V. Each data byte is framed typically with a Start Bit, Parity Bit,and Stop Bit. And only when the whole frame is sent across, will it instantiate the reception/transmission of another one into the Serial buffer. This leads to the next component of RS232 which is the input and output buffer. As transmission devices push data into the Serial stream, they need to be stored in an output buffer first and once the input buffer on the receiving end is cleared, it will send a "Ready to Receive" flag across. Every ASCII character can be represented in 1 byte using Serial RS232 and an example is the character 'K' which has the following 'waveform'.

The other communication protocol being used is NEC, which was invented by a popular Japanese television company to use for their infrared remotes. NEC is a

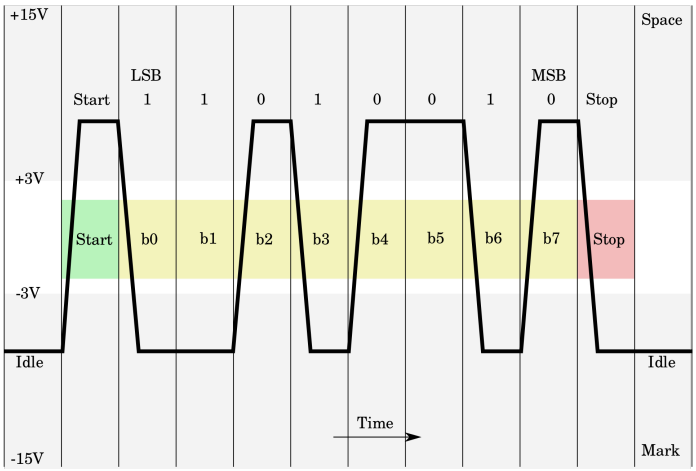


Figure 2.4 – Serial RS-232 Data Packet

pulse modulated waveform that operates strictly at 38 kHz carrier frequency, and was chosen as the modulation method because modulated waveforms are less likely to be affected by environmental light sources. NEC works by sending bytes in very specific timings which require careful configuration using interrupts to operate. Similar to serial, it is a sequence of logical 0's an 1's to build up a byte but is significantly slower due to the variable delays between transmissions. Each message consists of a:

1. Lead Pulse (9ms)
2. Space (4.5ms)
3. 8-bit Address of Receiver
4. 8-bit Inverse Address of Receiver
5. 8-bit Data
6. 8-bit Inverse Data
7. End of Frame Pulse
8. End of Transmission Pulse

As you can see, each message is significantly longer but at the same time much less prone to error due to the sheer amount of 'error proofing' done during transmission.

Unlike Serial which works on a handshake principle, the 8-bit address of the designated IR receiver must be hard coded into the transmission side such that only that receiver is able to receive that specific signal. On top of this, it sends the logical inverse of the receiver address in bits to confirm that specific receiver is indeed the one communicating with. Following this is the Data and Logical inverse of the data also being transmitted in case there is even a minute chance of error, the logical inverse is backup to prevent any data corruption at the receiver end. While testing this method with numerics and strings, we did not receive a single error at all. However, timing was an issue for image transmission and it resulted in a corrupted reconstruction. The image was not completely reconstructed in order due to the transmission speed being significantly higher than the receive speed due to the much longer transmission time of NEC. As shown below, each NEC byte takes a minimum of 108 ms (for binary 00000000) when including the message frame pulse, and double that time for binary 11111111. However NEC has the interesting way of reducing consecutive byte transmission times by introducing a 'repeat' code signal which is much shorter, taking only 11.8125 ms rather than 67.5ms per byte to tell the receiver to re-read the same byte.

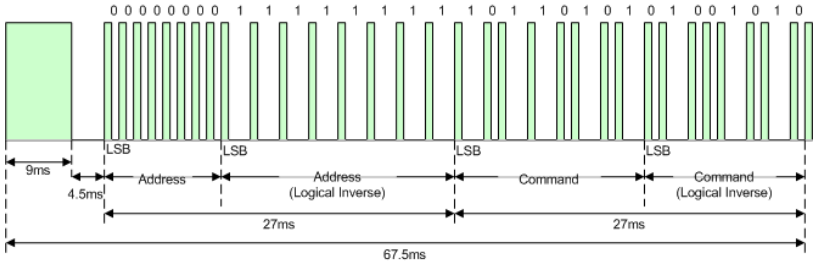


Figure 2.5 – NEC Framing Example

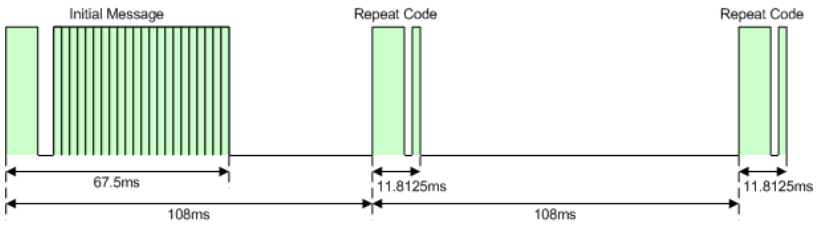


Figure 2.6 – NEC Repeat Code Example

2.9.1 Assumptions Made

- COTS Electronics components from Jaycar can fulfill the transmission requirements
- The transmitter will be able to focus and align perfectly onto the receiver during operation
- The internal clock timer on NEC transmission using interrupts is exact
- The transmitter will exactly turn on and off to signify bits according to the internal timer without any parasitic capacitance in the circuit and components slowing it down.

2.9.2 Constraints on Transmitter Performance

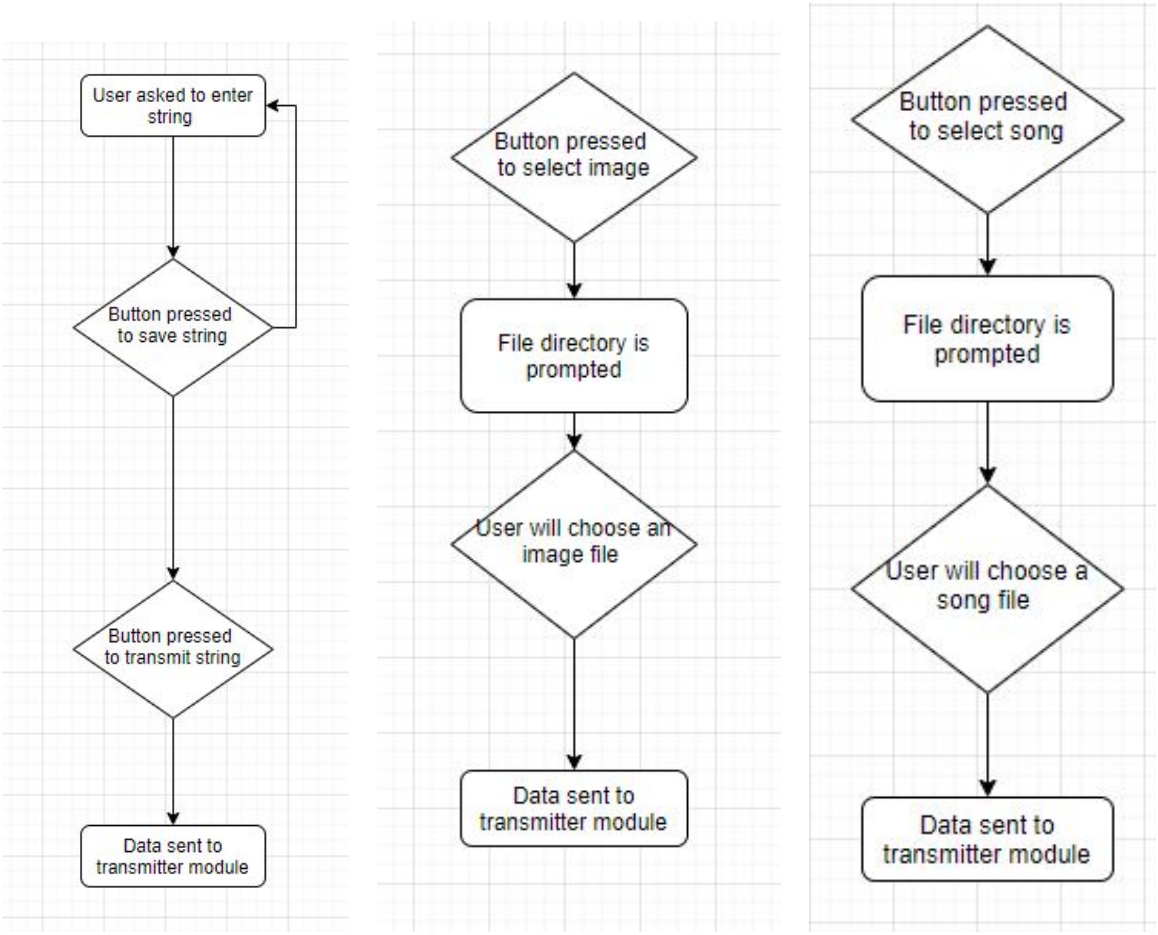
- Size of laser diode's beam. The width of it makes it hard to align with the receiver, especially at larger distances when millimeter displacements cause the beam to deviate multiple meters.
- Laser diode's power is limited to 5mW from the COTS components. With a larger power laser, it would be able to transmit much further as it would not lose intensity and signal strength.

2.10 Conceptual Design: User Interface

The user interface for this project will provide the ability for a user to enter 3 forms of data:

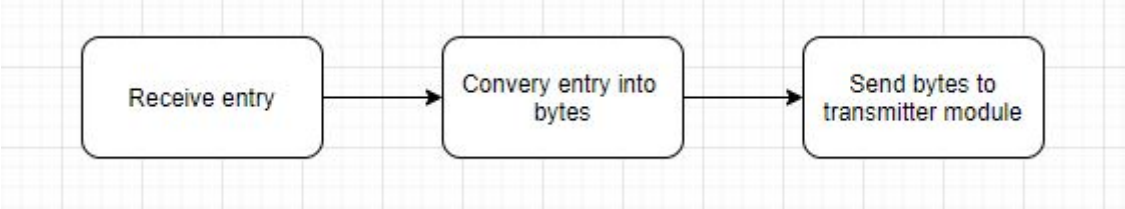
- String/Numerics
- Image
- Song

A computer will be used to run the user interface and will be connected to the transmitter module via a RS-232 cable. On the computer the user interface will prompt the user with many options, the flow of which the user will use the interface is as follows:



(a) User flow for string **(b)** User flow for image **(c)** User flow for song

From this the user interface will accordingly convert entries into a suitable byte form and will send to the transmitter module as bytes, the data flow for each type is as follows:



This process is relatively simple for string entries and more complex for images and songs, this is due to strings having a simple ASCII to binary conversions, however for songs and images a more complex process is needed which is further explored in Chapter 3.

2.10.1 Assumptions Made

- Availability of an IDE with required packages installed.
- Image and song files on local storage.
- COM port is correctly established by the user.

2.10.2 Constraints on Performance

- Very large images will take very long to read and send, as it needs to be read pixel by pixel and converted then sent.
- Similar to images songs will take time to send, this is due to songs having many framed that need to be read and converted into bytes for transmission.
- Sending speed is limited to the selected baud rate.

Chapter 3

User Interface

3.1 Classes of User

The user interface currently can be run on a PC and in future extended to a mobile phone or other application domains. To swap between these, a separate application must be run.

The primary user that we targeted was the general public, as the primary use case is a COVID check-in app for a phone, so the requirements for the targeted user is just between a mobile phone and a business receiver. This is a specific use case.

The testing GUI allows a developer, technical or trial user to select various types of data to send and when to send it. It shows the entire functionality of the laser communication system.

3.2 Interface Design: General Public

The interface was designed with the target audience in mind, due to this we have opted to designing a GUI that looks and feels seamless on a mobile device. A conceptual example of this GUI is shown in Fig. 3.1 below.

Figure 3.1 – Mobile GUI Concept

As you can see it has the essential components needed to a COVID check-in application, it has a function to check-in, check-out and also updating personal details. Check in and out details would transmit a collection of data including:

- An identification string which would user specific.
- Check-in/out time.
- Full name.

For details into how the application would send the data, inputs and outputs, or error trapping, see the following, Section 3.3, as the GUI for the general public only requires specific limited functionality of the developer version.

3.3 Interface Design: Developers

The GUI used in the testing and main functionality of this project is far from what a developed product would look like. We can see a much less visually pleasing approach was taken, and was substituted with functionality, see Fig. 3.2 below.

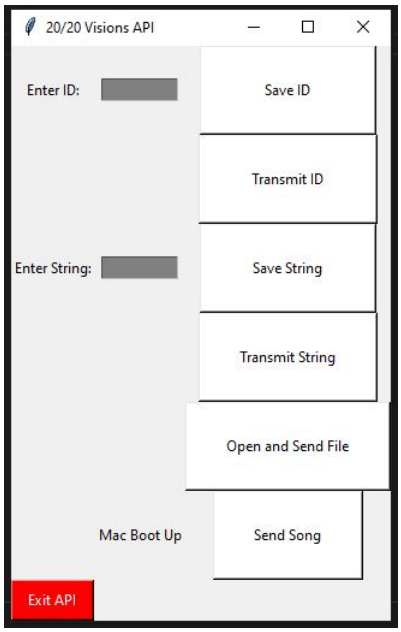


Figure 3.2 – PC Testing GUI

Here we can see an efficient GUI that allows a developer to input and correctly send controlled data on command. To effectively test each ID, string, Images and song files.

3.3.1 User Inputs and Outputs

ID and String

Inputs: In these sections the GUI will prompt the user to enter a series of characters to transmit, characters include:

- Standard A-Z.
- Number characters 0-9.
- Spaces.
- Special characters.

Outputs: The inputted data is saved into a string variable via the Save button, and transmitted via the Transmit button for both ID as well as String options. The python script takes the saved string and will send character by character via serial transmission to the Arduino. The data sent is a byte containing the ASCII value of each character represented in its binary form. The Arduino will collect byte by byte until the transmission has completed, and processed collected data accordingly.

Image

Inputs: In this sections the GUI will prompt the user to select an image file from local device's file directory, files that were tested were only BMP's, however the ideal use would be compatibility with all forms of image formats.

Outputs: When the user selects a file, they are saving the path to that file into a string, the GUI will then locate and make an image type from the saved path and will collect the pixels, from there it will go through each pixel in the image converting it form RGB to a hex number, for example red is FF0000 where the first 2 digits represent the level of red from 0-255 or rather 00-FF in hex the next 2 are green levels and finally the last 2 levels are the blue levels. The hex number is then sent in a similar manner as the ID/String where each character is sent as a byte of its ASCII number in binary. The Arduino will collect byte by byte until the transmission has completed, and processed collected data accordingly.

Song

Inputs: In this sections the GUI will prompt the user to select an song file from local device's file directory, files that were tested were only WAV files, however the ideal use would be compatibility with all forms of audio data.

Outputs: When the user selects a file, they are saving the path to that file into a string, the GUI will then locate and make an wave type variable from the saved path and will collect the number of frames that the WAV file has, from there it will go

through each frame in the file and send the data within the frame as a byte. The Arduino will collect byte by byte until the transmission has completed, and processed collected data accordingly.

3.3.2 Input Validation and Error Trapping

The way the GUI has been set up naturally has safeguarded input, looking at each form of transmission data:

- ID/String - every form of character is acceptable as every possible character entry has a recognisable ASCII number associated to it, so the GUI will read it and send it no matter the entry, this is how it can handle spaces, letters, numbers, and special characters with ease.
- Images and Song flies - when the user attempts to select a file weather its a song selection or an image selection the GUI will prompt the user with a file directory which represents the users local storage, from there we can manually choose the data types made visible within the code, for example a user wants to select an image and has a range of files including JPG, png and BMP, the code will only make visible types *.bmp, this says show all files that have * name meaning any name and of type .bmp, fo files with .png or .JPG will not be made visible, limiting the user and removing the possibility to select unaccepted files. The same implementation is set for the song selection in the form of *.wav.

With this in mind, there is no option available for a user to select an invalid or unaccepted data type.

Chapter 4

Hardware Design

4.1 Scope of the Laser Communication System Hardware

The following hardware design applies to the physical electronic circuitry used in the laser communication system. This includes interfaces between physical components and their individual purpose. Primary hardware components are split into two sides as seen below, the transmitter and receiver. For overall system reference, see Fig. 1.1.

Included in this section are references to research that rationalises design choices and data sheets that explain the operation of physical components. For more detailed information on the hardware, these references should be read.

4.2 Hardware Design

4.2.1 Power Supply

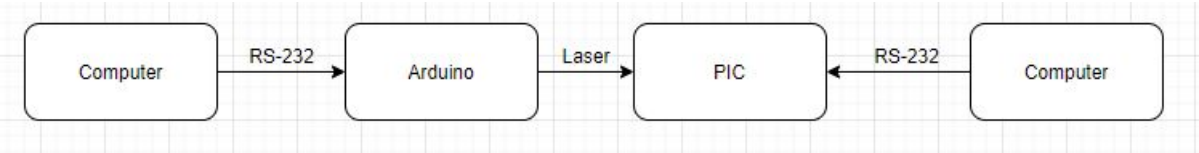
For the project, power supply was necessary to be small, lightweight, portable and fit the specifications of the micro-controllers used. Thus, standard 9V batteries were

chosen to power both the Arduino and PIC18F452 micro-controllers when not connected to the PC. Note that when using the 3.5mm DC power plug jack, the internal section must be wired to the positive terminal of the battery to avoid short circuiting the micro-controllers with reverse voltage.

For the PCs used and if deployed on a mobile phone for commercial use in accordance with use case (see Section 2.2), the internal standard on-board batteries will be used for power supply.

4.2.2 Computer Design

The overall computer-interfacing hardware design for this project can be visualised by the following:



Computer: As seen above a computer with user interface is connected to the Arduino via a RS-232 cable.

Arduino: As seen the Arduino which is responsible for transmission, has a laser attached to it and will transmit data from the user interface to PIC via one of the laser protocols through the medium of air.

PIC: The PIC has a receiver attached and will collect and process the data received from the laser communication sent by the Arduino and will send the received data to the final computer via a RS-232 cable.

4.2.3 Receiver Hardware

For the receiver hardware, the design consists of a photodiode as the primary sensor, allowing for light intensity to be converted to a signal ready to be received by the PIC micro-controller.

Once read by the microcontroller, the data is captured using software as per the defined communication protocol and it can be output in an appropriate form. For testing, the data was sent to a PC via an RS-232 link over USB Type-A cable with a MAX232 FTDI chip to do the conversion from TTL logic to RS-232.

Serial Receiver The photodiode [4] is connected to a large resistance circuit and power as seen in Fig. 4.1. A photodiode typically allows a small dark current (5nA) to pass through from the cathode to the anode of the diode (from 5V to V_- in the Figure). But, when irradiate with light, this reverse current can increase and produces a voltage on the resistor. The photodiode produces a current curve proportional to the intensity of light shone onto the receiver, where approximately 40μA is typical of 0.1mW/cm^2 of light (see [5] for more information about how photodiodes operate). Hence, the voltage at V_- can be used to determine the intensity of light on the receiver. Yet, due to only a very small current being produced, an op-amp is used to isolate the receiver from the micro-controller, boost the current output and acts as a 1-bit analogue-to-digital converter (see [6] for uses of op-amps as such functions). To boost current, the op-amp is placed in a circuit to act as a comparator, increasing the output to the maximum limits of the sign of the difference in input voltages (see Eqn. 4.1). As seen in Fig. 4.1, a reference voltage of 1.67V is used on V_+, after calibration and tests with the photodiode, meaning that whenever the V_- voltage is above 1.67V, the micro-controller will read a logical one on a digital input.

$$V_{out} = A(V_+ - V_-), \tag{4.1}$$

where A = 100V/mV large signal voltage gain for the op-amp used [7]. The physical setup can be seen in Fig. 4.2.

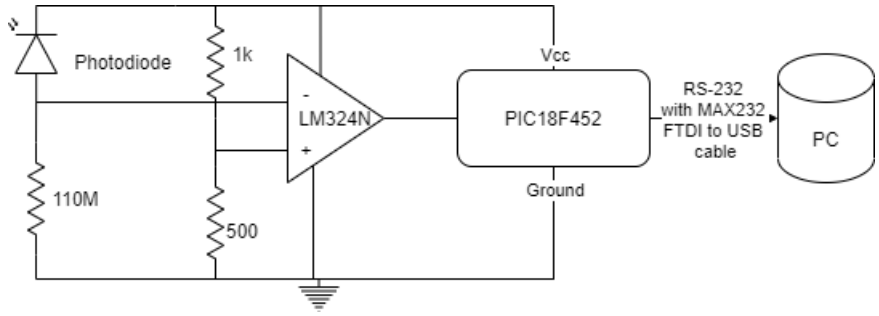


Figure 4.1 – Serial Receiver Hardware Diagram

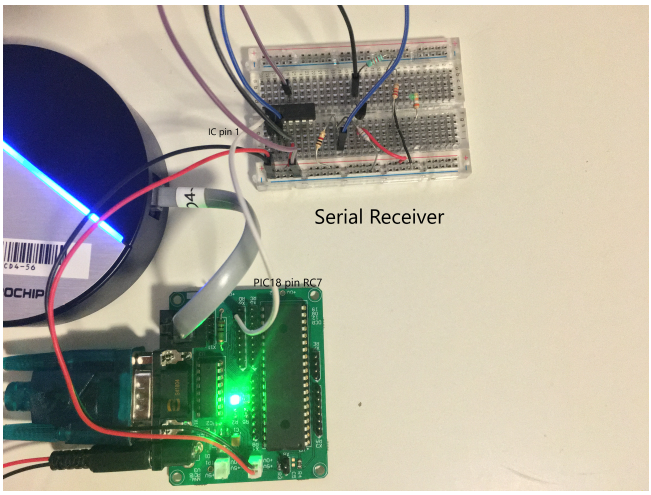


Figure 4.2 – Serial Receiver Physical Hardware Setup

NEC Receiver The photodiode is connected in a pre-built COTS hardware module [8, 9] which demodulates a carrier at 38kHz and is typically used as a receiver for infrared NEC signals used commercially for TV remotes. A diagram of the internal circuitry can be seen in Fig. 4.3 and a diagram of it's interface with the PIC18 microcontroller can be seen in Fig. 4.4. The physical setup can be seen in Fig. 4.5.

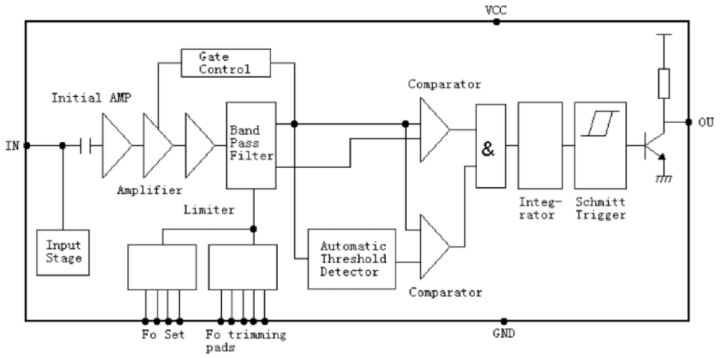


Figure 4.3 – NEC Receiver Hardware Internal Schematic [1]

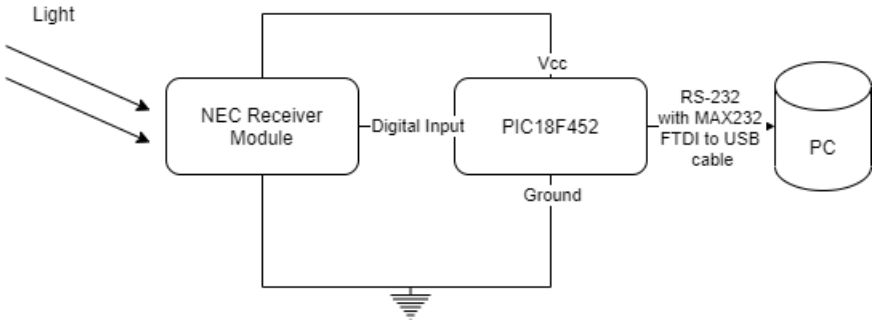


Figure 4.4 – NEC Receiver Hardware Diagram

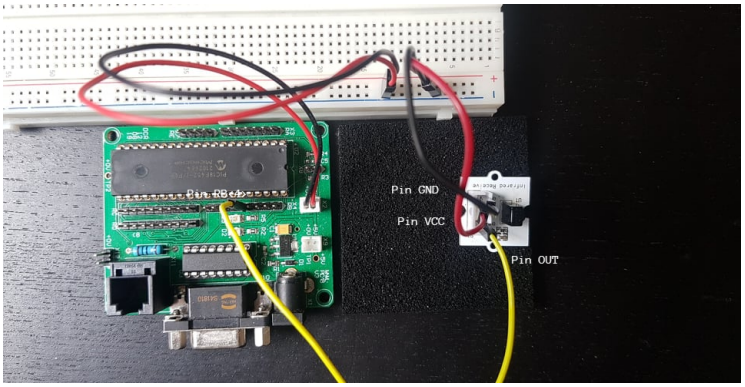


Figure 4.5 – NEC Receiver Physical Hardware Setup

4.2.4 Transmitter Hardware

For the transmitter hardware, the design consists of an incoming data stream (which can be in various forms) which is modulated according to the chosen protocol and

these electrical signals are converted to light using the laser diode module, seen in Fig. 4.6.

For testing purposes, the PC is used to transmit data over a standard USB Type-A to USB Type-B cable. This cable uses RS-232 protocol and is provided standard for serial communication with an Arduino micro-controller, the primary method of uploading software to the micro-controller and parsing serial data. The Arduino micro-controller will modulate a digital pin according to the defined protocol by the software uploaded, putting the data required to be sent on a form of carrier wave.

Red Light Laser This digital pin is the connected with a wire to the anode of the red light laser diode [10], and grounded on the cathode side. The laser diode module requires a 5V power signal only, with internal regulators to provide the required current for maximum power output (as these are proportional, see [11] for more information about how laser diodes operate). Thus, the laser diode will brightly flash with light according the modulation determined by the Arduino digital pin.

InfraRed LED Similarly, for InfraRed light transmission the digital pin is connected with a wire to the anode of the IR LED [12], and grounded on the cathode side. The IR LED module requires a 5V power signal to turn on. Thus, the IR LED will flash according to modulation. NOTE: The IR LED module does not contain internal power regulators or current-limiting resistors and can burn out if not connected to the correct pins on the Arduino due to current overload. This only operates correctly due to current sourcing limits of Arduino digital pins.

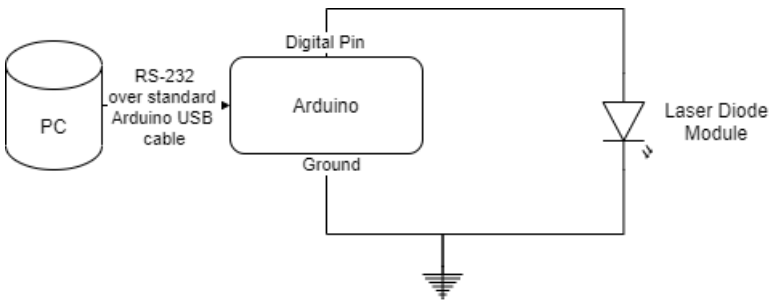


Figure 4.6 – General Transmitter Hardware Diagram

4.2.5 Operator Input Hardware

The only input hardware that was required for this program is a mouse and keyboard, in terms of string entry, this is from the keyboard. All buttons on the GUI are pressed using the mouse, including save buttons, transmit buttons, as well as song and image selection buttons.

4.2.6 Operator Output Hardware

When looking at string output, the received output is can be visually seen in PUTTY, as once it is received it is sent there, with NEC it is slower and accurate, however with serial it is very inaccurate and fast.
For images the received data is sent to a processing program that will receive the hex colors and convert and reconstruct the image, in this case NEC proved inaccurate because it was slower the timing was out of sync with the processing program and caused errors, with serial it was relatively accurate however still slow as the transmission was much longer due many pixels as well as the fact that 1 pixel was sent as 6 characters making up a hex color.
For sound output, the PCM signal can be played on any traditional passive speaker. It was tested both on a buzzer [13] and a speaker [14], with higher volume on the buzzer, but better sound quality on the speaker.

4.2.7 Hardware Quality Assurance

In the process of testing, we made many measures to ensure reliability and accuracy, they are as follows:

- Securing modules to fixtures - It is very difficult to ensure a narrow laser would remain in an even smaller receiver, so to combat this we secured both modules to a stool/chair to allow consistency as holding it seems almost impossible to have accuracy.

- Using lenses - At longer ranges a lens was used to collect more of the laser as it widens with the distance and focus it to a point on the receiver, this allowed us to go much further then with just laser and receiver.

4.3 Hardware Validation

- LED indicator - we utilised the on board LED on the Arduino and set it to flash if it received anything, this was used to ensure there wasn't any errors between receiver and computer, and if there were we could visually see where it would stop working.
- Serial monitor - we used the serial monitor to visualise the data that was being sent to ensure it was the correct data, this prevented confusion on weather the data received was correct or not.

4.4 Hardware Calibration Procedures

Only two hardware calibration processes were done during this project as it was largely software dependent. The first being alignment of the laser with the receiver every-time a test was performed. At larger distances, if the alignment is even 1mm off, it resulted in 5 to 8 m deviations at the receiver side, which would mean that the signal would not even be received. At smaller distances, this was also done to ensure validity with our experiments. The second component that required calibration was the focusing lens on the receiver side. It had to be situated at the perfect focal length at approximate 3 to 4 cm away from the receiver to have the maximum intensity and smallest beam.

Chapter 5

Software Design

The software design section will provide the high-level design approach undertaken within this project. System architecture and the individual system modular design will be covered, detailing how each module adhered to outline system requirements.

5.1 Software Design Process

In terms of software design for the overall process, initially a top-down design cycle was employed alongside a brief software requirements analysis (SRA). The SRA detailed the specifications which the system must adhere to and fulfill. Once the SRA was conducted, overall functionality was planned out with initial modular design made first to be later expanded upon in detail. As the overall system was split into many software modules, split across different microcontrollers and IDE's, there was no need to introduce Object Oriented Design into the system.

A Software Design Life Cycle (SDLC) was introduced for the overall system design process. An iterative Waterfall model was used to design both the overall system and individual modules. The approach involved first setting requirements, system design, implementation, testing and then finally integration and iteration.

The Waterfall SDLC is adept at handling smaller scale projects with fixed and known

requirements. Waterfall development has advantages that allow for better departmentalization and control throughout the project and has faces disadvantages due to the approach allowing for little revision and reflection throughout the design process. In more complex projects where modules are more dependant and less compartmentalised - OOP heavy, or when requirements shift throughout the project as beholden to a client, the Waterfall SDLC runs into issues. Given the circumstances of this project - heavy modularity with set requirements, the Waterfall model was suitable as the principle SDLC. .

5.1.1 Software Development Environment

String Transmission For String and Numeric transmission, there were two main modules - the Transmission side and the Reciever side. Each main module was developed independently on a separate microcontroller, the Arduino for the Transmission side and the PIC18 for the Receiver side. As a result, two independent Software Development Environments (SDE) were employed.

The Arduino side and the Transmission side respectively employed Arduino IDE and the MPLABX IDE. Both Serial and NEC protocol - which are two independent systems under ID and String Transmission employed the same IDE's for their respective sub-modules, Transmission and Receive.

Sound Transmission As above for string transmission, with additional timing requirements implying that only fast serial communication (usually 115,200 baud rate) can be used.

Image Transmission As mentioned earlier, the transmission breakdown of an image into an array of pixels to be sent over serial and the reconstruction of individual pixels into an array and therefore into an image was handled by Processing 4.0. However the actual transmission between the two ends was handled by 2 Arduinos communicating. Unfortunately due to time constraints, the receiver Arduino code

was unable to be refactored for use in the PIC Microcontroller in time. However the code running on Arduino is a simple event based Serial or NEC transmission through a laser diode which would then be received (and decoded if NEC) and then reconstructed on the receiver side Processing 4.0 program.

5.1.2 Software Implementation Stages and Test Plans

The Software Implementation Stages and Test Plans follow closely to the chosen SDLC, the Watermfall Model.

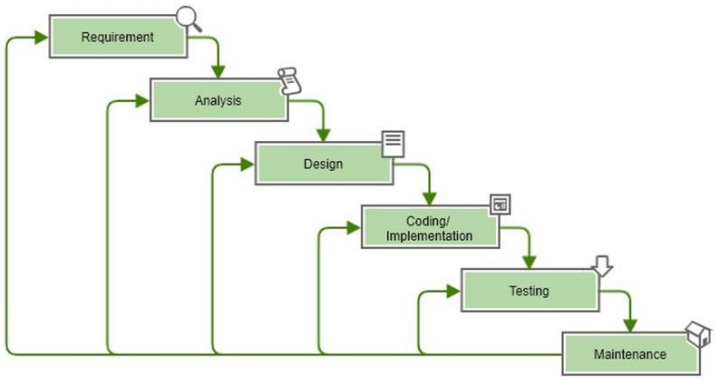


Figure 5.1 – Waterfall Model SDLC

The Software Implementation Stages follow the middle 1/3 of the Waterfall model. Requirements analysis and System Design is to be completed before the beginning of Implementation, following the process outlined in Sections 5.1 and 5.3 of this document.

A general Implementation process will be outlined, however each individual module will face slightly differently implementation processes due to the nature of the module. The general Implementation process consist of a 5-stage model:

1. Planning,
2. Process Design,
3. Solutions Design,

4. Configuration and Customization,

5. Integration.

Testing follows after the Implementation process is complete. A general Test Plan is drawn up for each of the main modules - ID and String Transmission, Image Transmission and Sound Transmission. Within each Test Plan, the Scope, Assumptions, a Schedule, Roles and Responsibilities and the Environment will need to be clearly defined to allow for a smooth testing process.

Actual Testing methodologies consist of a mix of functional and non-functional methodologies. Given that this project is limited in scope and also involves the usage of hardware as well as software, the majority of the software testing methodologies will focus on the function side. If possible, for each module - testing should go through the process of

1. Unit Testing,

2. Integration Testing,

3. System Testing.

Typically Acceptance testing will need to be performed as well, however within this project, this will fall under Section 6. System Performance, specifically Performance Testing where the majority of the non-function testing methodologies will also be carried out.

Image Transmission As the Image Data transmission is a lot more visual and takes a greater processing time due to the nature of the image deconstruction, transmission and reconstruction, it needs to be tested and verified independently. First, we aim to implement Unit Testing on each individual submodule of the Image Transmission process. Although the program is largely functional rather than object oriented, it can still be tested in isolation using automated tests to send multiple images across

to serial and verifying the hex code manually. This process must be repeated on the other receiving end as well, where the conversion from HEX ASCII to colour pixels can be tested in isolation by manually inputting values into the Serial COM Port interface in ARDUINO or sending dummy character arrays. Following Unit Testing comes Integration Testing between both ends of the Image Transmission process. Testing should often occur till the program ceases to work or there are sufficient accumulated errors in the pixel reconstruction. In this case, to test the robustness of the system, a variety of image formats were sent over (jpegs, bmp, png) of different sizes to test its consistency. The same integration will have to be performed when testing different protocols and different light types. Finally, the overall system requirements have to be tested and validated to prove the efficacy of the proposed system. The system will be tested over 50m in daylight, artificial light and at night. It will be transmitting images from a GUI and have a transmission time of less than 5 minutes for a 240 x 240 RGBA image.

5.2 Software Quality Assurance

A variety of general Software Quality Assurance techniques are to be employed throughout the software development process. During the coding process, revision control and documentation in the form of comments were made as development occurs with frequent reviews and revision of code.

A centralised GitHub was used for revision control, allowing each member of the group to working individually or collaborate on different modules with greater ease. Frequent Commits to the central GitHub also served as a form of higher-level documentation for the entire system, whilst reducing duplication and providing traceability for each change made.

Testing and validation was also completed was also conducted, with the general software-side testing utilising functional methodologies such as Unit Testing, Integration Testing and System Testing in order to better improve code quality.

5.3 Software Design Architecture

5.3.1 Architecture

System Decomposition From a high level, the system architecture and flow of control is simple. There is next to no feedback back to the user's side in the prototype model, but plans to expand to this for future iterations of the design. As shown in the process flow diagram below, it is quite a linear process. And each transmission cycle will continue to repeat until a stop command is sent or another new input is set into the buffer.

From the Flow Diagram, the system can be decomposed into its major components with each the branch of the flow diagram in terms of the transmitted format acting as a major component. As String and Numeric data transmission is essentially the same format within the coding language, these two branches will be combined and developed as one individual major component. The major components are therefore the 1. String and Numeric Transmission, 2. Image Transmission, 3. Sound Transmission.

Modular Decomposition and Functional Allocation Each major component can be further divided into modules, specifically the Transmission and Recieve module. The flow diagram contains a dotted line which visually divides the modules. Functional responsibilities are allocated to the individual modules as visualised by the flow diagram. The Transmission module would generally be responsible for inputting and transmitting whereas the Receiver module would be responsible for receiving, decoding, ordering and outputting.

Component Communication and Interaction As the flow diagram indicates, each major component is largely independent of one another, with the specific component of choice's Transmission module having to be manually pre-loaded onto the hardware as the decision is made on the GUI for the specific type of data transmission. Within each major component, the module interaction is very linear, with a top to bottom approach following the flow diagram exactly.

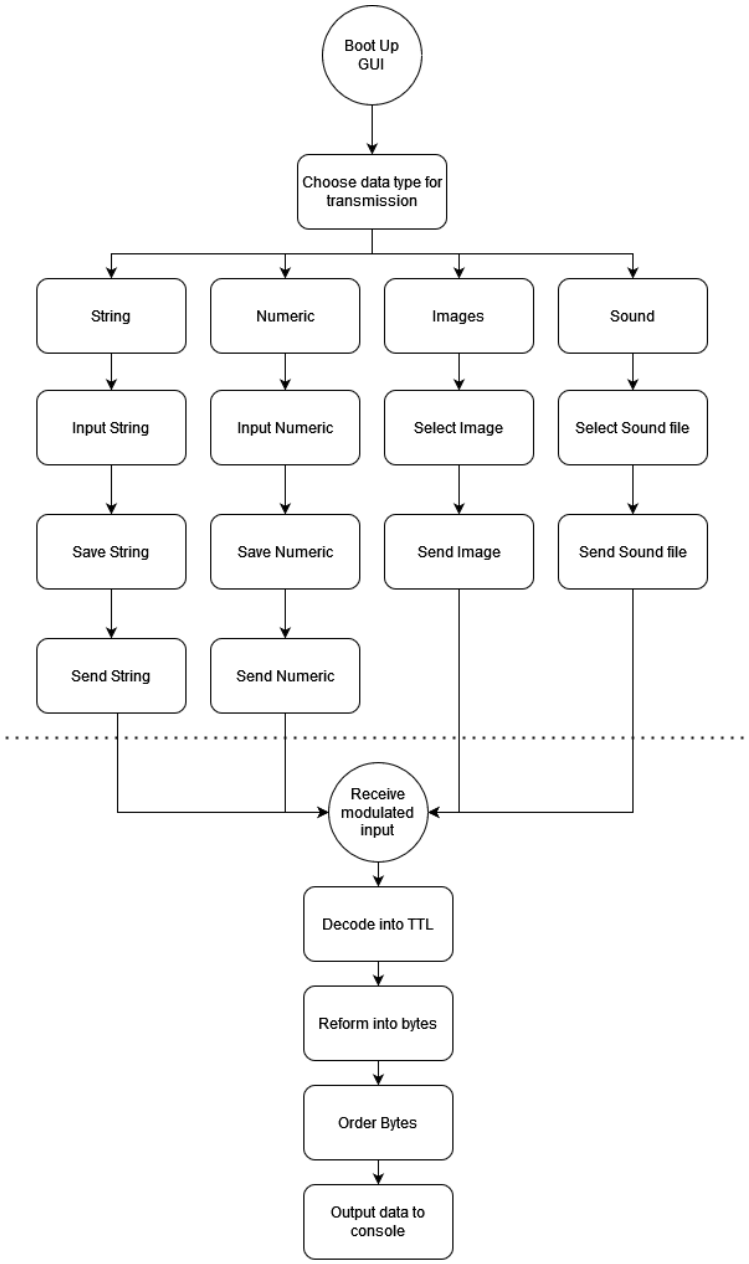


Figure 5.2 – Flow Diagram

5.3.2 Software Interface

The two software modules, "Transmission" and "Receiver" will have a public interface to both hardware and to a user. Starting first with the "Transmission" module, it is interfaced with both the GUI - a user initiated and controlled program, and with the transmission circuitry which involves laser diodes and IR diodes. The "Receiver"

module is interfaced with the receiver end circuitry, which involves a IR receiver component, analog to digital conversion, amplification and filtering circuitry. As well as the output to console with the second user (the recipient) is able to view. It is also important to note that since the majority of the code is functional based rather than Object Oriented Design, there is no need for private, protected and public interactions between code segments.

5.3.3 Software Components

This section contains a detailed view of the internal workings of each of the software modules. See the Code Appendix, Section A for more details.

String and Numeric Communication

Serial Protocol For the software to implement serial protocol transmission, the in-built Arduino serial library is used. This allows for specified baud rates and custom data packets, in accordance with needs. Low-level work is controlled by the library, with only one command required in a loop, being Serial.write(BYTE). Connecting the laser to pin 1 on the Arduino will cause the message to be sent.

For the receiver, the opposite is done, using the USART module of the PIC to receive text on pin RC7. It is configured with baud rates equivalent to the Arduino sending and stores one byte at a time in a circular buffer. Then, it is output to the PC through the USART module or some other peripheral as required.

NEC Protocol For the software to implement NEC protocol transmission, a similar approach is taken to serial, but required to have a custom definition due to no available libraries. This, though, is a pre-existing problem hence others have software implementations of the protocol that can be utilised. Delays with Timer 1 are used to determine timing of signal changes and the software toggles the output voltage on pin 2 to connect to the laser.

For the receiver, pin RB4 is used on the PIC to indicate an interrupt on change whenever the signal changes. Timers are used to determine whether the signal received is as expected by the protocol, and if the timing is not met, the process is restarted. If it is met, the signal is read one bit at a time into a buffer. When the message is complete, it is broken into components and checked to ensure correct, the output to peripherals as desired.

Sound Communication

For sound, the software is based on the concept of Pulse Code Modulation (PCM) [15], which is a method of digitally representing sampled analog signals commonly used for audio (see Fig. 5.3). For this system, a playback or sampling rate of 5kHz is chosen with 8-bit resolution and the PWM signal that plays the audio is performed at a frequency of 39kHz (see [16] for an example with Arduino). These frequencies are both limited by the speed of the micro-controller's clock, yet the higher they are, the higher the audio quality in a general case.

For this PCM method, each small section of audio is represented as an 8-bit number which is used to determine the duty cycle of the PWM signal played to the speaker. Files such as '.wav' are typically in this format already, so can be read from a computer with the GUI and dumped to the Arduino through serial.

For software, the transmitter side is only required to transmit the 8-bit numbers at a speed of at least 5kHz, which implies a baud rate of at least 40000 is required to avoid distortion in the audio. This done through outputting to the serial port with specified baud rate.

For the receiver side implemented on the PIC18 micro-controller, a serial input is taken in using the USART module configured at the baud rate of the sender. This baud rate is configurable, and tests have been done with 9600, 62500 and 110000 baud rates. The PCM audio playback is implemented using two timers. Details of exact setup can be found in code comments. Timer 1 is set up as a special event mode interrupt trigger occurring at a frequency of 5kHz using CCP2 on pin RC1. When

triggered, the software will write the next value in the string of received sound bytes to the duty cycle register of CCP1. Timer 2 is set as a PWM signal of 39kHz using CCP1 on pin RC2. This pin is directly connected to the speaker, thus producing audio based on a PWM signal that changes at a rate of 5kHz. Hence, sound communication software is complete.

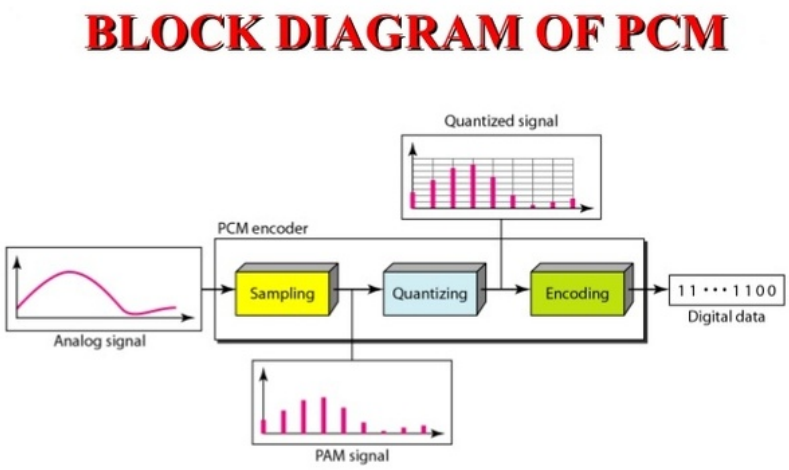


Figure 5.3 – PCM Block Diagram [2]

Image Communication

Image transmission involves loading an image up and breaking it down into byte by byte such that it sent as an array for reconstruction on the receiver side. This is done easily in Processing 4.0 which is largely designed in Java for graphical use-cases. Image transmissions can be done with PNGs, JPEGs, BMPs and TIFF file types. Typically, BMP files are the most structured image datatype in the sense they contain a bmp file header, pixel array and color profile. So sending image data over needs to be exactly in the same order. However, when working with Processing 4.0, it handles this image breakdown using the pixels() command to create a matrix of pixel values (32 bits per location) which signify the RGBA colour values. This just means that the pixels[] array needs to be extracted to give the derived variable type 'color'. However it is

not possible to send 'color' data types across serial is not possible, so it needs to be further transformed into a string. Hence we use the transformation function hex() to convert the color() pixel values into String ASCII hexcode (ie: FF0000) which can then be sent over serial like a string. On the receiver end, the same process needs to be done in reverse where the received String ASCII hexcode needs to be tranformed into color() values using the unhex() function. Then by using the updatePixels() keyword and a pre-initialised pixelsNew[] array, it is possible to reconstruct the pixels array on the receiver end using the information from the color() values. The pixels array can just be printed onto the screen using the draw() function. This will result in the image being rastered line by line similarly to old CRT televisions. However, this means that information about the image dimensions needs to be sent or known before hand such that the draw() function can iterate to the next row once the pixels for that specific row is completed, ie: for a 120 x 120 px image, it should iterate to the second coloumn once it has read and reconstructed 120 pixels.

5.4 Preconditions for Software

5.4.1 Preconditions for System Startup

As the system is hardware based, the pre-conditions for system start-up involve correctly setting up the hardware for each respective major component. The entire system will be primarily based on two Microcontrollers for the purpose of laser communication - any Arduino board and a PIC18. These two controllers will largely remain configured and connected the same across the different transmission protocols.

Arduino Configuration

The Arduino will be connected to both the PC which contains the GUI via a USB Type-A to Type-B cable. This multipurpose cable will be used to load the correct

transmission code and serve as both serial communication for the transmission of data to the laser/IR transmitter and as the power source for the Arduino.

Upon successful connection of the USB Type-A to Type-B cable, the ON LED should turn on.

Specific pin configuration for each transmitter will be detailed below.

PIC18 Configuration

The PIC18 is more complex, requiring: The MPLAB ICD3/4 Debugger and its respective cabling, a RS232 IC, a Male RS232 to USB Type-A Cable and a 9V Power Source. Both the ICD and the RS232 cable will be connected to the PC. Respectively, the components are for loading software, enabling serial transmission to the PC, serial transmission to PC and powering the PIC18.

The order of connection is also important. 1: RS232 IC, 2: 9V Power Supply, 3: ICD3/4 Debugger, 4: RS232 to USB Type-A cable. When disassembling the system, it is also important that these steps are followed in reverse order as well.

Upon successful connection of all the cabling, the a green LED on the PIC18 will turn on. The ICD will also have LED's which indicate its current status, differing between the ICD3/4.

Specific pin configuration for each receiver will be detailed below.

Transmitter & Receiver Configuration

Specific pin configuration for each form of transmitter and receiver are necessary with only minor changes. Both electromagnetic transmission frequencies can transmit in Serial or NEC protocols given the appropriate receivers and transmitters. Primarily, the transmitters determine the type of light used and the receivers + software determine the protocol used. These can and should be altered and configured depending on the use case of the system.

Visible Light - Laser Diode Setup The Laser Diode employed was a 5mW, XC4490 Arduino Compatible Red Laser Diode Module which requires a 5V power source. To connect it to the Arduino, two male to female pin connectors are required. Connect labeled pin S on the Laser Module to the Digital pin 2 on the Arduino. Then connect labeled pin - on the Laser Module to any Ground.

Photo-diode Serial Receiver Setup The Serial Receiver module currently involves a photo-diode in series with an op amp circuit which is connected to the micro controller. First, connect up 5V and GND to the power bus on the receiver module from the PIC18 X2 or X9 terminals. Then, connect the output from the op amp (IC pin 1) to pin RX/RC<7> on the PIC18.

Infrared Light - IR LED Setup The IR transmitter consists of a XC4426 Arduino Compatible Infrared Transmitter Module connected to the PIC18. Use two male to female pin connectors, connecting labelled pin - on the IR module to GND on the Arduino and labelled pin S on the IR module to pin 2 of the Arduino for Serial, or Pin 3 for NEC.

NEC Receiver Setup The IR NEC Receiver used was a XC4583 Linker Infrared Receiver Module for Arduino which requires a 5V power source. To connect it to the PIC18, 3 female to female pin connectors are required. First connect up the VCC Pin on the IR Receiver to a 5V power supply, then connect the GND Pin on the IR Receiver to ground. Finally connect the OUT pin on the IR Receiver to RB<4> on the PIC18.

Software Set-up

There are 4 main pieces of software required for this system to operate.

1. A GUI is ran from the transmission sided PC which allows selection of transmission data, E.g. ID, String, Image or Sound. This software will be used live during operation of the system.

2. The transmission sided Arduino require it's associated software, also called Arduino to load the relevant projects. As we can transmit via both Serial and NEC protocols, two different projects are required. The Arduino software can preload the Arduino with the desired code.

3. The receiver sided PIC18 requires it's associated software, MPLAB X to load the relevant projects. Once again, a separate project and code will be required for either Serial receive or NEC receive. The MPLAB X can preload the PIC18 with the desired code.

4. PuTTY, a serial communication software will be ran on the receiver sided PC which displays the data received via laser communication. This software will be used live during operation of the system.

5.4.2 Preconditions for System Shutdown

There are no pre-conditions for System Shutdown, the GUI and PuTTY can be shutdown on demand without affecting the system. The Transmission module and the Receiver module for whichever transmission format will be loaded onto the microcontrollers, therefore system shutdown is equivalent to hardware power off. To correctly power off all hardware, follow the instructions for setting up the hardware in the reverse order.

Chapter 6

System Performance

6.1 Performance Testing

6.1.1 Module Testing

See the Code Appendix, Section A, for software used to test modules.

String and Numeric Communication

The software was designed using an iterative Waterfall approach where the requirements were set first, then the system was designed, implemented, tested and then integrated and iterated. The completion of this module would fulfill the bulk of the CSR whilst serving as the technological basis for the COVID Check-In operational use-case.

Transmission The Transmission side which was built in Arduino was first tested. The first test concerns the serial reception from the GUI to the Arduino. Both the Serial and NEC transmission protocol uses an event based method rather than polling to read the incoming signal from the GUI and send that across to another serial port. This was tested using the Serial Monitor in the Arduino IDE to send ASCII across one

serial port to trigger the event, and monitor the second serial port using PuTTy to verify that the output was the exact same. As the Serial Protocol directly operates in this manner, completion of this this test also verifies the Serial Transmission module. A second test was also conducted using the oscilloscope, seen in Fig. 6.1.

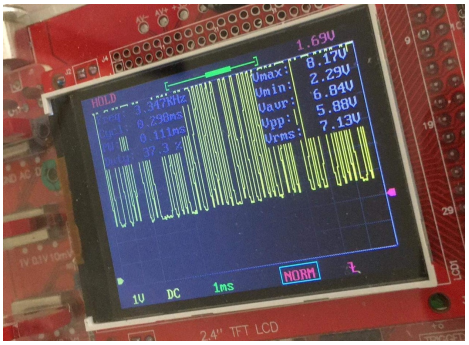


Figure 6.1 – Serial Test with Oscilloscope

Testing for the NEC protocol transmission involves using an oscilloscope to directly measure the output, see Fig. 6.2. As the NEC protocol transmits a very specific PWM signal as shown by the figure below, the output can be directly measured by the oscilloscope to check if it matches a PWM signal. Given the robustness of the protocol, it is safe to assume that the transmitted byte is correct if the signal output is correct, this can later be verified on the receiver side too. Testing was done according to the CSR, specifically CSR 4 which states that there must be a continuous 10 second transmission. Once all test cases are passed, it is safe to assume the that the Transmission module for both Serial and NEC Protocol is fully operational.

Receiver For the Serial Protocol sub-component, the software on the PIC18 was fundamentally identical to the Transmitter side on the Transmitter Arduino due to them both being just an event driven 'pipeline' for data to transfer across devices using the USART module. The same tests were done here as with the Transmitter Arduino. Additional testing was performed for the Receiver to terminal portion of the software which involved standard RS232 serial communication similar to the GUI to Transmission side. The same tests were performed with PuTTY as earlier in the Transmission side to verify serial communication is working, test cases employed

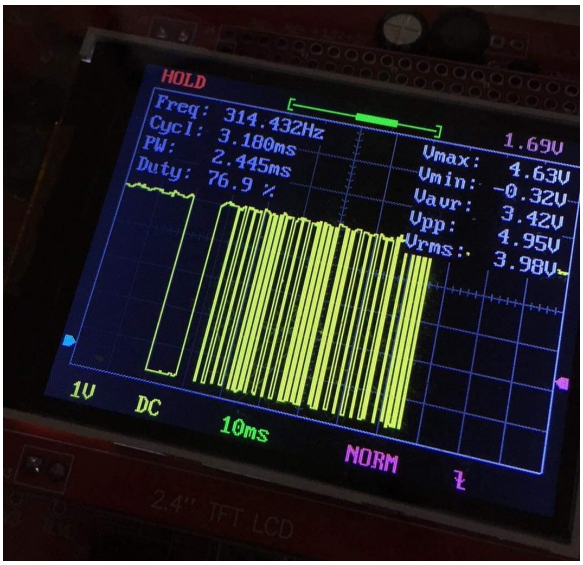


Figure 6.2 – NEC Test with Oscilloscope

manual input of strings and numerics. Once again testing was done according to CSR specifications.

The NEC protocol couldn't be tested until the integration phase as it required either a laser or IR transmission from the Transmission side as a NEC signal was unable to be replicated.

Integration Finally was the overall integration of both Transmission and Receiver sides. For Serial Protocol, initial testing was conducted using a wired connection from the Arduino's TX pins to PIC18's RX pins. A RS232 Cable connection was also completed on the PIC18 side with the terminal of the PC. No unexpected problems arise from the testing of the Serial Protocol as most of the tests were already previously conducted in individual module testing. However Serial Protocol itself was not robust and couldn't fulfill specifically CSR 4. This was attributed to the nature of the Serial Protocol, with the lack of modulation and demodulation, interference at longer ranges were extremely likely to occur due to the visible light, obstacles, etc.

NEC Protocol integration ran into many more issues. Individual unit tests on the receiver sided software ran into no issues during the Implementation phase, however upon integration, a series of problems arised involving the NEC signal. This was

slowly debugged using an oscilloscope which tested at which point the signal was being disrupted. It was discovered that the problem occured at the PIC18 pins as the received signal from the the photo-diode or the IR receiver was unproblematic. Two major issues specific to the PIC18 was determined to be the cause. As we were using RB<4:7> Interrupt on Change (IoC) to trigger the event, it was assumed that all 4 pins would be fine to use, however RB<5:7> pins are actually the associated ICD Debugger pins which made them unusable. As a result, specifically RB<4> had to be connected. The second major issue was that RB<4> pin has a pull-up effect which must be manually disabled. This effect would interfere with the precise timings required by the NEC protocol, ruining the signal received and outputting incorrect timings for each event that occurs within the software. Once these two major issues were resolved, the NEC Transmission and Reception system worked fine with further test cases satisfying all CSR except CSR 3: Daylight, artificial light and night time operation and CS 5: 50m transmission which required an independent test case.

Image Communication

The software was designed using an iterative Waterfall approach where the requirements were set first, then the system was designed, implemented, tested and then integrated and iterated. As this image transmission was an additional system requirement, it was not necessary towards our use-case, but a valuable technological demonstration nonetheless.

Transmission First the Transmission side of the code was done on the Transmitter Arduino, which uses an event based method rather than polling a serial port, to read and send that across to another serial port. This was tested using the Serial Monitor in the Arduino IDE to send ASCII across one serial port to trigger the event, and monitor the second serial port using PuTTy to verify that the output was the exact same.

Next the Transmission side of the Image Breakdown code was done on Processing 4.0. This stage involved testing the type of images that could be loaded - bmp files were

the most effective - and understanding the speed of transmission. In order to verify that the image breakdown into arrays from which each pixel was sent individually as a hex ASCII value, prior to connecting to serial, the void draw() method was used in Processing 4.0 to graphically verify that it was indeed splitting up the original image pixel by pixel. Naturally, the next logical progression was to integrate serial communication into this image breakdown process where the Transmitter Arduino was connected and initialised within the Image Breakdown code such that every void draw() loop, it would iterate and send the next pixel across. This was verified using the Serial Monitor to verify that the correct hex values for the image were being transmitted. For example, the image below was what was being tested, and could be verified simply by observing the hex values being read on the Serial Monitor. If it had a period of #000000 it meant it was transmitting the blacks, while if it was transmitting #D60000 it meant it was transmitting the reds, and if it was transmitting either #6F6700 or #F7A800 transmitting the greens and skin colour respectively.

Figure 6.3 – Test Image for Transmission

Receiver After that was the Receiver side of the code on the Receiver Arduino, which was identical to the Transmitter side on the Transmitter Arduino due to them both being just an event driven 'pipeline' for data to transfer across devices. The same tests were done here as with the Transmitter Arduino.

Following this was the creation of the Receiver side of the Image Reconstruction code

on Processing 4.0. This would continuously poll the serial port connected to the receiver Arduino (as that was already event driven) and every time it would receive an ASCII byte, it would store it into an array, unhex it into colour pixel values and draw it on the screen using the void draw() method. Testing this was a bit more difficult as there were issues due to synchronizing inputs manually from the Serial Monitor to the pixel being drawn up in Processing 4.0.

Integration Finally was the overall integration of both Transmission and Receiver sides. Initially it was tested using a wired connection from each Arduino's TX/RX pins. The Mario.bmp image shown above was the test piece and only resulted in 1 issue during integration. It was necessary to know the size and dimensions of the image before-hand such that the reconstruction side can create a carriage return and start on a new line on the reconstruction canvas it is drawing, otherwise the reconstructed image looked like a warped version of the original image. This could be fixed by two possible ways, sending the loaded image data's dimensions before sending any pixel data to store those in variables or send images always with fixed dimensions such that the reconstruction side code does not need to change. Colour-wise, there were no faults as the hex ASCII translated perfectly back into colour pixel data.

Expanding this however to use wireless Serial through pulsing light through a transmitter connected to the TX pin and receiver connected to the RX pin resulted in the image transmission being much slower than the wired. Furthermore, it was more prone to colour errors, sometimes putting in Greens where it should be Black. This could just be attributed to the lack of modulation done for image transmission, and the effect that ambient light has on this. In order to solve this, NEC pulse modulated protocol was used to send pixel data across, however it was incredibly slow (240 x 240 px image took 35 min) and was highly error prone due to the way that NEC works - just taking longer to process every byte of information, and as there are 4 bytes of information per pixel (Red, Green, Blue, Alpha), this would mean 4 times as longer to transmit and reconstruct the image. NEC also had timing / synchronicity issues where even though the image's size was fixed, it would sometimes miss pixels

or jump rows when that row hadn't been completed yet. These problems were yet to be solved at this point in time.

6.1.2 System Testing

The majority of the system testing occured within the integration testing phases of the numerous transmission types. The successful integration and completion of the String and Numeric transmission module automatically satisfies all CSR except for CSR 3 and 5 which required independent System Testing once all modules were implemented and integrated.

Independent System Testing involved outdoor testing with the entire system, this was performed in an open-space - a park with a clear 50m open space. Testing was performed during daylight and night time as well as introducing artificial interference in the forms of artificial light from streetlamps and torchlight. Results from the tests conducted were that both NEC and Serial Transmission protocols could reach beyond 50m under all test cases, satisfying CSR 3 and CSR 5. However only NEC Transmission could satisfy CSR 4, 10s of continuous accurate transmission under whilst passing the 50m demand. After this test was conducted with test cases satisfied, all CSR are satisfied by the String and Numeric system.

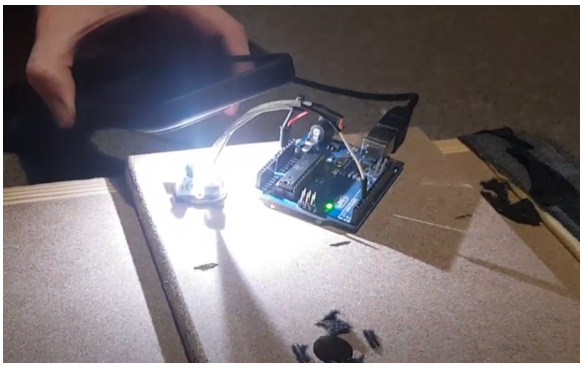


Figure 6.5 – Passing Test Cases under Artificial Torchlight

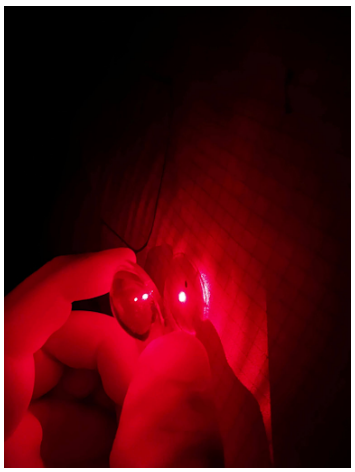
Figure 6.4 – Outdoor Night Range Test Demonstration

6.1.3 Miscellaneous Testing

A series of miscellaneous tests were further performed to demonstrate additional functionality and test the robustness of the system. Although these tests did not directly contribute to either the CSR or ASR, they serve as a proof of concepts and have given direction for future work for this system.

Optics were introduced into the system in an attempt to improve the range of the system and to potentially expand system functionality in the future. All experiments serve as a proof of concept and need further development. An optical focal lens - a convex lens was introduced to focus the laser light into a single point, drastically extending the range of only the laser diode. From experiments, range was extended roughly up to 40m beyond the initial maximum range however the focal lens was very finicky to use with the focal distance being very short - only about 3cm. A mirror was also introduced to bounce both IR and laser light, with standard test cases passed afterwards serving as a proof of concept. Laser transmission could also pass through translucent panes in testing serving to increase the robustness of the system. An optical fibre was also used to pass the laser through. Although the laser is no longer travelling through free space, this demonstrates potential for this system to bypass obstacles or see indoor use similar to current optical fibre communication.

(a) Unfocused Laser at 50m



(b) Focused Laser at 50m with Focal Lens

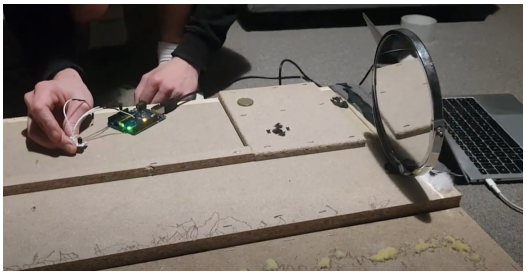


Figure 6.6 – Mirror Demonstration

Figure 6.7 – Translucent Pane Demonstration

6.1.4 System Performance

For overall system performance, as there were no set requirements by the Assignment brief, performance could only be judged by extraneously without further users to

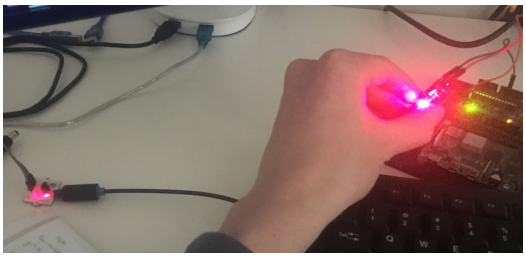


Figure 6.8 – Optical Fibre Demonstration

test the system. In terms of software performance, factors such as memory usage, loop time, etc. all conformed with standard software practices and the limits of our hardware - the Arduino Uno and the PIC18 and its corresponding IDE's.

System repeatably and ease of use is overall very high from both a development and prototype perspective as well as an operational use-case perspective. A user manual is provided with the system for correct detailed operation of the system. Within the prototype system, the only system set-up that requires extensive hardware calibration is the transmission with laser diodes as the laser must be calibrated to directly impact the receiver. The lack of enclosure further highlights this problem is one of the main components of future work. However infrared transmission is preferred in general due to its similar range whilst being less affected by obstacles and other interference.

As for error proofing of the system, the GUI has in-built, robust error proofing which ensures that the correct input for the system is always entered. In terms of actual system operation, the more practical and preferred NEC protocol has in-built modulation and demodulation ensuring that system accuracy is extremely high and is also one of the core CSR.

6.2 State of the System as Delivered

Overall, the system's design was adequate to fulfil the assigned Core System Requirements as well as satisfy most of the Additional System Requirements. Initially it was very finicky with calibration and transmission due to the protocols not being properly defined and timed, but as the data types and proper timing using interrupts

rather than polling was set up, the system was able to transmit strings and numerics consistently.

Based on the system performance, the following diagram (Fig. 6.9) has been developed to show each of the configuration options tested, and the basic decisions required to choose a configuration for a specific application.

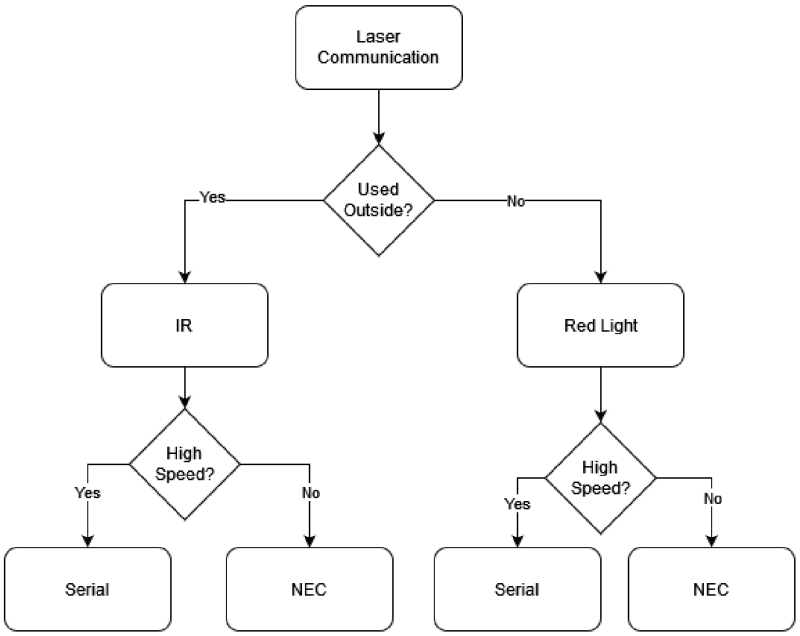


Figure 6.9 – Basic Decision Tree

6.3 Future Improvements

There are a number of improvements that can be made to the developed system, that we have noticed as we have undergone the work. Many of these will increase performance, robustness or reliability of the system. They include the following:

1. **Signal Power** - Both the transmitter and receiver modules were limited by their hardware in the power they could output. Primarily, this is due to safety regulations of commercially available lasers, but in future the system could utilise higher power components to achieve greater distances. It is also important to

note that the receivers used were all manufactured for infrared communication, yet red laser light was often used. This was able to work only due to high intensity of the red laser to pass through light filters, but to improve higher sensitivity purpose-built receivers could be developed.

2. **Optics** - Whilst minor optical focusing lenses were tested with the system developed and improved the distance of data transmission greatly, this could further be explored. The distance that a single beam of light can travel is largely probabilistic [17] due to the chance of coming into contact with a microscopic particle to block the light's path. Hence, there has been research and guides written on using beam expanders [18] to use a relatively wide beam rather than narrow to transmit the laser beam and reach longer distances. This would then be focused on the receiver side to a more intense beam, allowing for lower receiver sensitivity necessary.

3. **Optical Filtering** - The system developed was, at times, prone to errors from external environmental sources such as the sun or artificial lighting. Adding optical filters such as x-ray film or coloured translucent material to diffuse unwanted light pollution could reduce signal errors, improving performance and robustness.

4. **Enclosure** - The prototype system developed did not use an enclosure or physical structure other than the electronics themselves. In a commercial system, a simple 3D printed enclosure for electronics mounting the transmitter and receiver can increase the ease at which the user lines up the laser beam. This system, though, is to be deployed on a mobile device through software and existing technology as per use cases defined in Section 2.2. Further, custom enclosure could offer a purpose-built slot for optics or filters, supporting other improvements mentioned.

5. **Sound Frequency and Amplification** - It was noticed during testing that the sound transmission was of poor quality. To improve this, a micro-controller with a higher frequency clock could be used to increase fidelity, and a signal

amplification when output to the speaker to produce a louder sound.

Chapter 7

Safety Implications

7.1 Eye Safety

It is a legal requirement in Australia for lasers to be labelled according to their particular hazard. This information includes the class of the laser, the power output and the wavelength. See [19, 20] for more details on eye safety around lasers.

All laser pointers that are available to the Australian public must have a radiant power output of less than 1 milliwatt.

Diodes we used:

- Red light wavelength = 700nm
- Infrared light wavelength between 780 nm and 1 mm
- Both diodes have power output less then 1 milliwatt.

Laser diodes are focused beams of light which can be detrimental to the human eye if directly exposed, which can cause damage after only 10 seconds of exposure, causing injuries.

Methods to mitigate the exposure:

- Wear Eye Protection - Safety glasses designed to protect from light.
- Coordination - During trial runs, we coordinate specific the firing of the diodes to ensure no one was in way of the laser. One man was on watch to ensure no members of the general public walked into the way.
- Aversion Response - A human instinct which involves a blinking reflex when a laser beam shines onto the eye to protect itself.

Dangers and Injuries resulting laser beams include:

- If pointed up a laser can hit a plane and become very dangerous.
- Permanent damage - Loss of vision.
- Temporary damage - Eye irritation, spots, or headache.

Chapter 8

Conclusion

This report has presented the system definition, design and performance of a free-space laser communication mechanism. This system provides a simple way to send and receive data without the need for a wire or physical interfaces, yet is focused such that only receivers in the beam path can read the signal. This is proposed to see application in mobile devices for instant COVID check-ins through torch light.

This system was developed through a systematic and rigorous engineering design process in accordance with industry standards. The overall goals and operational scenarios were first defined, with high-level concept of operations and requirements to help guide design and ensure risk was managed throughout. It was designed to meet key requirements of transmitting data through free-space at a distance of 50m, as described in Section 2.3. Next, the system was broken down into modular subsystems with specified interfaces to logically simplify and separate parts for each team member to work on. Once a conceptual design was implemented for these subsystems, a preliminary design was crystallised and components were sourced to meet our design requirements. Then, the system was incrementally integrated and constructed, then tested with various functional and non-functional experiments to determine robustness. During construction, some issues were raised and the system was refined to improve robustness and ensure complete adherence to the core requirements.

Thus, the resultant system as described above was created, largely successfully.

Appendix A

Code

Generally, the code is commented to provide understanding and clarity to the reader. There are also links to related data sheets, guides and code repositories that are helpful for understanding the developed code. Much of the required software is copied and refactored from sources linked, as these problems have already been solved by experts. All credits go to original creators/developers, this software is not to be distributed or used for commercial purposes in its current state.

Note that the Listing caption occurs above the code to which it refers.

A.1 PIC18

Listing 1 – NEC Receiver C Software

```
/*
 * File:   PIC18_nec_receiver.c
 * Author: Rocco and James
 *
 * Created on 17 September 2021, 9:25 PM
 */

// a software module to make the PIC18 an IR NEC receiver
```

```
// connect a 38kHz receiver module to power from the board with signal
↪  output
// pin to RB4 and output to the PC PuTTY through RC6/TX and RS-232
↪  cable

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "ConfigRegsPIC18F452.h"

// Variables
char transmitString[12] = {0};
volatile char nec_done = 0;
volatile unsigned char i = 0, nec_state = 0, command = 0, inv_command
↪  = 0;
volatile unsigned int address = 0, timer_value = 0, timer_storage1 =
↪  0, timer_storage2 = 0, dummy = 0;
volatile unsigned long nec_code = 0, nec_buffer = 0;
const unsigned int myAddress = 0b1111000011110000;

// Functions
void setup(void);
void transmitMessage(char*);
void highPrioInterrupt(void);
void lowPrioInterrupt(void);
void timerOverflowISR(void);
void readISR(void);


#pragma code /* return to the default code section */
```

```
// ---------------------------------- Main
↪  -------------------------------------
void main(void) {
    setup();

    while(1){ // continue forever
        _asm NOP _endasm        // wait a cycle, for testing

        if(nec_done){               // If the mcu receives NEC message
        ↪  with successful
          T0CONbits.TMR0ON = 0;     // Disable Timer0 module
          INTCONbits.TMR0IE = 0;    // Disable Timer0 overflow
          ↪  interrupt
          nec_done = 0;             // Reset decoding process
          nec_state = 0;
          address = nec_code >> 16;
          command = nec_code >> 8;
          inv_command = nec_code;        // store received
          ↪  information

                  // Determine whether command and inverse match
          if((address == myAddress) && (command == ~inv_command)){
            sprintf(transmitString, (const rom far char *)
            ↪  "PASSED\r\n");
            transmitMessage(transmitString);
            sprintf(transmitString, (const rom far char *) "adr: %X,
            ↪  ", address);
            transmitMessage(transmitString);
            sprintf(transmitString, (const rom far char *) "cmd: %X,
            ↪  ", command);
            transmitMessage(transmitString);
```

```
            sprintf(transmitString, (const rom far char *) "icd:
            ↪ %X\r\n", inv_command);
            transmitMessage(transmitString);
            sprintf(transmitString, (const rom far char *) "ltr:
            ↪ %c\r\n", command);
            transmitMessage(transmitString);
          }
          else {
            sprintf(transmitString, (const rom far char *)
            ↪ "FAILED\r\n");
            transmitMessage(transmitString);
            sprintf(transmitString, (const rom far char *) "adr: %X,
            ↪ ", address);
            transmitMessage(transmitString);
            sprintf(transmitString, (const rom far char *) "cmd: %X,
            ↪ ", command);
            transmitMessage(transmitString);
            sprintf(transmitString, (const rom far char *) "icd:
            ↪ %X\r\n", inv_command);
            transmitMessage(transmitString);
            sprintf(transmitString, (const rom far char *) "ltr:
            ↪ %c\r\n", command);
            transmitMessage(transmitString);
          }

          INTCONbits.RBIE = 1;      // Enable PORTB IoC interrupt
          ↪ (RBIE)
        }
    }
}
```

```
// ---------------------------------- Functions
 ↪  ------------------------------------
/* this function sets up registers and values
 * in accordance with desired function
 */
void setup() {
    // Disable interrupts
    INTCONbits.GIEL = 0;
    INTCONbits.GIEH = 0;

    // Clear data registers
    TXREG = 0;
    RCREG = 0;

    // TRISC Setup: RX input, TX output
    TRISCbits.TRISC6 = 0;    // Make UART TX pin output
    TRISCbits.TRISC7 = 1;    // Make UART RX pin input
    PORTC = 0x00;            // Clear Port C

    // TXSTA and RCSTA Setup: Async, High Speed, 8-bit, Enable
    RCSTAbits.SPEN = 1;      // 1 = Serial port enabled
                             // (configures RX/DT and TX/CK pins as
                              ↪  serial port pins)
    TXSTAbits.SYNC = 0;      // 0 = Asynchronous mode
    TXSTAbits.TXEN = 1;      // 1 = Transmit enabled
    TXSTAbits.BRGH = 1;      // 1 = High speed
    SPBRG = 64;              // Baud Rate 9600

    // Priority Setup: High Priority Receive, Enable
    RCONbits.IPEN = 1;
```

```
    // INTCON Setup: Enable Timer0 Overflow + External PORTB Change
    ↪  Interrupts
    INTCONbits.TMR0IF = 0;
    INTCONbits.TMR0IE = 1;
    INTCONbits.RBIE = 1;     // IoC Interrupt Set-up
    dummy = PORTB;
    INTCONbits.RBIF = 0;
    INTCON2bits.RBIP = 1;    // High prio IoC
    INTCON2bits.TMR0IP = 0; // Low prio Timer Interrupt

    // Timer0 Module Configuration
    T0CONbits.TMR0ON = 0;
    T0CONbits.T0CS = 0;
    T0CONbits.T08BIT = 0;

    // PortB Ports Input + Digital
    TRISB = 0x10;            // Use RB4 for external interrupt
    INTCON2bits.RBPU = 1;    // Disable weak pull-up resistors

    // Enable global interrupts, high + low prio
    INTCONbits.GIEH = 1;
    INTCONbits.GIEL = 1;
}


/* this function is used to transmit messages with the USART module
 * without the need of interrupts (ie. polling)
 */
void transmitMessage(char* txPtr) {
    // Enable
```

```
    TXREG = 0;
    RCREG = 0;
    PIR1bits.TXIF = 0;

    while(*txPtr != '\0') {   // not null
        // Wait til TX reg is empty and loads in characters
        TXREG = *txPtr;
        txPtr++;
        while(PIR1bits.TXIF == 0);
    }

    // End
    while(PIR1bits.TXIF == 0);
}


// ------------------------------ Interrupts
 ↪  ---------------------------------
/* Interrupt on Change (pin RB4) */
#pragma code highPrioInterrupt = 0x08
void highPrioInterrupt(void) {
        _asm
        goto readISR
        _endasm
}


/* Timer Overflow Interrupt, due to message delay/transmission failure
 ↪  */
#pragma code lowPrioInterrupt = 0x18
void lowPrioInterrupt(void) {
```

```
        _asm
        goto timerOverflowISR
        _endasm
}


/* this subroutine will detect every part of the NEC message
 ↪  transmitted,
 * and determine whether it meets the protocol standards in timing,
 ↪  and if
 * so, it will store the message
 */
#pragma interrupt readISR
void  readISR(void) {
    if(INTCONbits.RBIF != 1){
        return;     // exit if not IoC triggered
    }

    // read timer and reset ready for interrupts to continue
    if(nec_state != 0){
        timer_storage1 = (unsigned int) TMR0L;
        timer_storage2 = ((unsigned int) TMR0H) << 8;    // Store
         ↪  Timer0 value
        timer_value = timer_storage1 | timer_storage2;
        TMR0H = 0;
        TMR0L = 0;  // Reset Timer0
    }

    // Reset flag
    dummy = PORTB;     // Read Port B to allow clearing
    INTCONbits.RBIF = 0;
```

```
// go through each of the parts of the NEC message to ensure
↪ proper communication
switch(nec_state){

    case 0 :                        // Start receiving IR data
    ↪ (we're at the beginning of 9ms pulse)
        nec_state = 1;              // Next state: end of 9ms
        ↪ pulse (start of 4.5ms space)
        i = 0;
        TMR0H = 0;
        TMR0L = 0;                  // Reset Timer0
        T0CON = 0b10000000;         // Enable Timer0 module with
        ↪ 1/2 prescaler (1 ticks every 0.8s)
        INTCONbits.TMR0IE = 1;      // Enable Timer overflow
        ↪ interrupt
        break;

    case 1 :                        // End of 9ms pulse
        _asm NOP _endasm
        if((timer_value > 20000) || (timer_value < 6000)) { //
        ↪ Invalid interval ==> stop decoding and reset
            nec_state = 0;          // Reset decoding process
            T0CONbits.TMR0ON = 0;   // Disable Timer0 module
            INTCONbits.TMR0IE = 0;  // Disable Timer0 overflow
            ↪ interrupt
        }
        else {
            nec_state = 2;          // Next state: end of 4.5ms
            ↪ space (start of 562.5s pulse)
        }
```

```
            break;

        case 2 :                        // End of 4.5ms space
            _asm NOP _endasm
            if((timer_value > 10000) || (timer_value < 3000)){  //
             ↪ Invalid interval ==> stop decoding and reset
                nec_state = 0;          // Reset decoding process
                T0CONbits.TMR0ON = 0;   // Disable Timer0 module
                INTCONbits.TMR0IE = 0;  // Disable Timer0 overflow
                 ↪ interrupt
            }
            else {
                nec_state = 3;          // Next state: end of 562.5s
                 ↪ pulse (start of 562.5s or 1687.5s space)
                nec_code = 0;           // clear NEC code
            }
            break;

        case 3 :                        // End of 562.5s pulse
            if((timer_value > 1000) || (timer_value < 300)){    //
             ↪ Invalid interval ==> stop decoding and reset
                nec_state = 0;          // Reset decoding process
                T0CONbits.TMR0ON = 0;   // Disable Timer0 module
                INTCONbits.TMR0IE = 0;  // Disable Timer0 overflow
                 ↪ interrupt
            }
            else {
                nec_state = 4;          // Next state: end of 562.5s
                 ↪ or 1687.5s space
            }
            break;
```

```
case 4 :                        // End of 562.5s or 1687.5s
↪ space
    if((timer_value > 4000) || (timer_value < 300)){    //
    ↪ Invalid interval ==> stop decoding and reset
        nec_state = 0;          // Reset decoding process
        T0CONbits.TMR0ON = 0;   // Disable Timer0 module
        INTCONbits.TMR0IE = 0;  // Disable Timer0 overflow
        ↪ interrupt
        break;
    }

    if(timer_value > 1750) {    // If space width > 1ms (short
    ↪ space)
        nec_buffer = 1;         // Write 1 to bit (31 - i)
        nec_buffer = nec_buffer << (31-i);
        nec_code = nec_code | nec_buffer;
    }
    else {                      // If space width < 1ms (long
    ↪ space)
        nec_buffer = 0;         // Write 0 to bit (31 - i)
        nec_buffer = nec_buffer << (31-i);
        nec_code = nec_code | nec_buffer;
    }
    i++;
    if(i > 31){                 // If all bits are received,
    ↪ stop bit will be detected on final bit
        nec_done = 1;           // Decoding process OK
        INTCONbits.RBIE = 0;    // Disable PORTB IoC interrupt
        ↪ (RBIE)
        break;
```

```
            }
            nec_state = 3;              // Next state: end of 562.5s
            ↪ pulse (start of 562.5s or 1687.5s space)
            break;
    }
}


/* this subroutine will reset the NEC message decoding process if the
 * timer has overflowed, indicating failure
 */
#pragma interrupt timerOverflowISR
void timerOverflowISR () {
    if(INTCONbits.TMR0IF == 1){
        INTCONbits.TMR0IF = 0;
        nec_state = 0;                  // Reset decoding process
        T0CONbits.TMR0ON = 0;           // Disable Timer0 module
        INTCONbits.TMR0IE = 0;          // Disable Timer0 overflow
        ↪ interrupt
    }
}
```

Listing 2 – Serial Receiver C Software

```
/*
 * File:   PIC18_serial_receiver.c
 * Author: Rocco and James
 *
 * Created on 5 November 2021, 12:21 PM
 */

// a software module to make the PIC18 an IR serial receiver
```

```
// connect receiver electronics to power from the board with signal
↪  output
// pin to RC7/RX and output to the PC PuTTY through RC6/TX and RS-232
↪  cable

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "ConfigRegsPIC18F452.h"

// definitions
#define BUFF_SIZE 128 // must be a power of 2
typedef struct circle_buffer {
    unsigned char buff[BUFF_SIZE];
    unsigned int write;
    unsigned int read;
} buffer;

// Declare functions used in this program
void init_uart(void);
void init_buffer(void);
//const char* get_rc_string(void);
char get_rc_letter(void);
int tx232C(char* txPtr);
void tx232Isr(void);
void rc232Isr(void);

// Declare global variables
const unsigned int buff_size_mask = BUFF_SIZE - 1;
buffer circ_buf;
char str[BUFF_SIZE+15] = {0};
```

```
char* cPtr;
int status;
int tx_flag;
int rc_flag;


#pragma code // return to the default code section
// ---------------------------------- Main
 ↪  -------------------------------------
void main(void) {
    init_buffer(); // clear circular buffer
    init_uart(); // init UART module

    while(1){
        if(rc_flag == 1){
            char letter = get_rc_letter();
            sprintf(str, (const rom far char *) "%c", letter);
            status = tx232C(str);
            rc_flag = 0;
        }
    } /* repeat forever */
} /* main() */


// ---------------------------------- Functions
 ↪  -------------------------------------
/* initialise USART module for 9600bps baud, 1 start bit, 1 stop bit,
 ↪
 * parity NONE, 8 data bits
 */
void init_uart(void){
```

```
// Disable interrupts
INTCONbits.GIEL = 0;
INTCONbits.GIEH = 0;

// Clear data registers
TXREG = 0;
RCREG = 0;

// Clear my flags
tx_flag = 0;
rc_flag = 0;

TRISCbits.TRISC6 = 0; // Make UART TX pin output
TRISCbits.TRISC7 = 1; // Make UART RX pin input
TXSTAbits.BRGH = 1;   // 1 = High speed
SPBRG = 64;           // Baud Rate 9600bps, 10MHz Fosc
// SPBRG = 9;            // Baud Rate 62500

RCSTAbits.CREN = 1;   // 1 = Enables receiver
RCSTAbits.SPEN = 1;   // 1 = Serial port enabled
                      // (configures RX/DT and TX/CK pins as
                      ↪  serial port pins)

TXSTAbits.SYNC = 0;   // 0 = Asynchronous mode
TXSTAbits.TXEN = 1;   // 1 = Transmit enabled

RCONbits.IPEN = 1;    // Enable Interrupt priority levels
IPR1bits.RCIP = 1;    // USART Receive Interrupt Priority 1 = High
↪  priority
IPR1bits.TXIP = 0;    // USART Transmit Interrupt Priority 0 = Low
↪  priority
```

```
    PIE1bits.RCIE = 1;     // 1 = Enables the USART receive interrupt

    INTCONbits.GIEL = 1;  // Enable low priority interrupts
    INTCONbits.GIE = 1;   // Enable interrupts
}


/* clear entire receive buffer */
void init_buffer(void){
    int i;
    for(i = 0; i < BUFF_SIZE - 1; i++){
        circ_buf.buff[i] = 0;
    }
    circ_buf.write = 0;
    circ_buf.read = 0;
}


/* transmit 1 string of data over serial with RS-232 protocol,
 ↪  interrupt-driven */
int tx232C(char* txPtr){
    cPtr = txPtr;
    PIE1bits.TXIE = 1;      // 1 = Enables the USART transmit
     ↪  interrupt

    while(tx_flag == 0);    // Wait until transfer complete
    tx_flag = 0;
    return 1;
}
```

```
/* return string of data that was received over serial */
//const char* get_rc_string(void){
//    const char* str = &circ_buf.buff[0] + circ_buf.read;
//
//    while(circ_buf.buff[circ_buf.read] != '\0'){
//        circ_buf.read++;
//        circ_buf.read &= buff_size_mask;    // reset insert index
 ↪  without test
//    }
//    circ_buf.read++;
//    circ_buf.read &= buff_size_mask;        // reset insert index
 ↪  without test
//
//    return str;
//}


/* return byte of data that was received over serial */
char get_rc_letter(void){
    char ltr = circ_buf.buff[circ_buf.read];
    circ_buf.read++;
    circ_buf.read &= buff_size_mask;        // reset insert index
     ↪  without test
    return ltr;
}


// ---------------------------------- Interrupts
 ↪  -------------------------------------
/* USART Receive Interrupt */
#pragma code highPrioInterrupt = 0x08
```

```
void highPrioInterrupt(void) {
        _asm
        goto rc232Isr
        _endasm
}


/* USART Transmit Interrupt */
#pragma code lowPrioInterrupt = 0x18
void lowPrioInterrupt(void) {
        _asm
        goto tx232Isr
        _endasm
}


/* this subroutine will read/store in a byte every time
 * a message is ready to be received in USART
 */
#pragma interrupt rc232Isr
void rc232Isr(void){
    if(!PIR1bits.RCIF || !PIE1bits.RCIE || !IPR1bits.RCIP){
        return;      // check if correct flags set
    }

    // Read in data to circular buffer
    if(RCSTAbits.FERR){          // Framing bit error
        char err = RCREG;
    }
    else if(RCSTAbits.OERR){     // Overrun error
        RCSTAbits.CREN = 0;
```

```
        RCSTAbits.CREN = 1;
    }
    else{
        circ_buf.buff[circ_buf.write] = RCREG;  // read character
        circ_buf.write++;
        circ_buf.write &= buff_size_mask;       // reset insert index
        ↪ without test
        rc_flag = 1;
    }

    PIR1bits.RCIF = 0;                          // Clear interrupt
    ↪ flag
}


/* this subroutine will transmit a byte every time
 * a message is ready to be transmitted with USART
 */
#pragma interrupt tx232Isr
void tx232Isr(void){
    if(!PIR1bits.TXIF || !PIE1bits.TXIE || IPR1bits.TXIP){
        return;     // check if correct flags set
    }

    // Transmit data
    if (*cPtr != '\0'){
        TXREG = *cPtr;              // Load TXREG with data
        cPtr++;                     // Increment pointer
    }
    else if (*cPtr == '\0'){
        tx_flag = 1;
```

```
        while(TXSTAbits.TRMT == 0); // Wait until TSR empty
        PIE1bits.TXIE = 0;          // 0 = Disables the USART transmit
         ↪  interrupt
    }
}
```

Listing 3 – Serial Receiver C Software, Output to Sound

```
/*
 * File:   PIC18_serial_audio_receiver.c
 * Author: James
 *
 * Created on 6 November 2021, 8:04 PM
 */

// a software module to make the PIC18 an IR serial receiver
// connect receiver electronics to power from the board with signal
 ↪  output
// pin to RC7/RX and output to a speaker on CCP1/RC2

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "ConfigRegsPIC18F452.h"

// definitions
#define BUFF_SIZE 128 // must be a power of 2
typedef struct circle_buffer {
    unsigned char buff[BUFF_SIZE];
    unsigned int write;
    unsigned int read;
} buffer;
```

```
// Declare functions used in this program
void init_uart(void);
void init_buffer(void);
//const char* get_rc_string(void);
char get_rc_letter(void);
int tx232C(char* txPtr);
void tx232Isr(void);
void speakerIsr(void);
void rc232Isr(void);

// Declare global variables
const unsigned int buff_size_mask = BUFF_SIZE - 1;
buffer circ_buf;
char str[BUFF_SIZE+15] = {0};
char* cPtr;
int status;
int tx_flag;
int rc_flag;
volatile char theByte = 150;


#pragma code // return to the default code section
// ---------------------------------- Main
↪  -------------------------------------
void main(void) {
    init_buffer(); // clear circular buffer
    init_uart(); // init UART module

    while(1){
    } /* repeat forever */
```

```
} /* main() */


// ---------------------------------- Functions
 ↪  -----------------------------------
/* initialise USART module for 9600bps baud, 1 start bit, 1 stop bit,
 ↪
 * parity NONE, 8 data bits
 */
void init_uart(void){
    // Disable interrupts
    INTCONbits.GIEL = 0;
    INTCONbits.GIEH = 0;

    // Clear data registers
    TXREG = 0;
    RCREG = 0;

    // Clear my flags
    tx_flag = 0;
    rc_flag = 0;

    TRISCbits.TRISC6 = 0; // Make UART TX pin output
    TRISCbits.TRISC7 = 1; // Make UART RX pin input
    TXSTAbits.BRGH = 1;   // 1 = High speed
    SPBRG = 64;           // Baud Rate 9,600bps, 10MHz Fosc
//    SPBRG = 9;            // Baud Rate 62,500
//    SPBRG = 4;            // Baud Rate 110,000

    RCSTAbits.CREN = 1;   // 1 = Enables receiver
    RCSTAbits.SPEN = 1;   // 1 = Serial port enabled
```

```
                        // (configures RX/DT and TX/CK pins as
                        ↪  serial port pins)

TXSTAbits.SYNC = 0;     // 0 = Asynchronous mode
TXSTAbits.TXEN = 1;     // 1 = Transmit enabled

RCONbits.IPEN = 1;      // Enable Interrupt priority levels
IPR1bits.RCIP = 1;      // USART Receive Interrupt Priority 1 = High
↪  priority
IPR1bits.TXIP = 0;      // USART Transmit Interrupt Priority 0 = Low
↪  priority
PIE1bits.RCIE = 1;      // 1 = Enables the USART receive interrupt

// Timer 2 PWM signal = 39kHz
TRISCbits.TRISC2 = 0; // Make CCP1 pin output
PR2 = 63;             // Freq = 39kHz
CCPR1L = 0b00011111;  // Duty = x
CCP1CON = 0b00111100; // Enable PWM, Duty = x
T2CON = 0b00000100;   // Enable 16-bit Timer2, Scale = 1

// Timer 1 Duty cycle updater = 5kHz
TRISCbits.TRISC1 = 0; // Make CCP2 pin output
T1CON = 0b10000001;   // Enable 8-bit Timer1, Scale = 1
CCP2CON = 0b00001011; // Compare, special event Mode for CCP2 on
↪  RC1
CCPR2L = 0b11110100;
CCPR2H = 0b00000011;  // = 500
PIE2bits.CCP2IE = 1;  // Enable CCP2 interrupt
IPR2bits.CCP2IP = 0;  // Low priority
PIR2bits.CCP2IF = 0;  // Clear flag
```

```
    INTCONbits.GIEL = 1;  // Enable low priority interrupts
    INTCONbits.GIE = 1;   // Enable interrupts
}


/* clear entire receive buffer */
void init_buffer(void){
    int i;
    for(i = 0; i < BUFF_SIZE - 1; i++){
        circ_buf.buff[i] = 0;
    }
    circ_buf.write = 0;
    circ_buf.read = 0;
}


/* return byte of data that was received over serial */
char get_rc_letter(void){
    char ltr = circ_buf.buff[circ_buf.read];
    circ_buf.read++;
    circ_buf.read &= buff_size_mask;        // reset insert index
    ↪ without test
    return ltr;
}


// ---------------------------------- Interrupts
↪ -------------------------------------
/* USART Receive Interrupt */
#pragma code highPrioInterrupt = 0x08
void highPrioInterrupt(void) {
```

```
        _asm
        goto rc232Isr
        _endasm
}


/* USART Transmit Interrupt */
#pragma code lowPrioInterrupt = 0x18
void lowPrioInterrupt(void) {
        _asm
        goto speakerIsr
        _endasm
}


/* this subroutine will read/store in a byte every time
 * a message is ready to be received
 */
#pragma interrupt rc232Isr
void rc232Isr(void){
    if(!PIR1bits.RCIF || !PIE1bits.RCIE || !IPR1bits.RCIP){
        return;      // check if correct flags set
    }

    // Read in data to circular buffer
    if(RCSTAbits.FERR){         // Framing bit error
        char err = RCREG;
    }
    else if(RCSTAbits.OERR){    // Overrun error
        RCSTAbits.CREN = 0;
        RCSTAbits.CREN = 1;
```

```
    }
    else{
        theByte = RCREG;
    }

    PIR1bits.RCIF = 0;          // Clear interrupt flag
}


/* this subroutine will write the byte received to the CCP1 duty
↪  cycle
 * at a frequency of 5kHz
 */
#pragma interrupt speakerIsr
void speakerIsr(void){
    if(!PIR2bits.CCP2IF || !PIE2bits.CCP2IE || IPR2bits.CCP2IP){
        return;         // check if correct flags set
    }

    // Read received byte into the duty cycle of speaker output PCM
    CCPR1L = theByte >> 2;
    CCP1CONbits.DC1B1 = theByte >> 1;
    CCP1CONbits.DC1B0 = theByte;

    // Clear flag
    PIR2bits.CCP2IF = 0;
}
```

A.2 Arduino

Listing 4 – GUI NEC Transmit Arduino Software

```
// see for details: https://gist.github.com/EEVblog/6206934

// variable declarations
const int OUTPIN = 2;
const int SinglePeriod = 562;
String myWord;
const unsigned long address = 0b1111000011110000;
const int BITMASK = 0xFFFF;
const int BITMASK2 = 0xFF;
const int BITWIDTH = 16;
const int BITWIDTH2 = BITWIDTH/2;
unsigned long message = 0;



/************************** SETUP ********************************/
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  pinMode(OUTPIN, OUTPUT);
  digitalWrite(OUTPIN, LOW);
}



/************************** LOOP *********************************/
void loop() {
  // put your main code here, to run repeatedly:
  if (Serial.available() > 0) {
    // read the oldest byte in the serial buffer:
    myWord = Serial.readStringUntil('\n');
  }
```

```
  // send a word
  for(int i = 0; i < myWord.length(); i++){
    message = ((address & BITMASK) << BITWIDTH) | (myWord[i] <<
    ↪  BITWIDTH2) | (~myWord[i] & BITMASK2);
    IRSendMessage(message);
//    delay(40.5);  // typical NEC delay between messages is 40.5ms or
↪   equivalently 108ms between start bits
    delay(70);
  }

  // delay to indicate end of word, for user
//  delay(500);
}


/************************* FUNCTIONS
↪  ******************************/
// Ouput the 38KHz carrier frequency for the required time in
↪  microseconds
// This is timing critial and just do-able on an Arduino using the
↪  standard I/O functions.
void IRCarrier(unsigned int IRtimemicroseconds){
  for(int i = 0; i < (IRtimemicroseconds / 26); i++){
    digitalWrite(OUTPIN, HIGH); // turn on the IR LED
    // NOTE: digitalWrite takes about 3.5us to execute, so we need to
    ↪  factor that into the timing.
    delayMicroseconds(9);       // delay for 13us (9us +
    ↪  digitalWrite), half the carrier frequnecy
    digitalWrite(OUTPIN, LOW);  // turn off the IR LED
```

```
    delayMicroseconds(9);        // delay for 13us (9us +
    ↪ digitalWrite), half the carrier frequnecy
    }
}

// NEC protocol message is 32-bits = 4 bytes (ADDR1 | ADDR2 | COMMAND1
↪ | COMMAND2)
void IRSendMessage(unsigned long msg){
  //send the leading pulse
  IRCarrier(9000);          // 9ms of carrier
  delayMicroseconds(4500); // 4.5ms of silence

  // send the user defined 4 byte/32-bit code
  for (int i = 0; i < 32; i++){            // send all 4 bytes or 32
  ↪ bits
    IRCarrier(SinglePeriod);               // turn on the 38kHz
    ↪ carrier for one period

    if(msg & 0x80000000){                  // get the current bit by
    ↪ masking all but the MSB
      delayMicroseconds(3 * SinglePeriod); // a HIGH is three period
      ↪ silence
    }
    else{
      delayMicroseconds(SinglePeriod);     // a LOW is only 1 period
      ↪ silence
    }
    msg <<= 1;                             // shift to the next bit
    ↪ for this byte
  }
```

```
  IRCarrier(SinglePeriod);                    // send a single STOP bit.
}
```

Listing 5 – GUI Serial Sound Transmit Arduino Software

```
// see for details: https://playground.arduino.cc/Code/PCMAudio/
// explanation/demo:
↪  https://www.youtube.com/watch?v=MCTqC2-AN7o&t=8s&ab_channel=iforce2d

// Declare variables
const int ledPin = LED_BUILTIN;
bool value = 0;
volatile byte myByte = 0;


/************************** SETUP ********************************/
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
}


/************************** LOOP *********************************/
void loop() {
  // put your main code here, to run repeatedly:
  if (Serial.available() > 0) {
    // send data as soon as received
    Serial.print(Serial.read());
    digitalWrite(ledPin, HIGH);
  }
```

```
  digitalWrite(ledPin, LOW);
}
```

Listing 6 – GUI Serial Transmit Arduino Software

```
// Connect laser to digital pin 1, TX

// Declare variables
String myWord = "HELLO EVERYONE\r\n";
const int ledPin = LED_BUILTIN;
bool value = 0;


/************************** SETUP ********************************/
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  pinMode(1, OUTPUT);
  pinMode(ledPin, OUTPUT);
}


/************************** LOOP *********************************/
void loop() {
  // put your main code here, to run repeatedly:
  if (Serial.available() > 0) {
    // read the oldest byte in the serial buffer:
    myWord = Serial.readStringUntil('\n');
  }

  // transmit each character of the word received one at a time
  for(int i = 0; i < myWord.length(); i++){
```

```
    Serial.print(myWord[i]);
    value = (bool) myWord[i];
    digitalWrite(ledPin, value);
  }

  // delay to indicate pause for testing
  digitalWrite(ledPin, LOW);
  delay(100);
}
```

Listing 7 – NEC Receive Testing Arduino Software

```
// see https://simple-circuit.com/arduino-nec-remote-control-decoder/
// or https://github.com/wagiminator/ATtiny13-TinyDecoder

// note: I had to increase the tolerance for the 9ms start bit, see
↪  the top link for more info

// Declare variables
int inPin = 3;
const int ledPin = LED_BUILTIN;
bool value = 0;
int count = 0;
boolean nec_done = 0;
byte i, nec_state = 0, command, inv_command;
unsigned int address;
unsigned long nec_code;
const unsigned long myAddress = 0b1111000011110000;


/************************* SETUP ********************************/
void setup() {
```

```
  // put your setup code here, to run once:
  Serial.begin(9600);
  pinMode(inPin, INPUT);
  pinMode(ledPin, OUTPUT);

  // Timer1 module configuration
  TCCR1A = 0;
  TCCR1B = 0;                                          // Disable Timer1
   ↪  module
  TCNT1  = 0;                                          // Set Timer1 preload
   ↪  value to 0 (reset)
  TIMSK1 = 1;                                          // enable Timer1
   ↪  overflow interrupt
  attachInterrupt(digitalPinToInterrupt(inPin), remote_read, CHANGE);
   ↪  // Enable external interrupt (INT1)
}


/************************** LOOP ********************************/
void loop() {
  // put your main code here, to run repeatedly:
  // reading messages too fast does not work as this code takes some
   ↪  time to run
  if(nec_done){                                          // If the mcu
   ↪  receives NEC message with successful
    nec_done = 0;                                        // Reset decoding
     ↪  process
    nec_state = 0;
    TCCR1B = 0;                                          // Disable Timer1
     ↪  module
    address = nec_code >> 16;
```

```
command = nec_code >> 8;
inv_command = nec_code;

if(address == myAddress && command != (byte) ~inv_command){  //
↪  Determine whether command and inverse match
  Serial.println("Command or Address FAILED to Match. ");
  Serial.print("Address: ");
  Serial.println(address, BIN);
  Serial.print("Command: ");
  Serial.println(command, BIN);
  Serial.print("Inverted inverse: ");
  Serial.print((byte) ~inv_command, BIN);
  Serial.println(". ");

  // display to built in LED
  value = (bool) command;
  digitalWrite(ledPin, value);
}
else{
  Serial.print("Received Command: ");
  Serial.print(command, BIN);
  Serial.println(", ");
  Serial.print("Address: ");
  Serial.print(address, BIN);
  Serial.println(". ");

  // display to built in LED
  value = (bool) command;
  digitalWrite(ledPin, value);
}
```

```
    // toggling led display
    if (count == 2){
      digitalWrite(ledPin, HIGH);
      count = 0;
    }
    else{
      digitalWrite(ledPin, LOW);
      count++;
    }

    attachInterrupt(digitalPinToInterrupt(inPin), remote_read,
    ↪  CHANGE);     // Enable external interrupt (INT1)
  }
}


/************************* FUNCTIONS
↪  ******************************/
// determine where we are in the message and ensure protocol is
↪  adhered to
void remote_read() {
  unsigned int timer_value;

  // read timer and reset ready for interrupts to continue
  if(nec_state != 0){
    timer_value = TCNT1;                          // Store Timer1
    ↪  value
    TCNT1 = 0;                                    // Reset Timer1
  }
```

```
// go through each of the parts of the NEC message to ensure proper
 ↪  communication
switch(nec_state){

 case 0 :                                          // Start receiving IR
 ↪  data (we're at the beginning of 9ms pulse)
  TCNT1  = 0;                                      // Reset Timer1
  TCCR1B = 2;                                      // Enable Timer1
  ↪  module with 1/8 prescaler ( 2 ticks every 1 us)
  nec_state = 1;                                   // Next state: end of
  ↪  9ms pulse (start of 4.5ms space)
  i = 0;
  return;

 case 1 :                                          // End of 9ms pulse
  if((timer_value > 21000) || (timer_value < 15000)){         //
  ↪  Invalid interval ==> stop decoding and reset
    nec_state = 0;                                 // Reset decoding
    ↪  process
    TCCR1B = 0;                                    // Disable Timer1
    ↪  module
  }
  else
    nec_state = 2;                                 // Next state: end of
    ↪  4.5ms space (start of 562µs pulse)
  return;

 case 2 :                                          // End of 4.5ms
 ↪  space
  if((timer_value > 10000) || (timer_value < 8000)){          //
  ↪  Invalid interval ==> stop decoding and reset
```

```
      nec_state = 0;                                  // Reset decoding
       ↪ process
      TCCR1B = 0;                                     // Disable Timer1
       ↪ module
    }
    else
      nec_state = 3;                                  // Next state: end of
       ↪ 562µs pulse (start of 562µs or 1687µs space)
    return;

  case 3 :                                            // End of 562µs
   ↪ pulse
    if((timer_value > 1400) || (timer_value < 800)){          //
     ↪ Invalid interval ==> stop decoding and reset
      TCCR1B = 0;                                     // Disable Timer1
       ↪ module
      nec_state = 0;                                  // Reset decoding
       ↪ process
    }
    else
      nec_state = 4;                                  // Next state: end of
       ↪ 562µs or 1687µs space
    return;

  case 4 :                                            // End of 562µs or
   ↪ 1687µs space
    if((timer_value > 3600) || (timer_value < 800)){          //
     ↪ Invalid interval ==> stop decoding and reset
      TCCR1B = 0;                                     // Disable Timer1
       ↪ module
```

```
      nec_state = 0;                                  // Reset decoding
       ↪ process
      return;
    }
    if(timer_value > 2000)                            // If space width >
     ↪ 1ms (short space)
      bitSet(nec_code, (31 - i));                     // Write 1 to bit (31
       ↪ - i)
    else                                              // If space width <
     ↪ 1ms (long space)
      bitClear(nec_code, (31 - i));                   // Write 0 to bit (31
       ↪ - i)
    i++;
    if(i > 31){                                       // If all bits are
     ↪ received, stop bit will be detected on final bit
      nec_done = 1;                                   // Decoding process
       ↪ OK
      detachInterrupt(digitalPinToInterrupt(inPin));
       ↪ // Disable external interrupt (INT1)
      return;
    }
    nec_state = 3;                                    // Next state: end of
     ↪ 562µs pulse (start of 562µs or 1687µs space)

  }
}


// if this activates, the timer has overrun and has taken too long, so
 ↪ message transmission failure
```

```
ISR(TIMER1_OVF_vect) {                          // Timer1 interrupt
↪  service routine (ISR)
  nec_state = 0;                                // Reset decoding
  ↪  process
  TCCR1B = 0;                                   // Disable Timer1
  ↪  module
}
```

Listing 8 – NEC Transmit Arduino Software

```
// see for details: https://gist.github.com/EEVblog/6206934

// Declare variables
const int outPin = 2;
const int ledPin = LED_BUILTIN;
bool value = 0;
const int SinglePeriod = 562;
String myWord = "HELLO EVERYONE ";
const unsigned long IRcode = 0b11000001110001111100000000111111; //
↪  holds 32-bits!
const unsigned long address = 0b1111000011110000;
const int BITMASK = 0xFFFF;
const int BITMASK2 = 0xFF;
const int BITWIDTH = 16;
const int BITWIDTH2 = BITWIDTH/2;
unsigned long message = 0;


/************************** SETUP ********************************/
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
```

```
  pinMode(outPin, OUTPUT);
  digitalWrite(outPin, LOW);
  pinMode(ledPin, OUTPUT);
}


/************************* LOOP ******************************/
void loop() {
  // put your main code here, to run repeatedly:

  // send a word
  for(int i = 0; i < myWord.length(); i++){
    // create and send individual message
    message = ((address & BITMASK) << BITWIDTH) | (myWord[i] <<
    ↪  BITWIDTH2) | (~myWord[i] & BITMASK2);
    IRSendMessage(message);

    // display to built in LED
//    value = (bool) myWord[i];
//    digitalWrite(ledPin, value);
//
//    // display to serial
//    Serial.print("Character: ");
//    Serial.print(myWord[i], BIN);
//    Serial.print(", ");
//    Serial.println(~myWord[i] & BITMASK2, BIN);
//    Serial.print("Full Message: ");
//    Serial.println(message, BIN);

//    delay(40.5);     // this slight delay between characters can
↪  make reading the sent values easier,
```

```
                    // as traditionally only 1 message needs to be
                    ↪  sent rather than a string of them
    delay(70);
  }

  // delay for ease of testing
//  digitalWrite(ledPin, LOW);
//  delay(500);
}



/************************* FUNCTIONS
↪  ******************************/
// Ouput the 38KHz carrier frequency for the required time in
↪  microseconds
// This is timing critial and just do-able on an Arduino using the
↪  standard I/O functions.
void IRCarrier(unsigned int IRtimemicroseconds){
  for(int i = 0; i < (IRtimemicroseconds / 26); i++){
    digitalWrite(outPin, HIGH); // turn on the IR LED
    // NOTE: digitalWrite takes about 3.5us to execute, so we need to
    ↪  factor that into the timing.
    delayMicroseconds(9);       // delay for 13us (9us +
    ↪  digitalWrite), half the carrier frequnecy
    digitalWrite(outPin, LOW);  // turn off the IR LED
    delayMicroseconds(9);       // delay for 13us (9us +
    ↪  digitalWrite), half the carrier frequnecy
    }
}
```

```
// NEC protocol message is 32-bits = 4 bytes (ADDR1 | ADDR2 | COMMAND1
↪  | COMMAND2)
void IRSendMessage(unsigned long msg){
  //send the leading pulse
  IRCarrier(9000);          // 9ms of carrier
  delayMicroseconds(4500); // 4.5ms of silence

  // send the user defined 4 byte/32-bit code
  for (int i = 0; i < 32; i++){            // send all 4 bytes or 32
  ↪  bits
    IRCarrier(SinglePeriod);               // turn on the 38kHz
    ↪  carrier for one period

    if(msg & 0x80000000){                  // get the current bit by
    ↪  masking all but the MSB
      delayMicroseconds(3 * SinglePeriod); // a HIGH is three period
      ↪  silence
    }
    else{
      delayMicroseconds(SinglePeriod);     // a LOW is only 1 period
      ↪  silence
    }
    msg <<= 1;                             // shift to the next bit
    ↪  for this byte
  }

  IRCarrier(SinglePeriod);                 // send a single STOP bit.
}
```

Listing 9 – Serial Receive Sound Testing Arduino Software

```
// see for details: https://playground.arduino.cc/Code/PCMAudio/
```

```
// explanation/demo:
↪  https://www.youtube.com/watch?v=MCTqC2-AN7o&t=8s&ab_channel=iforce2d

/*
 * speaker_pcm
 *
 * Plays 8-bit PCM audio on pin 11 using pulse-width modulation
↪  (PWM).
 * For Arduino with Atmega168 at 16 MHz.
 *
 * Uses two timers. The first changes the sample value 8000 times a
↪  second.
 * The second holds pin 11 high for 0-255 ticks out of a 256-tick
↪  cycle,
 * depending on sample value. The second timer repeats 62500 times per
↪  second
 * (16000000 / 256), much faster than the playback rate (8000 Hz), so
 * it almost sounds halfway decent, just really quiet on a PC
↪  speaker.
 *
 * Takes over Timer 1 (16-bit) for the 8000 Hz timer. This breaks PWM
 * (analogWrite()) for Arduino pins 9 and 10. Takes Timer 2 (8-bit)
 * for the pulse width modulation, breaking PWM for pins 11 & 3.
 *
 * References:
 *
↪  https://www.uchobby.com/index.php/2007/11/11/arduino-sound-part-1/
 *     https://www.atmel.com/dyn/resources/prod_documents/doc2542.pdf
 *     https://www.evilmadscientist.com/article.php/avrdac
 *     https://gonium.net/md/2006/12/27/i-will-think-before-i-code/
 *     https://fly.cc.fer.hr/GDM/articles/sndmus/speaker2.html
```

```
 *     https://www.gamedev.net/reference/articles/article442.asp
 *
 * Michael Smith <michael@hurts.ca>
 */

#include <stdint.h>
#include <avr/interrupt.h>
#include <avr/io.h>
#include <avr/pgmspace.h>

#define SAMPLE_RATE 8000

/*
 * The audio data needs to be unsigned, 8-bit, 8000 Hz, and small
↪  enough
 * to fit in flash. 10000-13000 samples is about the limit.
 *
 * sounddata.h should look like this:
 *     const int sounddata_length=10000;
 *     const unsigned char sounddata_data[] PROGMEM = { ..... };
 *
 * You can use wav2c from GBA CSS:
 *     https://thieumsweb.free.fr/english/gbacss.html
 * Then add "PROGMEM" in the right place. I hacked it up to dump the
↪  samples
 * as unsigned rather than signed, but it shouldn't matter.
 *
 *
↪  https://musicthing.blogspot.com/2005/05/tiny-music-makers-pt-4-mac-startup.h
 * mplayer -ao pcm macstartup.mp3
 * sox audiodump.wav -v 1.32 -c 1 -r 8000 -u -1 macstartup-8000.wav
```

```
 * sox macstartup-8000.wav macstartup-cut.wav trim 0 10000s
 * wav2c macstartup-cut.wav sounddata.h sounddata
 *
 * (starfox) nb. under sox 12.18 (distributed in CentOS 5), i needed
↪  to run
 * the following command to convert my wav file to the appropriate
↪  format:
 * sox audiodump.wav -c 1 -r 8000 -u -b macstartup-8000.wav
 */

#include "sounddata.h"

// Declare variables
int ledPin = LED_BUILTIN;
int speakerPin = 11; // Can be either 3 or 11, two PWM outputs
↪  connected to Timer 2
volatile uint16_t sample;
byte lastSample;

volatile byte theByte = 0;

void stopPlayback(){
    // Disable playback per-sample interrupt.
    TIMSK1 &= ~_BV(OCIE1A);

    // Disable the per-sample timer completely.
    TCCR1B &= ~_BV(CS10);

    // Disable the PWM timer.
    TCCR2B &= ~_BV(CS10);
```

```
    digitalWrite(speakerPin, LOW);
}

// This is called at 8000 Hz to load the next sample.
ISR(TIMER1_COMPA_vect) {
  // load byte into duty cycle register
  OCR2A = theByte;
//  OCR2A = pgm_read_byte(&sounddata_data[sample]); // previously
}

void startPlayback(){
    pinMode(speakerPin, OUTPUT);

    // Set up Timer 2 to do pulse width modulation on the speaker
    // pin.

    // Use internal clock (datasheet p.160)
    ASSR &= ~(_BV(EXCLK) | _BV(AS2));

    // Set fast PWM mode  (p.157)
    TCCR2A |= _BV(WGM21) | _BV(WGM20);
    TCCR2B &= ~_BV(WGM22);

    if(speakerPin==11){
        // Do non-inverting PWM on pin OC2A (p.155)
        // On the Arduino this is pin 11.
        TCCR2A = (TCCR2A | _BV(COM2A1)) & ~_BV(COM2A0);
        TCCR2A &= ~(_BV(COM2B1) | _BV(COM2B0));
        // No prescaler (p.158)
        TCCR2B = (TCCR2B & ~(_BV(CS12) | _BV(CS11))) | _BV(CS10);
```

```
    // Set initial pulse width to the first sample.
    OCR2A = pgm_read_byte(&sounddata_data[0]);
} else {
    // Do non-inverting PWM on pin OC2B (p.155)
    // On the Arduino this is pin 3.
    TCCR2A = (TCCR2A | _BV(COM2B1)) & ~_BV(COM2B0);
    TCCR2A &= ~(_BV(COM2A1) | _BV(COM2A0));
    // No prescaler (p.158)
    TCCR2B = (TCCR2B & ~(_BV(CS12) | _BV(CS11))) | _BV(CS10);

    // Set initial pulse width to the first sample.
    OCR2B = pgm_read_byte(&sounddata_data[0]);
}


// Set up Timer 1 to send a sample every interrupt.
cli();

// Set CTC mode (Clear Timer on Compare Match) (p.133)
// Have to set OCR1A *after*, otherwise it gets reset to 0!
TCCR1B = (TCCR1B & ~_BV(WGM13)) | _BV(WGM12);
TCCR1A = TCCR1A & ~(_BV(WGM11) | _BV(WGM10));

// No prescaler (p.134)
TCCR1B = (TCCR1B & ~(_BV(CS12) | _BV(CS11))) | _BV(CS10);

// Set the compare register (OCR1A).
// OCR1A is a 16-bit register, so we have to do this with
// interrupts disabled to be safe.
OCR1A = F_CPU / SAMPLE_RATE;    // 16e6 / 8000 = 2000
```

```
    // Enable interrupt when TCNT1 == OCR1A (p.136)
    TIMSK1 |= _BV(OCIE1A);

    lastSample = pgm_read_byte(&sounddata_data[sounddata_length-1]);
    sample = 0;
    sei();
}


/************************* SETUP ********************************/
void setup(){
    Serial.begin(115200);
//    pinMode(ledPin, OUTPUT);
//    digitalWrite(ledPin, HIGH);
    startPlayback();
}


/************************* LOOP *********************************/
void loop(){
    if (Serial.available() > 0){
      theByte = Serial.read();
//      bool value = (bool) theByte;
//      digitalWrite(ledPin, value);
    }
}
```

Listing 10 – Serial Receive Testing Arduino Software

```
// Connect receiver to RX pin, 0

// Declare variables
```

```
const int ledPin = LED_BUILTIN;
bool value = 0;
char incomingByte = 0; // for incoming serial data


/************************** SETUP ********************************/
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
//  pinMode(readPin, INPUT);
  pinMode(ledPin, OUTPUT);
//  pinMode(8, OUTPUT);
}


/************************** LOOP *********************************/
void loop() {
  // put your main code here, to run repeatedly:
  if (Serial.available() > 0) {
    // read the incoming byte:
    incomingByte = Serial.read();

    // say what you got:
    Serial.print("I received: ");
    Serial.println(incomingByte);
  }

  // just an example value
  value = (bool) incomingByte;
  digitalWrite(ledPin, value);
}
```

Listing 11 – Serial Transmit Sound Arduino Software

```
// see for details: https://playground.arduino.cc/Code/PCMAudio/
// explanation/demo:
↪  https://www.youtube.com/watch?v=MCTqC2-AN7o&t=8s&ab_channel=iforce2d

#include "sounddata.h"

// Declare variables
const int ledPin = LED_BUILTIN;
bool value = 0;
volatile uint16_t sample;


/************************** SETUP ********************************/
void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  pinMode(ledPin, OUTPUT);
  Serial.println("info: ");
  Serial.println(sounddata_length);
  Serial.println(sounddata_data[0]);
  delay(1000);
}


/************************** LOOP *********************************/
void loop() {
  // put your main code here, to run repeatedly:
  // send sound data, one byte at a time
```

```
  sample = 0;
  digitalWrite(ledPin, HIGH);
  while(sample < sounddata_length){
    Serial.write(sounddata_data[sample]);
    sample++;
  }

  // delay after sound is finished playing
  digitalWrite(ledPin, LOW);
  delay(1000);
}
```

Listing 12 – MAC Startup Testing Sound Header File

```
// sounddata sound made by wav2c
// (wav2c modified to use unsigned samples)

/* const int sounddata_sampleRate=8000; */
const int sounddata_length=10000;

const unsigned char sounddata_data[] PROGMEM = {128,
128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128,
↪  128, 128, 128, 128, 128, 128,
128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128,
↪  128, 128, 128, 128, 128, 128,
128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128,
↪  128, 128, 128, 128, 128, 128,
128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128,
↪  128, 128, 128, 128, 128, 128,
128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128,
↪  128, 128, 128, 128, 128, 128,
```

```
128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128,
↪  128, 128, 128, 128, 128, 128,
128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128,
↪  128, 128, 128, 128, 128, 128,
128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128,
↪  128, 128, 128, 128, 128, 128,
128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128,
↪  128, 128, 128, 128, 128, 128,
128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128,
↪  129, 127, 129, 128, 127, 133,
117, 109, 125, 121, 116, 132, 140, 126, 114, 114, 116, 120, 114, 93,
↪  73, 66, 76, 116, 142, 129,
128, 129, 120, 119, 118, 104, 87, 123, 181, 194, 196, 198, 189, 176,
↪  160, 162, 172, 164, 164, 183,
197, 188, 168, 167, 170, 165, 185, 209, 206, 196, 196, 199, 185, 162,
↪  156, 167, 176, 173, 170, 166,
151, 142, 140, 134, 130, 127, 113, 86, 67, 66, 69, 75, 73, 75, 86, 90,
↪  91, 84, 65, 48,
41, 30, 26, 56, 91, 88, 72, 70, 73, 82, 89, 73, 57, 60, 74, 89, 92,
↪  77, 63, 60,
53, 47, 56, 64, 63, 61, 56, 54, 52, 36, 16, 22, 51, 66, 67, 70, 76,
↪  88, 99, 92,
77, 74, 85, 100, 106, 97, 83, 85, 96, 108, 133, 160, 164, 144, 113,
↪  96, 91, 82, 74, 76,
89, 97, 97, 97, 82, 54, 40, 41, 41, 43, 56, 74, 78, 64, 55, 64, 72,
↪  72, 84, 102,
108, 116, 126, 127, 124, 127, 134, 134, 138, 148, 152, 156, 164, 165,
↪  169, 171, 160, 156, 157, 152,
151, 145, 133, 136, 153, 166, 165, 163, 165, 161, 156, 158, 155, 147,
↪  148, 160, 185, 209, 215, 220,
```

```
220, 204, 200, 208, 205, 200, 202, 209, 214, 213, 205, 198, 194, 194,
↪  203, 219, 231, 235, 230, 219,
200, 184, 177, 170, 170, 177, 172, 164, 163, 158, 156, 160, 163, 161,
↪  142, 116, 103, 96, 89, 93,
101, 105, 111, 116, 120, 110, 89, 80, 78, 75, 73, 80, 93, 91, 77, 69,
↪  70, 77, 91, 98,
89, 87, 93, 95, 95, 94, 97, 96, 91, 94, 99, 100, 101, 95, 83, 78, 79,
↪  71, 56, 41,
37, 53, 64, 63, 72, 82, 83, 82, 80, 73, 67, 69, 69, 66, 68, 79, 99,
↪  121, 143, 165,
180, 174, 148, 131, 122, 112, 115, 120, 121, 126, 122, 108, 87, 72,
↪  71, 73, 79, 81, 83, 86,
83, 77, 70, 71, 85, 100, 112, 118, 130, 146, 154, 166, 174, 172, 172,
↪  161, 147, 146, 153, 157,
161, 165, 168, 170, 162, 138, 122, 121, 121, 123, 128, 138, 151, 161,
↪  165, 161, 153, 150, 149, 147,
136, 129, 140, 150, 156, 176, 194, 193, 179, 168, 167, 174, 185, 188,
↪  181, 174, 164, 156, 156, 155,
163, 185, 210, 224, 229, 235, 233, 215, 195, 176, 168, 170, 171, 168,
↪  162, 162, 163, 165, 174, 181,
184, 172, 151, 135, 125, 125, 132, 138, 139, 139, 139, 133, 121, 116,
↪  117, 115, 104, 94, 94, 94,
92, 90, 82, 70, 64, 69, 77, 82, 87, 85, 85, 92, 97, 105, 112, 108,
↪  103, 107, 116, 122,
121, 115, 101, 87, 80, 71, 67, 72, 70, 68, 78, 82, 78, 78, 79, 81, 79,
↪  68, 59, 57,
53, 60, 83, 106, 125, 146, 174, 192, 188, 186, 180, 161, 155, 157,
↪  158, 156, 152, 148, 131, 117,
111, 100, 97, 101, 104, 107, 110, 113, 112, 108, 106, 108, 122, 130,
↪  141, 164, 175, 180, 185, 186,
```

```
186, 182, 174, 167, 155, 150, 154, 155, 143, 132, 136, 139, 127, 114,
↪  108, 107, 104, 103, 114, 120,
124, 131, 134, 132, 123, 115, 109, 101, 108, 130, 144, 154, 161, 171,
↪  184, 184, 171, 155, 147, 155,
165, 165, 151, 142, 144, 136, 137, 152, 158, 162, 177, 200, 209, 206,
↪  201, 181, 163, 159, 154, 154,
151, 146, 161, 176, 170, 168, 175, 181, 176, 160, 148, 141, 138, 140,
↪  140, 139, 140, 148, 155, 152,
146, 135, 123, 111, 103, 110, 113, 100, 81, 62, 55, 52, 40, 33, 38,
↪  60, 86, 95, 99, 106,
111, 113, 105, 91, 87, 94, 101, 106, 103, 90, 76, 67, 63, 68, 72, 68,
↪  63, 58, 68, 86,
82, 68, 60, 56, 53, 45, 37, 40, 58, 77, 92, 110, 128, 149, 169, 174,
↪  161, 151, 144, 139,
142, 146, 146, 147, 142, 132, 129, 127, 116, 99, 94, 103, 113, 119,
↪  122, 128, 133, 128, 119, 118,
132, 160, 193, 215, 221, 222, 226, 224, 217, 211, 200, 181, 166, 158,
↪  152, 148, 139, 125, 118, 118,
119, 122, 123, 123, 124, 126, 127, 124, 127, 141, 143, 131, 118, 107,
↪  110, 127, 146, 159, 163, 165,
166, 164, 164, 160, 146, 131, 124, 135, 147, 145, 140, 138, 130, 124,
↪  130, 136, 145, 163, 177, 182,
181, 179, 177, 169, 159, 154, 155, 165, 176, 184, 195, 195, 183, 173,
↪  163, 156, 158, 160, 159, 165,
171, 164, 154, 154, 159, 167, 170, 167, 157, 141, 128, 120, 115, 111,
↪  102, 95, 87, 64, 50, 49,
45, 54, 77, 101, 123, 136, 139, 136, 128, 119, 112, 105, 101, 110,
↪  123, 112, 94, 88, 78, 72,
83, 89, 80, 69, 65, 57, 58, 64, 59, 53, 39, 16, 18, 36, 46, 66, 92,
↪  107, 119, 135,
```

```
145, 150, 160, 158, 147, 145, 144, 150, 160, 153, 150, 150, 140, 128,
↪  120, 116, 104, 91, 88, 90,
106, 123, 123, 123, 114, 100, 105, 119, 142, 181, 211, 222, 220, 214,
↪  208, 204, 201, 186, 171, 166,
162, 154, 138, 128, 120, 101, 93, 94, 103, 119, 117, 109, 109, 112,
↪  119, 121, 121, 124, 122, 119,
117, 124, 142, 158, 174, 183, 173, 168, 165, 149, 135, 132, 126, 119,
↪  124, 127, 125, 133, 126, 111,
116, 123, 127, 135, 145, 157, 167, 174, 176, 177, 182, 181, 184, 194,
↪  194, 198, 213, 219, 219, 219,
206, 184, 164, 153, 154, 163, 166, 162, 165, 164, 154, 154, 160, 161,
↪  165, 166, 158, 146, 140, 130,
122, 121, 109, 95, 89, 74, 61, 65, 74, 88, 110, 132, 149, 159, 149,
↪  124, 107, 99, 91, 92,
98, 101, 101, 90, 81, 84, 86, 82, 82, 80, 68, 58, 56, 53, 47, 42, 37,
↪  35, 35, 30,
28, 31, 40, 56, 74, 91, 99, 98, 101, 110, 114, 111, 110, 119, 127,
↪  133, 140, 139, 128, 118,
116, 109, 94, 87, 83, 79, 89, 110, 119, 116, 117, 117, 114, 117, 116,
↪  119, 137, 164, 191, 204,
192, 180, 180, 175, 161, 152, 149, 139, 128, 122, 111, 98, 89, 88, 93,
↪  97, 94, 98, 104, 101,
107, 119, 117, 121, 140, 152, 157, 164, 165, 171, 183, 190, 194, 191,
↪  182, 172, 166, 154, 137, 132,
134, 134, 138, 141, 130, 120, 123, 123, 120, 117, 109, 110, 125, 150,
↪  168, 164, 163, 179, 196, 210,
218, 220, 224, 227, 230, 238, 237, 218, 205, 202, 194, 189, 188, 184,
↪  181, 181, 182, 174, 162, 161,
168, 181, 194, 187, 176, 170, 156, 151, 143, 127, 125, 125, 116, 103,
↪  94, 95, 107, 124, 145, 161,
```

```
159, 151, 153, 145, 123, 106, 95, 85, 82, 86, 87, 78, 74, 79, 79, 73,
↪  64, 58, 62, 62,
64, 59, 43, 41, 53, 59, 57, 51, 47, 49, 71, 99, 107, 105, 98, 87, 93,
↪  109, 117, 114,
110, 113, 120, 132, 136, 131, 129, 123, 112, 105, 97, 95, 103, 115,
↪  123, 125, 130, 140, 145, 145,
137, 134, 142, 147, 157, 176, 187, 183, 171, 157, 142, 132, 132, 133,
↪  131, 127, 111, 92, 84, 83,
81, 72, 63, 60, 69, 90, 112, 122, 115, 112, 124, 131, 135, 144, 145,
↪  149, 161, 174, 184, 181,
171, 160, 148, 143, 138, 127, 119, 119, 126, 130, 120, 107, 100, 99,
↪  104, 109, 105, 95, 95, 106,
121, 138, 149, 158, 165, 170, 183, 200, 214, 227, 233, 236, 236, 225,
↪  214, 206, 194, 188, 181, 173,
174, 175, 176, 174, 164, 159, 159, 153, 149, 150, 154, 166, 172, 160,
↪  146, 136, 130, 131, 127, 112,
96, 91, 97, 107, 117, 125, 125, 120, 119, 120, 119, 112, 96, 80, 70,
↪  65, 67, 69, 63, 63,
61, 48, 41, 46, 58, 73, 84, 91, 90, 85, 88, 88, 84, 79, 74, 84, 94,
↪  99, 116, 128,
122, 111, 104, 99, 96, 101, 117, 128, 127, 124, 130, 139, 139, 138,
↪  133, 115, 105, 115, 131, 141,
147, 149, 147, 149, 159, 162, 159, 158, 155, 156, 160, 162, 168, 168,
↪  163, 166, 168, 148, 121, 118,
128, 127, 127, 124, 108, 93, 85, 76, 67, 56, 58, 82, 102, 108, 122,
↪  137, 135, 131, 134, 133,
135, 145, 158, 165, 166, 167, 161, 152, 151, 147, 140, 128, 117, 116,
↪  116, 113, 117, 115, 108, 105,
105, 99, 91, 98, 112, 115, 120, 133, 145, 158, 171, 182, 188, 190,
↪  201, 222, 235, 239, 241, 245,
```

```
239, 230, 224, 214, 196, 178, 176, 183, 191, 200, 199, 189, 175, 159,
↪  148, 140, 135, 150, 173, 185,
185, 178, 157, 135, 125, 115, 104, 101, 102, 100, 95, 93, 91, 81, 73,
↪  74, 80, 87, 77, 72,
72, 56, 45, 47, 45, 42, 39, 37, 27, 16, 24, 54, 82, 92, 91, 94, 92,
↪  88, 84, 75,
73, 80, 93, 113, 127, 129, 117, 107, 100, 90, 85, 82, 83, 101, 122,
↪  134, 139, 141, 137, 130,
126, 116, 108, 120, 145, 168, 176, 173, 165, 151, 142, 144, 146, 149,
↪  150, 155, 163, 158, 152, 144,
136, 140, 142, 136, 132, 126, 123, 122, 118, 120, 120, 115, 115, 108,
↪  91, 82, 85, 98, 120, 144,
163, 171, 156, 138, 134, 129, 123, 128, 140, 155, 166, 169, 157, 140,
↪  132, 125, 121, 121, 115, 110,
115, 113, 101, 92, 87, 83, 83, 86, 85, 81, 88, 106, 120, 127, 126,
↪  121, 126, 144, 161, 167,
166, 173, 196, 217, 224, 226, 220, 211, 213, 217, 208, 200, 197, 187,
↪  184, 190, 194, 198, 199, 186,
174, 167, 160, 160, 173, 188, 198, 206, 200, 181, 169, 152, 130, 119,
↪  110, 110, 125, 126, 115, 102,
83, 68, 66, 72, 75, 79, 75, 61, 51, 49, 43, 38, 40, 41, 35, 32, 39,
↪  48, 59, 72,
87, 98, 91, 87, 92, 84, 71, 66, 73, 91, 110, 123, 127, 113, 95, 88,
↪  82, 76, 76, 84,
96, 109, 119, 123, 129, 127, 119, 121, 122, 126, 139, 148, 162, 173,
↪  169, 158, 142, 127, 121, 121,
124, 125, 134, 149, 154, 158, 145, 119, 114, 119, 119, 128, 140, 142,
↪  138, 130, 123, 125, 128, 125,
128, 137, 141, 147, 157, 156, 149, 158, 174, 171, 153, 139, 127, 123,
↪  130, 136, 151, 166, 166, 158,
```

```
144, 130, 122, 113, 110, 115, 119, 122, 122, 112, 96, 85, 82, 79, 82,
↪  95, 108, 112, 107, 103,
111, 126, 125, 113, 118, 129, 136, 149, 162, 173, 184, 194, 198, 186,
↪  179, 191, 197, 190, 190, 198,
201, 193, 187, 185, 188, 197, 202, 206, 212, 213, 207, 198, 191, 193,
↪  203, 203, 191, 178, 160, 137,
123, 116, 112, 120, 129, 122, 103, 88, 80, 63, 47, 46, 56, 65, 54, 41,
↪  42, 45, 46, 47,
54, 64, 60, 61, 77, 79, 73, 90, 111, 109, 98, 100, 102, 93, 88, 95,
↪  116, 137, 141, 140,
137, 122, 104, 97, 99, 101, 106, 110, 115, 127, 130, 129, 139, 148,
↪  146, 143, 147, 144, 139, 149,
168, 172, 155, 139, 129, 117, 107, 110, 121, 133, 144, 153, 154, 138,
↪  118, 103, 94, 96, 112, 136,
150, 143, 129, 125, 129, 133, 145, 158, 156, 149, 151, 156, 160, 162,
↪  160, 153, 143, 131, 117, 107,
107, 110, 113, 123, 134, 131, 118, 111, 112, 104, 90, 99, 118, 121,
↪  123, 131, 121, 104, 101, 112,
127, 132, 127, 126, 128, 125, 118, 125, 133, 123, 111, 107, 113, 124,
↪  133, 150, 170, 179, 181, 183,
184, 180, 176, 182, 188, 193, 198, 205, 213, 211, 209, 215, 222, 226,
↪  225, 228, 232, 224, 204, 190,
192, 199, 196, 193, 186, 168, 151, 136, 126, 125, 126, 123, 112, 95,
↪  77, 64, 53, 40, 32, 34,
43, 47, 44, 40, 42, 48, 50, 48, 51, 50, 47, 52, 57, 63, 75, 81, 88,
↪  95, 96, 98,
97, 88, 89, 111, 137, 148, 149, 144, 125, 109, 101, 101, 109, 120,
↪  123, 124, 135, 142, 143, 153,
160, 156, 152, 148, 141, 135, 131, 128, 129, 126, 113, 105, 108, 101,
↪  88, 91, 99, 104, 114, 118,
```

```
116, 114, 98, 78, 78, 91, 101, 119, 134, 135, 134, 135, 142, 157, 162,
↪  155, 150, 147, 140, 140,
151, 154, 145, 133, 119, 115, 116, 102, 91, 99, 106, 106, 109, 104,
↪  92, 89, 87, 82, 85, 91,
102, 123, 134, 131, 129, 121, 116, 126, 133, 132, 127, 121, 113, 114,
↪  121, 124, 123, 118, 110, 113,
119, 114, 117, 137, 153, 158, 160, 159, 158, 158, 156, 152, 154, 163,
↪  176, 193, 210, 213, 206, 205,
209, 214, 223, 226, 220, 217, 209, 199, 198, 192, 182, 178, 178, 174,
↪  161, 151, 153, 150, 145, 136,
119, 104, 86, 68, 60, 49, 44, 52, 61, 68, 69, 58, 52, 56, 57, 57, 60,
↪  56, 49, 51,
59, 69, 74, 80, 95, 115, 130, 134, 133, 134, 134, 145, 168, 173, 168,
↪  168, 162, 149, 145, 147,
154, 168, 178, 183, 184, 174, 166, 168, 175, 174, 160, 154, 152, 138,
↪  131, 133, 126, 111, 100, 94,
97, 93, 86, 97, 105, 101, 102, 107, 108, 102, 92, 87, 89, 97, 117,
↪  149, 170, 162, 148, 150,
159, 168, 176, 176, 165, 147, 133, 131, 137, 138, 130, 122, 119, 117,
↪  114, 110, 104, 104, 107, 101,
96, 91, 82, 81, 83, 79, 82, 100, 122, 141, 147, 140, 132, 121, 113,
↪  119, 125, 122, 111, 103,
108, 113, 117, 124, 120, 116, 119, 117, 128, 142, 141, 148, 159, 161,
↪  163, 169, 171, 171, 176, 183,
186, 181, 187, 202, 205, 198, 192, 192, 193, 195, 203, 204, 191, 184,
↪  182, 179, 189, 195, 185, 180,
176, 165, 159, 161, 162, 155, 142, 130, 121, 104, 78, 66, 65, 46, 34,
↪  51, 65, 66, 62, 48,
35, 34, 36, 35, 42, 49, 49, 53, 57, 57, 66, 76, 84, 95, 109, 120, 130,
↪  137, 129, 122,
```

```
133, 143, 148, 152, 152, 157, 165, 164, 168, 183, 186, 181, 188, 189,
↪  176, 171, 173, 173, 165, 149,
140, 139, 139, 134, 127, 118, 95, 78, 82, 82, 75, 76, 69, 64, 78, 90,
↪  85, 80, 79, 83,
98, 117, 131, 145, 153, 153, 153, 151, 151, 155, 157, 152, 146, 146,
↪  141, 133, 136, 134, 125, 123,
121, 123, 136, 137, 117, 102, 94, 84, 90, 98, 87, 82, 93, 104, 108,
↪  115, 130, 138, 135, 128,
125, 122, 114, 113, 112, 101, 99, 107, 109, 110, 114, 117, 113, 103,
↪  102, 118, 141, 151, 146, 144,
151, 165, 181, 181, 176, 181, 179, 177, 185, 190, 188, 185, 182, 174,
↪  172, 175, 172, 177, 185, 187,
187, 183, 181, 192, 203, 202, 185, 175, 188, 200, 200, 190, 175, 162,
↪  152, 146, 141, 128, 112, 104,
94, 76, 69, 77, 78, 73, 63, 51, 36, 24, 28, 39, 47, 54, 55, 54, 59,
↪  67, 74, 74,
73, 86, 103, 113, 121, 126, 125, 127, 136, 142, 145, 155, 169, 183,
↪  185, 178, 170, 166, 172, 183,
190, 187, 172, 166, 172, 167, 161, 158, 145, 134, 126, 115, 108, 94,
↪  74, 66, 67, 63, 58, 61,
62, 65, 74, 73, 66, 69, 85, 103, 117, 131, 137, 139, 148, 152, 146,
↪  138, 138, 144, 140, 133,
140, 150, 140, 128, 129, 124, 116, 118, 121, 124, 128, 128, 116, 96,
↪  86, 91, 105, 116, 115, 117,
127, 128, 130, 140, 147, 145, 138, 123, 118, 126, 125, 112, 102, 101,
↪  106, 112, 110, 112, 113, 102,
92, 93, 102, 117, 141, 164, 165, 155, 161, 173, 176, 178, 181, 192,
↪  198, 188, 181, 185, 183, 172,
165, 159, 158, 164, 163, 158, 164, 171, 169, 164, 164, 176, 190, 194,
↪  185, 173, 169, 179, 189, 186,
```

```
179, 169, 157, 141, 127, 126, 124, 109, 93, 80, 74, 78, 77, 65, 46,
↪  33, 32, 34, 33, 38,
51, 62, 58, 52, 59, 68, 69, 68, 79, 96, 109, 117, 120, 127, 134, 139,
↪  148, 148, 152, 175,
191, 190, 190, 191, 183, 166, 160, 171, 189, 199, 189, 169, 157, 161,
↪  171, 173, 166, 155, 144, 129,
112, 107, 107, 96, 86, 88, 86, 83, 82, 83, 89, 88, 89, 96, 95, 106,
↪  133, 152, 149, 130,
123, 127, 122, 123, 131, 132, 132, 130, 124, 120, 120, 121, 123, 123,
↪  123, 128, 132, 131, 132, 130,
118, 103, 97, 113, 142, 163, 164, 149, 132, 127, 133, 142, 146, 146,
↪  137, 116, 102, 103, 103, 97,
95, 96, 100, 108, 106, 98, 97, 94, 87, 80, 82, 102, 138, 167, 171,
↪  159, 151, 154, 168, 183,
189, 188, 188, 182, 172, 168, 169, 166, 154, 148, 153, 156, 155, 154,
↪  159, 170, 169, 156, 151, 167,
193, 208, 199, 177, 169, 177, 188, 193, 191, 183, 170, 155, 139, 120,
↪  110, 105, 96, 89, 83, 75,
71, 60, 42, 32, 28, 22, 21, 30, 51, 71, 70, 53, 42, 42, 51, 63, 75,
↪  95, 116, 127,
128, 125, 132, 145, 153, 163, 171, 176, 184, 193, 196, 190, 183, 174,
↪  159, 157, 173, 185, 183, 170,
160, 165, 177, 184, 176, 157, 139, 125, 107, 96, 97, 97, 98, 100, 95,
↪  93, 97, 99, 103, 100,
89, 89, 98, 113, 137, 148, 145, 129, 106, 97, 96, 97, 102, 108, 118,
↪  124, 119, 112, 110, 110,
116, 126, 125, 125, 139, 151, 153, 150, 137, 126, 128, 138, 156, 168,
↪  169, 159, 150, 145, 141, 141,
145, 140, 125, 109, 95, 88, 86, 86, 92, 96, 97, 99, 99, 97, 92, 80,
↪  67, 73, 96, 117,
```

```
139, 157, 162, 164, 160, 158, 164, 167, 169, 172, 171, 167, 160, 154,
↪  148, 144, 142, 142, 146, 151,
153, 160, 172, 172, 168, 172, 173, 179, 192, 192, 188, 186, 178, 182,
↪  193, 194, 194, 185, 169, 152,
133, 115, 98, 86, 84, 80, 73, 67, 56, 43, 37, 31, 24, 27, 32, 34, 51,
↪  68, 65, 56,
54, 57, 63, 74, 91, 106, 123, 141, 154, 164, 170, 170, 173, 186, 193,
↪  193, 199, 200, 196, 187,
172, 163, 158, 155, 167, 174, 168, 166, 166, 165, 174, 180, 167, 151,
↪  143, 131, 115, 111, 110, 106,
106, 116, 126, 129, 122, 111, 104, 95, 85, 95, 113, 117, 122, 125,
↪  116, 102, 86, 76, 79, 85,
86, 88, 93, 93, 95, 101, 98, 99, 109, 115, 120, 130, 136, 138, 140,
↪  144, 147, 148, 146, 144,
153, 160, 159, 155, 146, 138, 137, 135, 127, 118, 112, 105, 97, 93,
↪  87, 79, 83, 97, 100, 90,
87, 91, 83, 63, 62, 78, 90, 113, 150, 172, 169, 159, 153, 154, 160,
↪  165, 166, 162, 158, 156,
156, 157, 154, 155, 161, 162, 162, 169, 169, 168, 175, 178, 180, 194,
↪  205, 202, 197, 199, 198, 188,
178, 182, 202, 210, 190, 168, 158, 141, 123, 118, 110, 91, 77, 76, 70,
↪  56, 51, 50, 40, 30,
28, 32, 33, 34, 50, 64, 62, 69, 80, 75, 78, 94, 105, 121, 147, 175,
↪  196, 198, 197, 206,
214, 217, 218, 216, 210, 199, 191, 175, 160, 165, 172, 165, 164, 171,
↪  172, 165, 158, 157, 161, 158,
152, 151, 142, 124, 117, 118, 115, 122, 141, 154, 148, 131, 120, 112,
↪  105, 102, 108, 112, 104, 100,
105, 104, 98, 89, 76, 61, 53, 58, 62, 56, 56, 66, 72, 81, 98, 105,
↪  101, 107, 125, 134,
```

```
135, 135, 134, 136, 139, 142, 141, 138, 140, 143, 139, 128, 118, 114,
↪  108, 101, 101, 106, 103, 89,
75, 66, 61, 67, 83, 95, 94, 87, 76, 61, 49, 51, 62, 72, 89, 115, 139,
↪  149, 146, 141,
139, 139, 141, 139, 131, 129, 132, 138, 143, 148, 156, 163, 164, 166,
↪  174, 178, 172, 170, 178, 194,
217, 227, 220, 217, 225, 229, 225, 219, 213, 209, 204, 200, 194, 181,
↪  169, 153, 132, 121, 110, 100,
98, 100, 103, 94, 74, 59, 53, 49, 47, 47, 51, 57, 73, 93, 98, 94, 91,
↪  95, 105, 118,
141, 166, 183, 189, 192, 200, 200, 204, 214, 209, 204, 197, 181, 174,
↪  165, 156, 160, 158, 151, 153,
156, 155, 158, 157, 156, 164, 165, 163, 163, 151, 139, 140, 141, 145,
↪  151, 155, 158, 160, 162, 151,
127, 111, 112, 114, 110, 100, 90, 90, 98, 100, 94, 84, 70, 59, 50, 45,
↪  47, 51, 59, 76,
90, 104, 112, 111, 114, 131, 155, 165, 155, 145, 141, 141, 146, 140,
↪  125, 124, 129, 126, 116, 110,
106, 98, 90, 85, 82, 82, 78, 64, 47, 43, 51, 59, 65, 77, 90, 89, 73,
↪  58, 57, 70,
87, 94, 92, 106, 127, 138, 142, 137, 133, 135, 131, 123, 117, 115,
↪  120, 126, 130, 139, 146, 147,
148, 152, 156, 164, 168, 166, 181, 209, 230, 231, 222, 225, 236, 237,
↪  231, 220, 210, 204, 196, 188,
174, 156, 143, 130, 114, 101, 91, 80, 69, 68, 74, 77, 73, 59, 47, 46,
↪  49, 55, 57, 56,
63, 88, 113, 119, 120, 128, 131, 137, 153, 166, 179, 189, 193, 199,
↪  200, 198, 200, 198, 191, 183,
174, 165, 156, 156, 158, 151, 143, 147, 153, 155, 155, 150, 147, 162,
↪  180, 186, 177, 159, 152, 155,
```

```
153, 152, 154, 149, 146, 160, 159, 135, 120, 112, 110, 113, 107, 97,
↪  85, 77, 84, 94, 88, 70,
61, 59, 52, 51, 53, 48, 56, 94, 130, 138, 134, 129, 134, 150, 162,
↪  168, 164, 151, 145, 144,
137, 132, 128, 126, 129, 127, 129, 130, 115, 102, 95, 80, 74, 82, 82,
↪  72, 61, 59, 61, 62,
74, 99, 113, 106, 92, 85, 84, 86, 91, 96, 99, 110, 129, 138, 140, 140,
↪  138, 137, 133, 124,
120, 121, 126, 134, 140, 137, 136, 147, 161, 171, 176, 174, 170, 189,
↪  221, 237, 238, 236, 232, 231,
231, 232, 228, 216, 201, 190, 179, 164, 155, 150, 131, 111, 101, 90,
↪  82, 72, 64, 68, 74, 75,
76, 71, 61, 54, 49, 50, 54, 58, 78, 116, 136, 137, 132, 130, 134, 140,
↪  153, 170, 175, 185,
197, 202, 209, 210, 197, 185, 184, 181, 171, 161, 154, 151, 147, 143,
↪  141, 144, 151, 156, 160, 173,
190, 200, 205, 197, 180, 174, 169, 164, 165, 160, 152, 154, 155, 154,
↪  149, 137, 122, 106, 94, 90,
86, 80, 67, 56, 64, 72, 66, 57, 48, 40, 35, 34, 46, 70, 87, 99, 118,
↪  129, 126, 129,
135, 143, 149, 147, 147, 141, 129, 127, 132, 131, 121, 111, 120, 133,
↪  132, 129, 121, 105, 90, 80,
81, 91, 93, 85, 82, 88, 98, 109, 117, 122, 118, 112, 114, 111, 99,
↪  101, 102, 87, 78, 90,
113, 133, 140, 136, 128, 117, 109, 109, 109, 108, 113, 123, 131, 132,
↪  135, 141, 148, 160, 171, 178,
186, 196, 207, 213, 219, 225, 217, 207, 209, 214, 207, 188, 181, 180,
↪  167, 155, 151, 138, 110, 87,
80, 77, 68, 64, 65, 68, 71, 71, 69, 67, 62, 59, 64, 68, 61, 58, 78,
↪  107, 130, 145,
```

```
145, 139, 141, 145, 151, 164, 181, 196, 205, 212, 214, 205, 191, 181,
↪  181, 187, 191, 183, 166, 156,
155, 154, 143, 141, 157, 175, 188, 199, 205, 206, 202, 191, 181, 170,
↪  162, 164, 167, 155, 140, 143,
147, 142, 136, 122, 100, 83, 77, 83, 89, 79, 58, 47, 52, 61, 63, 53,
↪  45, 47, 51, 59,
71, 75, 75, 84, 97, 105, 107, 107, 112, 121, 129, 134, 138, 137, 124,
↪  116, 118, 116, 105, 99,
112, 134, 149, 145, 129, 116, 103, 99, 109, 118, 118, 115, 121, 129,
↪  129, 128, 123, 114, 113, 114,
115, 120, 117, 107, 106, 107, 104, 104, 115, 125, 127, 125, 121, 123,
↪  125, 119, 116, 113, 110, 124,
147, 152, 151, 165, 173, 173, 188, 202, 203, 203, 210, 217, 221, 220,
↪  208, 199, 204, 208, 202, 196,
186, 176, 164, 147, 129, 111, 93, 81, 74, 70, 63, 56, 55, 57, 70, 78,
↪  74, 67, 61, 55,
55, 58, 59, 55, 59, 85, 115, 130, 139, 148, 149, 153, 170, 182, 183,
↪  185, 188, 189, 188, 190,
195, 188, 181, 185, 188, 180, 168, 160, 163, 168, 161, 161, 179, 190,
↪  194, 199, 200, 198, 195, 192,
177, 158, 155, 158, 157, 151, 140, 134, 132, 122, 105, 94, 85, 80, 87,
↪  94, 91, 81, 63, 49,
55, 69, 78, 77, 72, 75, 90, 101, 100, 87, 74, 73, 87, 106, 111, 107,
↪  110, 114, 117, 126,
132, 122, 105, 92, 92, 99, 99, 95, 106, 122, 125, 128, 126, 112, 113,
↪  135, 144, 134, 123, 119,
119, 119, 117, 110, 106, 106, 103, 104, 114, 122, 127, 122, 112, 109,
↪  106, 108, 117, 116, 113, 124,
132, 131, 134, 136, 126, 114, 108, 122, 152, 170, 167, 164, 173, 184,
↪  184, 182, 182, 189, 202, 210,
```

```
212, 209, 202, 203, 208, 205, 200, 196, 188, 175, 163, 158, 151, 131,
↪  104, 82, 74, 72, 70, 68,
67, 71, 80, 83, 79, 70, 64, 58, 52, 51, 54, 60, 74, 89, 104, 127, 151,
↪  156, 150, 161,
179, 178, 164, 159, 168, 181, 189, 190, 190, 187, 181, 182, 187, 191,
↪  195, 198, 193, 177, 166, 177,
195, 196, 186, 184, 189, 187, 182, 185, 189, 178, 158, 143, 140, 139,
↪  130, 119, 112, 106, 96, 86,
85, 91, 96, 95, 87, 74, 64, 69, 81, 82, 78, 82, 91, 92, 92, 97, 97,
↪  88, 73, 66,
78, 99, 112, 118, 118, 109, 104, 111, 116, 113, 110, 107, 103, 99, 93,
↪  96, 104, 106, 113, 125,
128, 128, 135, 143, 145, 135, 120, 111, 108, 106, 104, 100, 96, 94,
↪  97, 104, 109, 115, 117, 114,
114, 107, 98, 105, 116, 115, 114, 122, 122, 114, 113, 121, 126, 118,
↪  112, 121, 136, 146, 151, 159,
172, 175, 166, 165, 175, 185, 190, 196, 203, 198, 192, 195, 195, 190,
↪  186, 185, 181, 171, 160, 156,
155, 141, 109, 86, 77, 74, 78, 83, 83, 83, 82, 83, 82, 77, 70, 66, 61,
↪  54, 53, 67,
84, 93, 103, 120, 133, 143, 154, 164, 173, 173, 156, 150, 162, 174,
↪  178, 180, 180, 179, 179, 185,
195, 199, 197, 196, 189, 184, 190, 204, 203, 186, 177, 183, 187, 186,
↪  187, 191, 194, 187, 167, 147,
136, 133, 130, 125, 121, 116, 108, 101, 95, 90, 89, 87, 85, 86, 90,
↪  100, 100, 83, 79, 93,
100, 93, 85, 80, 81, 83, 80, 80, 90, 98, 100, 106, 108, 108, 108, 107,
↪  106, 109, 113, 108,
100, 95, 92, 98, 103, 102, 115, 138, 146, 146, 145, 143, 150, 150,
↪  131, 108, 94, 91, 95, 101,
```

```
103, 104, 104, 94, 88, 93, 96, 99, 104, 104, 105, 115, 119, 108, 107,
↪  117, 122, 116, 111, 125,
147, 152, 143, 130, 129, 140, 152, 165, 173, 173, 172, 173, 176, 180,
↪  189, 193, 189, 181, 179, 190,
199, 190, 181, 177, 165, 149, 138, 132, 133, 128, 107, 88, 81, 84, 86,
↪  83, 79, 77, 78, 76,
68, 62, 56, 47, 40, 39, 53, 81, 98, 95, 104, 125, 143, 156, 163, 170,
↪  176, 171, 162, 156,
150, 156, 167, 169, 172, 179, 186, 192, 192, 182, 173, 175, 179, 176,
↪  179, 188, 187, 182, 183, 189,
197, 196, 185, 181, 182, 181, 175, 161, 147, 141, 133, 122, 122, 125,
↪  114, 106, 100, 85, 77, 84,
92, 89, 81, 82, 88, 89, 87, 91, 93, 77, 65, 66, 65, 70, 83, 84, 85,
↪  95, 99, 102,
111, 112, 108, 102, 96, 95, 94, 93, 89, 83, 79, 89, 104, 120, 139,
↪  152, 154, 150, 149, 150,
150, 151, 132, 104, 94, 91, 93, 107, 116, 107, 91, 83, 88, 94, 96, 95,
↪  95, 95, 99, 113,
128, 129, 131, 140, 139, 133, 138, 153, 163, 162, 156, 153, 158, 165,
↪  170, 171, 174, 183, 186, 184,
188, 190, 188, 185, 177, 174, 179, 187, 199, 201, 191, 180, 170, 157,
↪  142, 135, 135, 127, 114, 112,
117, 114, 100, 84, 82, 84, 77, 70, 64, 55, 47, 43, 38, 40, 53, 74, 94,
↪  103, 115, 143,
161, 162, 165, 165, 167, 178, 176, 161, 158, 168, 175, 181, 188, 192,
↪  193, 186, 174, 173, 180, 181,
176, 173, 171, 179, 194, 198, 196, 201, 203, 198, 190, 189, 197, 195,
↪  180, 168, 158, 148, 139, 127,
119, 121, 119, 113, 105, 88, 84, 98, 97, 80, 71, 80, 96, 102, 93, 82,
↪  79, 76, 65, 61,
```

```
71, 82, 85, 88, 94, 101, 105, 104, 100, 100, 96, 87, 81, 73, 71, 78,
↪  76, 63, 59, 75,
104, 124, 124, 127, 140, 147, 143, 134, 127, 128, 124, 107, 91, 87,
↪  92, 101, 105, 98, 85, 82,
81, 74, 76, 83, 85, 86, 86, 91, 109, 127, 131, 129, 134, 138, 140,
↪  151, 161, 167, 175, 170,
163, 169, 175, 178, 183, 183, 178, 181, 189, 187, 185, 189, 187, 178,
↪  171, 178, 194, 198, 188, 172,
156, 145, 141, 138, 129, 122, 123, 121, 120, 116, 106, 103, 99, 89,
↪  81, 71, 57, 50, 51, 48,
46, 51, 61, 80, 100, 110, 113, 124, 137, 148, 161, 166, 166, 171, 169,
↪  157, 155, 160, 166, 175,
184, 186, 180, 174, 168, 162, 159, 159, 157, 155, 159, 171, 184, 190,
↪  195, 201, 201, 196, 191, 187,
188, 185, 173, 161, 152, 146, 140, 132, 125, 127, 124, 109, 93, 80,
↪  77, 86, 90, 85, 82, 87,
88, 78, 67, 64, 69, 72, 72, 80, 91, 98, 102, 103, 103, 105, 112, 117,
↪  115, 110, 98, 88,
80, 67, 70, 84, 85, 75, 74, 83, 102, 119, 121, 126, 141, 142, 132,
↪  124, 120, 123, 121, 109,
103, 105, 108, 113, 114, 106, 96, 88, 82, 76, 76, 84, 90, 94, 103,
↪  114, 126, 137, 135, 126,
128, 142, 154, 161, 168, 175, 173, 162, 159, 164, 169, 178, 187, 189,
↪  192, 188, 178, 170, 164, 164,
171, 181, 183, 179, 178, 176, 163, 148, 140, 139, 134, 126, 118, 109,
↪  108, 112, 115, 115, 110, 102,
95, 86, 79, 75, 69, 66, 62, 58, 62, 65, 72, 83, 89, 96, 106, 116, 129,
↪  141, 151, 156,
161, 168, 173, 172, 164, 162, 169, 180, 190, 194, 193, 187, 175, 167,
↪  156, 147, 160, 174, 181, 193,
```

```
200, 202, 204, 203, 201, 196, 187, 182, 179, 177, 175, 169, 160, 150,
↪  138, 135, 135, 135, 131, 116,
102, 92, 85, 90, 103, 106, 99, 93, 90, 86, 84, 78, 78, 85, 89, 96,
↪  102, 106, 107, 104,
105, 115, 120, 123, 130, 127, 109, 95, 93, 90, 86, 84, 89, 88, 82, 90,
↪  101, 108, 118, 124,
124, 121, 118, 119, 118, 122, 128, 122, 110, 106, 108, 107, 103, 99,
↪  93, 87, 84, 85, 83, 78,
80, 85, 93, 107, 114, 121, 134, 138, 138, 141, 149, 159, 164, 170,
↪  175, 172, 176, 189, 192, 190,
193, 194, 193, 187, 178, 169, 164, 169, 177, 181, 181, 182, 175, 161,
↪  150, 144, 138, 129, 121, 117,
111, 103, 98, 95, 96, 101, 97, 89, 83, 83, 79, 71, 65, 63, 61, 57, 55,
↪  55, 54, 61,
68, 73, 87, 105, 119, 127, 129, 132, 142, 159, 168, 162, 155, 151,
↪  155, 161, 165, 171, 175, 170,
162, 156, 151, 152, 161, 167, 170, 179, 188, 191, 197, 200, 195, 185,
↪  178, 179, 181, 185, 188, 182,
172, 161, 150, 144, 136, 129, 124, 117, 107, 99, 97, 101, 109, 108,
↪  100, 96, 94, 91, 92, 90,
86, 89, 97, 99, 97, 99, 100, 99, 99, 104, 114, 122, 126, 123, 111,
↪  100, 96, 94, 85, 76,
75, 77, 79, 90, 108, 117, 124, 129, 129, 128, 128, 129, 128, 129, 137,
↪  136, 119, 100, 90, 88,
87, 89, 93, 97, 99, 100, 89, 72, 70, 84, 97, 105, 107, 110, 122, 132,
↪  133, 135, 149, 160,
159, 162, 171, 180, 185, 186, 189, 193, 194, 190, 184, 176, 169, 167,
↪  163, 159, 163, 169, 172, 171,
165, 153, 148, 149, 139, 129, 122, 114, 109, 106, 106, 113, 112, 102,
↪  93, 89, 93, 100, 100, 97,
```

```
99, 94, 79, 60, 48, 53, 64, 72, 81, 91, 103, 118, 124, 121, 124, 133,
↪  145, 159, 167, 165,
159, 153, 156, 162, 169, 182, 188, 187, 183, 177, 171, 163, 158, 169,
↪  181, 183, 185, 186, 186, 192,
194, 188, 183, 183, 182, 180, 176, 168, 160, 161, 156, 142, 139, 135,
↪  123, 125, 127, 121, 114, 104,
98, 97, 95, 101, 102, 88, 81, 87, 87, 83, 79, 79, 81, 83, 90, 95, 94,
↪  100, 109, 113,
112, 112, 113, 112, 107, 94, 75, 56, 51, 68, 83, 95, 108, 119, 125,
↪  128, 132, 137, 135, 129,
125, 131, 137, 127, 109, 102, 93, 78, 80, 95, 106, 113, 111, 99, 85,
↪  78, 81, 89, 94, 96,
101, 109, 119, 133, 146, 150, 148, 149, 154, 163, 175, 177, 172, 175,
↪  186, 196, 201, 193, 183, 183,
179, 170, 161, 151, 147, 148, 149, 151, 147, 138, 133, 127, 121, 115,
↪  109, 104, 106, 115, 121, 113,
102, 98, 101, 106, 104, 103, 105, 104, 104, 100, 87, 69, 57, 58, 68,
↪  76, 92, 110, 113, 112,
119, 120, 122, 132, 141, 147, 156, 160, 156, 157, 169, 181, 188, 197,
↪  205, 210, 214, 207, 191, 181,
173, 171, 181, 188, 187, 192, 195, 191, 189, 187, 183, 179, 176, 174,
↪  175, 171, 160, 155, 152, 148,
143, 137, 136, 141, 148, 144, 127, 114, 111, 110, 111, 109, 104, 97,
↪  84, 76, 83, 85, 75, 76,
81, 84, 92, 91, 87, 91, 98, 105, 115, 123, 119, 107, 96, 84, 71, 63,
↪  68, 79, 84, 92,
110, 125, 130, 126, 124, 127, 126, 125, 122, 124, 131, 123, 107, 94,
↪  81, 74, 77, 86, 97, 102,
103, 101, 94, 82, 75, 74, 79, 88, 100, 114, 121, 125, 132, 135, 134,
↪  142, 150, 158, 167, 169,
```

```
171, 173, 177, 191, 204, 203, 196, 191, 191, 185, 167, 153, 150, 152,
↪  151, 153, 149, 135, 120, 112,
115, 115, 113, 122, 130, 130, 130, 130, 125, 119, 116, 113, 110, 118,
↪  127, 121, 110, 107, 96, 74,
67, 73, 73, 76, 89, 99, 98, 92, 96, 108, 109, 106, 115, 127, 135, 144,
↪  154, 159, 156, 163,
188, 211, 215, 208, 205, 205, 200, 191, 188, 193, 192, 189, 187, 180,
↪  175, 174, 177, 179, 174, 166,
161, 161, 165, 162, 150, 140, 135, 134, 144, 157, 159, 154, 147, 146,
↪  143, 126, 117, 118, 114, 110,
107, 99, 82, 70, 72, 71, 63, 61, 69, 83, 89, 85, 86, 85, 80, 86, 104,
↪  113, 105, 91,
74, 63, 62, 61, 67, 77, 83, 96, 108, 106, 104, 107, 114, 124, 133,
↪  134, 128, 121, 113, 103,
92, 82, 73, 70, 79, 93, 102, 100, 92, 91, 92, 84, 78, 79, 85, 94, 102,
↪  115, 121, 122,
135, 145, 137, 132, 141, 151, 160, 171, 178, 183, 187, 193, 207, 219,
↪  218, 212, 202, 194, 189, 175,
159, 158, 156, 154, 152, 133, 110, 104, 105, 112, 126, 132, 132, 141,
↪  144, 136, 126, 120, 122, 124,
128, 140, 145, 134, 116, 103, 94, 82, 75, 82, 92, 99, 102, 100, 95,
↪  92, 99, 112, 116, 110,
109, 118, 126, 132, 142, 153, 163, 180, 199, 210, 211, 210, 212, 213,
↪  212, 203, 193, 190, 191, 193,
193, 186, 177, 174, 177, 185, 184, 171, 159, 153, 154, 155, 148, 141,
↪  137, 138, 154, 177, 186, 183,
178, 170, 153, 134, 124, 121, 125, 128, 121, 105, 80, 65, 61, 58, 57,
↪  57, 62, 72, 79, 81,
78, 73, 74, 85, 99, 102, 90, 72, 61, 59, 59, 59, 68, 81, 94, 95, 83,
↪  87, 98, 105,
```

```
121, 129, 121, 110, 98, 93, 88, 79, 72, 66, 63, 71, 85, 93, 92, 91,
↪  93, 90, 83, 76,
81, 89, 95, 114, 133, 138, 138, 133, 127, 132, 131, 126, 133, 145,
↪  158, 168, 168, 162, 171, 195,
219, 230, 226, 217, 207, 198, 190, 174, 158, 155, 159, 162, 151, 131,
↪  116, 110, 118, 140, 153, 149,
143, 140, 139, 134, 131, 131, 130, 139, 146, 137, 129, 123, 112, 101,
↪  91, 81, 80, 88, 95, 98,
92, 84, 86, 91, 90, 96, 99, 92, 88, 96, 114, 132, 144, 149, 159, 184,
↪  204, 210, 217, 219,
213, 204, 192, 183, 182, 188, 198, 205, 205, 198, 191, 184, 178, 175,
↪  170, 156, 145, 143, 145, 143,
134, 124, 130, 158, 196, 214, 207, 194, 182, 170, 162, 152, 137, 136,
↪  143, 138, 121, 96, 77, 69,
63, 63, 69, 68, 62, 66, 77, 77, 70, 68, 71, 87, 104, 102, 89, 78, 68,
↪  69, 78, 74,
71, 80, 85, 86, 92, 98, 106, 114, 113, 109, 107, 104, 99, 97, 91, 80,
↪  71, 66, 65, 76,
87, 87, 86, 86, 84, 79, 72, 71, 79, 89, 112, 136, 145, 146, 141, 130,
↪  122, 122, 126, 119,
112, 119, 132, 140, 143, 151, 168, 189, 217, 230, 217, 200, 189, 184,
↪  180, 169, 157, 148, 147, 152,
150, 140, 133, 134, 141, 152, 159, 155, 149, 151, 154, 149, 141, 137,
↪  142, 149, 151, 143, 132, 126,
119, 111, 101, 89, 86, 88, 88, 89, 84, 84, 98, 105, 101, 97, 99, 106,
↪  113, 124, 134, 140,
148, 165, 179, 189, 200, 207, 205, 199, 194, 193, 191, 192, 195, 196,
↪  199, 205, 201, 194, 191, 181,
165, 158, 158, 153, 143, 132, 126, 127, 133, 146, 166, 181, 199, 212,
↪  200, 177, 161, 155, 154, 149,
```

```
136, 121, 119, 123, 116, 104, 87, 71, 69, 67, 65, 65, 57, 51, 58, 62,
↪  62, 70, 81, 87,
93, 100, 104, 102, 91, 83, 80, 76, 73, 72, 74, 80, 82, 87, 100, 106,
↪  104, 104, 109, 108,
100, 93, 84, 70, 64, 67, 70, 66, 67, 70, 70, 68, 61, 55, 60, 79, 97,
↪  99, 105, 119,
127, 133, 138, 131, 123, 122, 122, 115, 104, 101, 114, 130, 141, 155,
↪  171, 189, 205, 209, 201, 182,
162, 156, 159, 159, 155, 140, 132, 143, 150, 153, 155, 143, 138, 148,
↪  156, 156, 144, 135, 137, 140,
142, 146, 147, 152, 158, 152, 136, 121, 110, 101, 93, 85, 81, 75, 66,
↪  71, 81, 87, 100, 109,
110, 115, 133, 147, 139, 129, 133, 143, 161, 176, 176, 183, 198, 206,
↪  209, 210, 208, 207, 205, 198,
196, 195, 192, 196, 198, 193, 186, 176, 164, 156, 158, 163, 157, 145,
↪  140, 153, 179, 193, 192, 194,
205, 211, 200, 181, 168, 157, 147, 139, 129, 120, 117, 117, 107, 90,
↪  82, 79, 72, 68, 64, 56,
46, 40, 42, 53, 71, 86, 89, 91, 106, 125, 127, 115, 102, 91, 85, 79,
↪  70, 66, 68, 78,
90, 95, 102, 113, 115, 114, 117, 116, 109, 99, 90, 88, 84, 73, 64, 58,
↪  62, 73, 82, 87,
82, 73, 79, 88, 86, 84, 90, 101, 109, 107, 105, 115, 121, 121, 126,
↪  128, 119, 112, 119, 139,
154, 162, 168, 171, 181, 192, 185, 175, 170, 161, 160, 165, 160, 152,
↪  147, 149, 158, 156, 149, 149,
148, 142, 138, 138, 142, 149, 157, 165, 167, 163, 154, 149, 149, 147,
↪  142, 132, 119, 103, 85, 70,
65, 64, 63, 72, 87, 90, 93, 106, 119, 133, 138, 137, 141, 142, 144,
↪  152, 159, 167, 171, 174,
```

```
181, 194, 204, 202, 193, 185, 179, 176, 174, 172, 173, 173, 166, 158,
↪  153, 154, 159, 165, 171, 172,
168, 166, 172, 185, 193, 194, 195, 193, 193, 201, 208, 205, 193, 177,
↪  162, 147, 126, 109, 104, 104,
101, 90, 74, 67, 67, 62, 55, 50, 51, 67, 87, 96, 103, 102, 96, 98,
↪  106, 116, 125, 121,
110, 98, 87, 78, 75, 77, 80, 84, 87, 88, 96, 110, 120, 120, 113, 110,
↪  115, 116, 104, 91,
83, 74, 72, 80, 85, 90, 99, 96, 80, 74, 74, 74, 74, 70, 66, 76, 93,
↪  96, 93, 95,
98, 105, 113, 117, 124, 131, 131, 137, 147, 152, 157, 163, 171, 184,
↪  196, 196, 187, 179, 174, 169,
161, 146, 138, 144, 150, 143, 137, 139, 137, 139, 146, 147, 155, 167,
↪  167, 163, 156, 147, 145, 146,
143, 139, 139, 128, 109, 99, 90, 74, 65, 63, 67, 83, 97, 96, 93, 101,
↪  115, 124, 131, 137,
150, 164, 169, 174, 178, 172, 167, 164, 165, 180, 194, 189, 179, 174,
↪  165, 157, 155, 153, 145, 147,
158, 158, 151, 150, 153, 164, 170, 165, 171, 185, 187, 187, 192, 195,
↪  193, 193, 206, 220, 223, 210,
187, 169, 153, 132, 114, 98, 82, 79, 84, 76, 61, 54, 50, 51, 59, 62,
↪  76, 100, 114, 114,
105, 100, 101, 104, 105, 104, 111, 113, 104, 99, 94, 82, 71, 63, 57,
↪  59, 73, 86, 92, 96,
99, 102, 104, 105, 112, 120, 116, 107, 96, 85, 79, 77, 84, 89, 88, 88,
↪  84, 75, 68, 67,
66, 65, 67, 74, 93, 104, 91, 81, 79, 85, 101, 110, 121, 138, 147, 150,
↪  155, 159, 165, 178,
188, 192, 196, 196, 189, 187, 182, 170, 157, 147, 142, 144, 156, 161,
↪  157, 155, 155, 156, 159, 154,
```

```
152, 162, 175, 178, 174, 165, 159, 152, 141, 138, 144, 148, 141, 123,
↪  104, 89, 73, 66, 73, 83,
97, 112, 113, 109, 110, 116, 127, 145, 164, 177, 184, 184, 179, 171,
↪  160, 153, 160, 170, 173, 177,
175, 162, 157, 163, 161, 155, 153, 153, 159, 159, 147, 145, 153, 157,
↪  156, 157, 164, 173, 182, 195,
205, 207, 209, 215, 219, 211, 201, 197, 185, 169, 153, 129, 104, 85,
↪  77, 80, 86, 88, 79, 65,
55, 50, 47, 54, 76, 102, 115, 116, 111, 106, 103, 99, 93, 92, 100,
↪  105, 99, 92, 79, 62,
53, 49, 52, 62, 74, 86, 86, 81, 85, 94, 101, 106, 108, 108, 105, 101,
↪  96, 92, 89, 84,
79, 82, 89, 89, 84, 78, 70, 66, 63, 58, 63, 72, 77, 79, 74, 71, 76,
↪  86, 95, 98,
105, 123, 144, 164, 179, 181, 183, 187, 181, 178, 180, 184, 191, 188,
↪  174, 167, 161, 155, 153, 153,
160, 168, 163, 157, 154, 150, 145, 137, 139, 156, 175, 182, 178, 171,
↪  167, 162, 160, 156, 150, 149,
144, 130, 110, 95, 88, 88, 94, 100, 116, 133, 127, 115, 114, 121, 133,
↪  143, 153, 168, 180, 185,
178, 169, 167, 164, 163, 167, 166, 170, 180, 180, 179, 184, 185, 176,
↪  165, 153, 150, 158, 159, 154,
153, 158, 162, 156, 151, 163, 181, 201, 217, 225, 226, 221, 210, 199,
↪  187, 180, 174, 161, 141, 119,
103, 93, 87, 84, 84, 84, 76, 60, 44, 33, 30, 30, 45, 74, 98, 113, 113,
↪  98, 89, 89,
92, 95, 93, 90, 88, 81, 73, 71, 72, 72, 75, 78, 75, 78, 87, 92, 95,
↪  103, 106, 99,
91, 95, 98, 94, 96, 97, 97, 98, 101, 102, 94, 86, 82, 78, 79, 81, 77,
↪  69, 65, 65,
```

```
69, 76, 81, 84, 84, 86, 95, 98, 103, 127, 153, 169, 186, 196, 194,
↪  189, 183, 175, 171, 179,
184, 181, 181, 180, 169, 157, 153, 156, 155, 151, 149, 150, 146, 141,
↪  137, 129, 125, 140, 154, 161,
169, 172, 176, 185, 183, 173, 160, 145, 132, 121, 111, 107, 104, 103,
↪  113, 126, 125, 115, 108, 103,
106, 115, 123, 131, 141, 153, 161, 164, 167, 168, 164, 161, 166, 174,
↪  179, 184, 193, 193, 190, 193,
190, 175, 159, 148, 142, 147, 160, 168, 170, 165, 155, 143, 143, 161,
↪  172, 182, 210, 227, 220, 213,
211, 206, 193, 181, 171, 157, 147, 138, 126, 116, 104, 87, 76, 68, 62,
↪  53, 42, 33, 30, 34,
53, 84, 101, 98, 92, 88, 85, 91, 103, 105, 95, 86, 80, 83, 85, 81, 75,
↪  74, 81, 91,
90, 88, 93, 96, 96, 98, 100, 95, 91, 95, 93, 86, 93, 101, 105, 113,
↪  115, 112, 103, 87,
79, 81, 79, 76, 72, 61, 56, 58, 64, 71, 74, 74, 76, 80, 85, 94, 114,
↪  140, 148, 146,
165, 187, 187, 182, 184, 186, 187, 191, 200, 201, 186, 173, 169, 161,
↪  156, 155, 151, 149, 153, 152,
146, 141, 133, 123, 122, 138, 157, 162, 168, 182, 188, 184, 182, 179,
↪  165, 144, 133, 134, 127, 119,
119, 121, 128, 126, 113, 103, 93, 86, 87, 99, 117, 127, 138, 154, 156,
↪  145, 141, 147, 157, 167,
179, 190, 193, 193, 193, 189, 181, 175, 170, 160, 151, 142, 134, 137,
↪  153, 165, 158, 148, 145, 137,
132, 145, 160, 172, 194, 213, 219, 217, 211, 202, 189, 179, 180, 181,
↪  170, 156, 140, 122, 105, 87,
67, 58, 57, 44, 29, 27, 31, 37, 52, 75, 85, 81, 85, 91, 92, 96, 106,
↪  112, 102, 92,
```

```
98, 102, 93, 84, 83, 90, 99, 110, 119, 115, 106, 100, 91, 83, 87, 93,
↪  93, 96, 97, 96,
102, 108, 110, 114, 115, 116, 113, 101, 97, 101, 92, 83, 77, 68, 63,
↪  66, 67, 68, 73, 76,
73, 75, 86, 94, 102, 117, 128, 137, 162, 188, 196, 195, 192, 191, 190,
↪  185, 189, 193, 185, 178,
170, 158, 152, 151, 151, 150, 149, 145, 139, 133, 129, 130, 136, 148,
↪  162, 172, 181, 185, 185, 185,
183, 181, 174, 158, 150, 147, 135, 124, 118, 114, 115, 118, 115, 104,
↪  90, 78, 72, 80, 98, 111,
125, 140, 144, 144, 155, 168, 175, 184, 193, 199, 202, 199, 199, 198,
↪  188, 178, 173, 164, 153, 145,
144, 149, 153, 155, 154, 150, 144, 134, 128, 136, 151, 169, 198, 221,
↪  226, 224, 212, 198, 192, 190,
195, 201, 190, 167, 142, 120, 101, 84, 73, 66, 55, 39, 29, 31, 36, 39,
↪  46, 63, 78, 89,
96, 98, 100, 102, 104, 107, 104, 102, 107, 103, 90, 82, 84, 90, 103,
↪  115, 114, 108, 96, 84,
82, 75, 70, 74, 80, 85, 88, 96, 112, 119, 120, 123, 120, 114, 108,
↪  104, 107, 106, 94, 78,
65, 64, 66, 62, 62, 64, 63, 64, 65, 65, 63, 58, 68, 89, 103, 116, 146,
↪  179, 189, 185,
184, 186, 184, 186, 192, 193, 187, 172, 157, 154, 152, 147, 145, 144,
↪  141, 138, 137, 137, 132, 127,
137, 155, 163, 167, 176, 186, 183, 175, 171, 164, 158, 155, 151, 150,
↪  146, 135, 121, 112, 114, 117,
111, 98, 85, 77, 75, 74, 79, 90, 105, 123, 136, 147, 162, 179, 193,
↪  199, 200, 199, 199, 200,
200, 200, 194, 184, 173, 164, 155, 151, 155, 156, 145, 141, 148, 144,
↪  129, 115, 118, 131, 137, 156,
```

```
195, 222, 226, 222, 211, 200, 201, 208, 214, 212, 197, 176, 153, 128,
↪  102, 82, 70, 64, 53, 40,
39, 43, 41, 37, 41, 57, 75, 87, 90, 96, 109, 113, 107, 105, 108, 112,
↪  113, 114, 111, 105,
101, 104, 117, 120, 113, 109, 96, 78, 72, 72, 71, 73, 73, 75, 90, 109,
↪  120, 124, 126, 128,
121, 107, 102, 108, 111, 109, 105, 95, 79, 68, 65, 62, 61, 60, 60, 63,
↪  62, 56, 52, 50,
53, 65, 83, 108, 140, 167, 180, 192, 201, 194, 189, 194, 200, 204,
↪  196, 179, 168, 161, 157, 151,
144, 145, 144, 140, 145, 154, 156, 154, 157, 163, 166, 175, 184, 188,
↪  190, 185, 174, 162, 157, 159,
165, 172, 170, 157, 138, 122, 123, 123, 107, 95, 92, 83, 73, 71, 80,
↪  89, 97, 114, 140, 164,
178, 186, 196, 203, 200, 196, 198, 199, 195, 194, 196, 191, 182, 174,
↪  173, 170, 161, 152, 140, 126,
119, 117, 116, 115, 114, 118, 127, 144, 177, 207, 220, 228, 226, 212,
↪  206, 212, 212, 205, 193, 169,
145, 126, 103, 81, 64, 53, 49, 51, 51, 46, 43, 37, 38, 48, 57, 68, 84,
↪  94, 99, 100,
96, 96, 105, 115, 127, 138, 136, 129, 125, 123, 123, 120, 110, 97, 83,
↪  71, 62, 60, 63, 70,
85, 96, 99, 105, 112, 115, 115, 106, 95, 96, 102, 102, 102, 103, 100,
↪  89, 79, 75, 66, 56,
53, 51, 47, 39, 35, 35, 36, 39, 50, 61, 72, 96, 130, 161, 184, 197,
↪  198, 194, 191, 186,
183, 185, 186, 183, 181, 175, 165, 156, 155, 158, 159, 162, 166, 167,
↪  165, 163, 165, 166, 165, 172,
181, 175, 168, 165, 159, 156, 161, 171, 178, 178, 172, 157, 140, 131,
↪  123, 113, 109, 106, 92, 78,
```

```
70, 70, 82, 104, 126, 139, 145, 156, 172, 181, 185, 185, 188, 192,
↪  191, 189, 187, 183, 184, 185,
188, 193, 189, 169, 145, 127, 113, 102, 100, 109, 115, 115, 120, 130,
↪  140, 158, 184, 207, 225, 233,
233, 227, 214, 200, 192, 189, 176, 155, 135, 115, 95, 78, 64, 59, 61,
↪  54, 44, 41, 40, 40,
43, 46, 51, 64, 81, 91, 97, 101, 106, 112, 122, 133, 137, 137, 133,
↪  128, 131, 130, 126, 124,
113, 96, 81, 63, 49, 49, 60, 80, 99, 104, 98, 93, 95, 96, 91, 91, 101,
↪  110, 116, 119,
116, 110, 101, 96, 96, 95, 86, 68, 52, 43, 38, 35, 33, 31, 36, 46, 55,
↪  67, 75, 83,
110, 144, 163, 174, 181, 178, 176, 172, 165, 169, 176, 178, 182, 184,
↪  182, 176, 169, 168, 173, 174,
171, 169, 167, 165, 172, 178, 169, 165, 170, 169, 161, 157, 157, 159,
↪  167, 174, 172, 159, 150, 148,
143, 132, 129, 135, 133, 122, 114, 104, 86, 85, 100, 117, 136, 148,
↪  146, 147, 158, 168, 179, 186,
186, 188, 194, 193, 184, 179, 186, 197, 198, 193, 189, 173, 146, 123,
↪  107, 96, 96, 112, 131, 142,
151, 160, 153, 147, 162, 185, 207, 218, 217, 207, 196, 185, 170, 154,
↪  143, 131, 116, 103, 94, 84,
75, 70, 64, 53, 44, 44, 51, 53, 56, 62, 65, 75, 92, 103, 105, 102,
↪  103, 106, 110, 119,
123, 117, 117, 120, 116, 111, 116, 116, 104, 90, 74, 53, 41, 46, 61,
↪  74, 87, 97, 92, 83,
84, 91, 99, 109, 120, 127, 130, 127, 118, 114, 113, 108, 100, 94, 88,
↪  75, 58, 46, 39, 31,
24, 35, 54, 62, 71, 83, 83, 87, 116, 148, 164, 176, 181, 173, 168,
↪  168, 167, 173, 188, 196,
```

```
191, 185, 187, 195, 196, 189, 185, 179, 174, 177, 184, 190, 193, 189,
↪  172, 156, 157, 161, 157, 155,
155, 151, 150, 150, 149, 148, 145, 138, 131, 130, 138, 149, 151, 139,
↪  122, 106, 97, 103, 110, 115,
130, 147, 145, 141, 158, 183, 188, 183, 177, 171, 172, 175, 177, 185,
↪  190, 189, 184, 178, 174, 166,
151, 133, 123, 124, 130, 143, 159, 168, 171, 170, 164, 157, 162, 183,
↪  203, 207, 202, 191, 179, 169,
151, 134, 125, 111, 93, 85, 84, 83, 78, 69, 60, 57, 60, 62, 61, 63,
↪  74, 83, 88, 92,
97, 108, 113, 106, 100, 99, 95, 97, 108, 113, 111, 110, 106, 96, 89,
↪  89, 84, 70, 58, 49,
45, 53, 60, 68, 84, 93, 95, 97, 100, 108, 116, 119, 121, 123, 121,
↪  114, 108, 111, 111, 99,
85, 75, 72, 70, 61, 50, 40, 30, 30, 38, 46, 58, 73, 82, 91, 103, 119,
↪  143, 161, 165,
167, 171, 169, 163, 168, 179, 184, 186, 183, 176, 178, 181, 176, 172,
↪  178, 185, 185, 181, 181, 187,
191, 188, 181, 170, 163, 162, 159, 160, 163, 154, 139, 127, 130, 142,
↪  149, 146, 138, 134, 139, 146,
145, 134, 128, 126, 123, 118, 115, 122, 142, 157, 163, 166, 171, 178,
↪  181, 178, 173, 172, 172, 167,
170, 185, 188, 173, 158, 152, 155, 157, 151, 140, 133, 134, 141, 144,
↪  145, 155, 175, 187, 186, 177,
177, 189, 196, 193, 190, 183, 167, 146, 127, 115, 110, 107, 98, 90,
↪  87, 82, 71, 64, 66, 75,
83, 81, 79, 83, 93, 107, 114, 106, 100, 107, 110, 109, 110, 106, 101,
↪  96, 92, 96, 99, 91,
80, 71, 63, 61, 63, 61, 59, 60, 61, 57, 52, 61, 88, 116, 126, 121,
↪  115, 118, 123, 130,
```

```
136, 131, 123, 120, 114, 111, 109, 98, 85, 77, 72, 65, 54, 43, 34, 28,
↪  28, 35, 45, 63,
87, 105, 113, 112, 117, 141, 164, 170, 171, 169, 162, 161, 164, 164,
↪  166, 168, 169, 169, 168, 166,
164, 167, 175, 178, 173, 172, 174, 181, 192, 190, 180, 172, 167, 164,
↪  158, 154, 151, 145, 139, 131,
129, 134, 132, 124, 128, 136, 135, 136, 137, 129, 125, 131, 137, 138,
↪  132, 133, 154, 172, 172, 169,
172, 174, 174, 180, 185, 186, 187, 184, 180, 183, 181, 171, 167, 164,
↪  159, 154, 144, 134, 136, 140,
138, 140, 142, 151, 178, 194, 191, 185, 180, 186, 194, 189, 181, 171,
↪  155, 137, 123, 116, 112, 112,
111, 102, 93, 85, 77, 72, 71, 78, 86, 84, 86, 104, 121, 123, 113, 100,
↪  94, 98, 105, 107,
105, 100, 96, 93, 86, 79, 78, 72, 58, 52, 54, 57, 65, 69, 65, 62, 60,
↪  60, 62, 70,
92, 112, 111, 106, 114, 126, 131, 130, 130, 128, 123, 121, 121, 122,
↪  119, 110, 99, 85, 72, 63,
54, 44, 34, 27, 31, 43, 57, 71, 94, 112, 112, 109, 111, 128, 147, 148,
↪  147, 153, 155, 154,
150, 150, 154, 157, 157, 160, 167, 168, 166, 171, 176, 171, 163, 161,
↪  167, 178, 193, 196, 184, 175,
166, 163, 166, 162, 156, 149, 137, 126, 121, 125, 132, 138, 137, 132,
↪  132, 137, 144, 146, 139, 138,
143, 146, 148, 150, 157, 167, 164, 158, 159, 166, 176, 182, 188, 193,
↪  196, 198, 198, 197, 195, 184,
169, 160, 155, 149, 151, 156, 150, 140, 137, 138, 142, 147, 164, 183,
↪  187, 181, 173, 169, 171, 165,
153, 148, 142, 130, 118, 112, 116, 119, 115, 111, 108, 99, 87, 80, 78,
↪  80, 84, 98, 114, 120,
```

```
125, 125, 113, 103, 98, 94, 102, 107, 102, 102, 103, 94, 78, 66, 67,
↪  70, 66, 61, 55, 56,
70, 80, 80, 76, 67, 62, 58, 53, 69, 92, 101, 113, 123, 122, 123, 127,
↪  131, 133, 133, 132,
129, 130, 130, 123, 109, 92, 76, 63, 55, 52, 48, 43, 46, 60, 71, 76,
↪  95, 110, 105, 95,
89, 94, 112, 120, 124, 137, 146, 144, 137, 133, 139, 146, 153, 160,
↪  167, 169, 166, 165, 171, 170,
164, 166, 170, 172, 176, 181, 181, 177, 171, 165, 163, 157, 145, 134,
↪  132, 133, 125, 119, 131, 145,
145, 142, 142, 139, 139, 145, 150, 152, 155, 161, 159, 152, 156, 167,
↪  173, 175, 174, 170, 175, 181,
185, 191, 199, 205, 203, 191, 182, 177, 172, 169, 164, 161, 165, 167,
↪  163, 156, 154, 158, 157, 154,
158, 168, 175, 172, 163, 155, 156, 158, 151, 145, 143, 137, 127, 119,
↪  115, 117, 125, 127, 113, 96,
84, 81, 85, 88, 98, 123, 133, 125, 125, 128, 121, 111, 98, 90, 92, 94,
↪  92, 87, 87, 91,
85, 73, 66, 61, 61, 60, 55, 50, 49, 58, 66, 61, 51, 46, 46, 47, 54,
↪  74, 99, 115,
116, 116, 120, 121, 119, 120, 124, 130, 132, 127, 117, 108, 108, 107,
↪  93, 77, 65, 61, 59, 54,
59, 75, 83, 85, 91, 96, 94, 92, 91, 88, 93, 115, 134, 135, 131, 136,
↪  142, 141, 138, 140,
149, 158, 161, 158, 153, 159, 171, 169, 166, 174, 178, 172, 171, 174,
↪  177, 177, 168, 156, 147, 137,
131, 130, 126, 128, 137, 143, 138, 132, 135, 141, 141, 133, 131, 138,
↪  145, 148, 150, 156, 165, 169,
169, 174, 181, 185, 184, 175, 169, 175, 182, 184, 185, 189, 191, 186,
↪  178, 171, 167, 171, 176, 176,
```

```
171, 173, 179, 175, 172, 176, 173, 170, 172, 174, 176, 181, 182, 173,
↪  162, 156, 154, 154, 145, 134,
134, 134, 128, 119, 115, 114, 106, 96, 89, 85, 87, 98, 117, 135, 139,
↪  134, 134, 135, 135, 127,
112, 102, 95, 94, 99, 101, 97, 95, 95, 90, 82, 68, 52, 46, 45, 40, 35,
↪  42, 50, 47,
44, 47, 45, 46, 65, 91, 106, 115, 124, 123, 117, 115, 118, 117, 122,
↪  126, 122, 120, 122, 119,
110, 100, 90, 78, 68, 63, 57, 53, 62, 79, 86, 87, 93, 97, 96, 93, 88,
↪  86, 93, 112,
130, 130, 126, 129, 136, 143, 148, 147, 144, 143, 142, 150, 159, 162,
↪  168, 170, 171, 174, 174, 170,
169, 174, 178, 177, 175, 167, 152, 145, 141, 130, 133, 148, 156, 155,
↪  151, 141, 134, 134, 129, 121,
121, 127, 130, 136, 149, 166, 178, 184, 193, 195, 192, 191, 184, 174,
↪  172, 175, 180, 183, 184, 181,
178, 179, 172, 163, 163, 170, 172, 169, 168, 172, 171, 168, 167, 169,
↪  167, 162, 162, 165, 170, 174,
174, 168, 155, 145, 141, 137, 126, 115, 117, 125, 125, 118, 108, 100,
↪  91, 79, 74, 79, 90, 109,
128, 140, 146, 149, 153, 147, 134, 127, 118, 107, 105, 109, 112, 107,
↪  98, 96, 94, 84, 72, 57,
43, 36, 30, 22, 21, 25, 30, 40, 46, 42, 48, 70, 91, 104, 113, 126,
↪  136, 132, 128, 127,
126, 126, 126, 129, 137, 138, 132, 125, 113, 102, 92, 77, 67, 57, 52,
↪  62, 72, 80, 89, 90,
93, 95, 87, 89, 99, 99, 98, 110, 123, 129, 127, 125, 127, 133, 137,
↪  137, 140, 148, 151, 152,
156, 157, 158, 165, 172, 172, 166, 162, 166, 170, 166, 162, 162, 157,
↪  152, 149, 145, 146, 147, 152,
```

```
162, 164, 157, 150, 140, 129, 120, 112, 105, 108, 123, 145, 163, 170,
↪  177, 186, 192, 195, 192, 189,
187, 182, 181, 181, 181, 188, 191, 189, 189, 182, 173, 174, 176, 176,
↪  177, 178, 174, 165, 160, 163,
167, 164, 158, 157, 162, 165, 160, 155, 153, 142, 133, 137, 140, 132,
↪  124, 116, 110, 109, 108, 106,
102, 97, 91, 87, 83, 85, 94, 114, 135, 145, 146, 144, 142, 140, 130,
↪  119, 115, 113, 112, 109,
106, 108, 103, 91, 87, 80, 67, 54, 42, 29, 18, 9, 5, 13, 27, 38, 48,
↪  54, 56, 68,
91, 110, 117, 119, 121, 123, 129, 134, 135, 138, 144, 144, 144, 142,
↪  133, 120, 105, 92, 83, 72,
62, 58, 57, 61, 70, 78, 83, 92, 99, 101, 104, 107, 106, 107, 111, 119,
↪  129, 135, 135, 131,
129, 132, 133, 134, 136, 142, 149, 150, 149, 150, 153, 161, 169, 169,
↪  163, 157, 157, 159, 156, 152,
155, 160, 161, 164, 169, 170, 172, 172, 173, 173, 166, 152, 137, 128,
↪  119, 109, 109, 121, 134, 145,
156, 160, 161, 166, 174, 184, 191, 189, 186, 185, 178, 173, 182, 193,
↪  203, 204, 198, 197, 196, 192,
187, 182, 180, 177, 175, 173, 168, 166, 169, 171, 170, 169, 168, 165,
↪  154, 143, 134, 128, 126, 130,
133, 130, 124, 113, 101, 101, 102, 98, 98, 98, 97, 96, 99, 104, 108,
↪  119, 131, 134, 132, 130,
132, 135, 131, 123, 121, 121, 117, 110, 105, 99, 95, 89, 85, 81, 69,
↪  55, 43, 29, 17, 12,
16, 26, 33, 42, 55, 61, 67, 77, 91, 107, 118, 120, 125, 128, 130, 134,
↪  140, 145, 146, 143,
142, 136, 124, 111, 96, 80, 69, 62, 59, 57, 52, 50, 56, 65, 74, 86,
↪  96, 109, 116, };
```

Listing 13 – Serial Transmit Arduino Software

```
// Connect laser to digital pin 1, TX

// Declare variables
String myWord = "HELLO EVERYONE\r\n";
const int ledPin = LED_BUILTIN;
bool value = 0;



/************************** SETUP ********************************/
void setup() {
  // put your setup code here, to run once:
  Serial.begin(62500);
  pinMode(1, OUTPUT);
  pinMode(ledPin, OUTPUT);
}



/************************** LOOP *********************************/
void loop() {
  // put your main code here, to run repeatedly:
  // transmit each letter of the word one at a time
  for(int i = 0; i < myWord.length(); i++){
    Serial.print(myWord[i]);
    value = (bool) myWord[i];
    digitalWrite(ledPin, value);
  }

  // delay for ease of testing
  digitalWrite(ledPin, LOW);
```

```
  delay(100);
}
```

A.3 PC

Listing 14 – Image Transmission Processing Software

```
import processing.serial.*;

Serial mySerialPort;

PImage myImage;
int imageDimension;
int x = 0;
int y = 0;

void setup()
{
  size(240,240);

  //myImage = loadImage("jaxa.png");
  myImage = loadImage("index.jpg");
  myImage.loadPixels();
  imageDimension = myImage.width * myImage.height;
   println(myImage.width);
   println(imageDimension);
  mySerialPort = new Serial(this, "/dev/cu.usbmodem1412301", 9600);
}

void draw()
{
```

```
color c = myImage.pixels[y*myImage.width+x];
fill(c);
noStroke();
ellipse(x,y,1,1);
mySerialPort.write(hex(c,6)+"\n");
//println(int(c));//
x++;
if(x > myImage.width)
{
  x = 0;
  y ++;
}

//delay(200);

if(y*myImage.width + x == imageDimension)
{
  exit();
}



//imageMode(CORNERS);
//image(myImage,0,0);

//print(hex(myImage.pixels[j]));
//print(": ");
//mySerialPort.write(int(hex(myImage.pixels[j])));
```

```
  //if(j > imageDimension)
  //{
  //  j = 0;
  //}
  //delay(200);

  //println(mySerialPort.read());
}
```

Listing 15 – Image Reconstruction Testing Processing Software

```
import processing.serial.*;


Serial mySerialPort;

int x = 0;
int y = 0;

String retStr;

color c;
int RedCol;
int GreenCol;
int BlueCol;

String[] list;
int NewLineAscii = 10;

boolean readSerial = false;

void setup()
```

```
{
  size(240,240);
  mySerialPort = new Serial(this, "/dev/cu.usbserial-0001", 9600);
  mySerialPort.clear();
  delay(500);
}

void draw()
{

}

void serialEvent(Serial myPort)
{
  retStr = myPort.readStringUntil(NewLineAscii);
  if(retStr != null)
  {
    ReconstructImage(retStr);
  }
}

void ReconstructImage(String inputString)
{
      String red_hex = inputString.substring(0, 2);
      int red = unhex(red_hex);

      String green_hex = inputString.substring(2, 4);
      int green = unhex(green_hex);

      String blue_hex = inputString.substring(4, 6);
      int blue = unhex(blue_hex);
```

```
      c = color(red, green , blue);

      set(x,y,c);

      x++;


       if(x > 225) // change to 200 for jaxa image, 255 for windows
      {
        x = 0;
        y ++;
      }

}
```

Listing 16 – GUI Python Software

```
from tkinter import *
import serial
import time
from tkinter import filedialog
from PIL import ImageTk, Image
import glob, os
import wave, struct, math, random

ser = serial.Serial('COM5', 9600)        # serial link to the arduio


root = Tk()                              # create a blank window
root.title('20/20 Visions API')          # give the window a title
```

```
#
↪ ------------------------------------------------------------------------
ID = ""                                    #initialise some variables
dataInput = ""
#
↪ ------------------------------------------------------------------------

# function to convert rgb value to hex
def rgb2hex(r, g, b):
    return '#{:02x}{:02x}{:02x}'.format(r, g, b)

# save ID button function and place a label for user to see what is
↪ saved
def myClick():
    global ID
    ID = entry.get()

    myLabel = Label(root, text ="Current ID:    " + ID + "    ")
    myLabel.grid(row=2, column=1)

# save string button function and place a label for user to see what
↪ is saved
def stringClick():
    global dataInput
    dataInput = stringInput.get()
    myLabel5 = Label(root, text ="Saved String:    " + dataInput + "
    ↪ ")
    myLabel5.grid(row=4, column=1)


# transmit saved ID
```

```
def sendIDstring():
    global ID
    ser.write(ID.encode())


# transmit saved string
def sendEntrystring():
    global dataInput
    ser.write(dataInput.encode())


# open file directory and send chosen file
def findFile():
    root.filename = filedialog.askopenfilename(initialdir =
    ↪  "\mtrx3700-major\Pics", title = "Select File", filetypes =
    ↪  [("BMP files","*.bmp")])
    location = Label(root, text=root.filename)
    location.grid(row=5, column=1)
    my_image = Image.open(root.filename)

    pixels = list(my_image.convert('RGB').getdata())

    for r, g, b in pixels:
            col_hex = rgb2hex(r, g, b)
            ser.write(col_hex.encode())


# open file directory and send chosen file
def songSend():
```

```
    root.songname = filedialog.askopenfilename(initialdir =
    ↪  "\mtrx3700-major\Pics", title = "Select File", filetypes =
    ↪  [("WAV files","*.wav")])
    location44 = Label(root, text=root.songname)
    location44.grid(row=6, column=1)

    obj = wave.open(root.songname, 'r')
    frames = obj.getnframes()
    print(frames)
    for i in range(frames):
        send_data = obj.readframes(i)
        ser.write(send_data)

    obj.close()



#
↪  ------------------------------------------------------------------------
# some labels for user to see where to enter data
myLabel2 = Label(root, text="Enter ID:")
myLabel2.grid(row=1, column=0)

myLabel3 = Label(root, text="Enter String:")
myLabel3.grid(row=3, column=0)
#
↪  ------------------------------------------------------------------------
```

```
# creating a buttons
↪  ========================================================

# Save ID button
myButton = Button(root, text="Save ID", padx = 50, pady = 25,
↪  fg="black", bg = "white",  command = myClick)
myButton.grid(row=1, column=2)

# Exit program button
exitButton = Button(root, text="Exit API", padx = 10, pady = 5,
↪  fg="white", bg = "#FF0000",  command = root.quit)
exitButton.grid(row=7, column=0)

# Transmit ID button
sendID = Button(root, text="Transmit ID", padx = 40, pady = 25,
↪  fg="black", bg = "white", command= sendIDstring)
sendID.grid(row=2, column=2)

# Save string button
stringButton = Button(root, text="Save String", padx = 40, pady = 25,
↪  fg="black", bg = "white",  command = stringClick)
stringButton.grid(row=3, column=2)

# Transmit string button
sendString = Button(root, text="Transmit String", padx = 30, pady =
↪  25, fg="black", bg = "white", command= sendEntrystring)
sendString.grid(row=4, column=2)

# Open image and send button
Open_File = Button(root, text="Open and Send File", padx = 30, pady =
↪  25, fg="black", bg = "white", command= findFile)
```

```
Open_File.grid(row=5, column=2)

# Open song and send button
send_song = Button(root, text="Send Song", padx = 30, pady = 25,
↪  fg="black", bg = "white", command= songSend)
send_song.grid(row=6, column=2)

#
↪  ================================================================================
# entry wideget to enter ID/string
entry = Entry(root, width=10, bg="gray", fg="white")
entry.grid(row=1, column=1)

stringInput = Entry(root, width=10, bg="gray", fg="white")
stringInput.grid(row=3, column=1)
#
↪  ================================================================================




root.mainloop()
```

List of References

[1] Vishay Semiconductors. Infrared Receiver Module, 2016. https://datasheetspdf.com/pdf-file/700398/ETC/VS1838B/1 [Accessed: Nov 18, 2021].

[2] Quora. How do I differentiate between pulse-code modulation and quantization?, 2021. https://www.quora.com/unanswered/How-do-I-differentiate-between-pulse-code-modulation-and-quantization [Accessed: Nov 18, 2021].

[3] Lucie Dordova and Otakar Wilfert. Laser Beam Attenuation Determined by the Method of Available Optical Power in Turbulent Atmosphere. *Journal of Telecommunications and Information Technology*, 2009. https://www.itl.waw.pl/czasopisma/JTIT/2009/2/53.pdf [Accessed: Nov 18, 2021].

[4] Jaycar. ZD1948-dataSheetMain, 2021. https://www.jaycar.com.au/medias/sys_master/images/images/9592271011870/ZD1948-dataSheetMain.pdf [Accessed: Nov 18, 2021].

[5] Lewis Loflin. Photodiode Circuits Operation and Uses, 2018. https://www.bristolwatch.com/ele2/pdiodeTran.htm [Accessed: Nov 18, 2021].

[6] SHARP Corporation. Photodiode/Phototransistor Application Circuit, 1999. https://physlab.org/wp-content/uploads/2016/03/Photodiode_circuit.pdf [Accessed: Nov 18, 2021].

[7] ST Microelectronics. LOW POWER QUAD OPERATIONAL AMPLIFIERS, 2021. https://pdf1.alldatasheet.com/datasheet-pdf/view/22756/STMICROELECTRONICS/LM324N.html [Accessed: Nov 18, 2021].

[8] Duinotech. Infrared Receiver Module, 2021. https://www.jaycar.com.au/medias/sys_master/images/images/9594834845726/XC4427-dataSheetMain.pdf [Accessed: Nov 18, 2021].

[9] Vishay Semiconductors. IR Receiver Modules for Remote Control Systems, 2016. https://www.farnell.com/datasheets/2243458.pdf [Accessed: Nov 18, 2021].

[10] Duinotech. Red Laser Module, 2021. https://www.jaycar.com.au/medias/sys_master/images/images/9594851196958/XC4490-dataSheetMain.pdf [Accessed: Nov 18, 2021].

[11] Electron Test Equipment. What Is a Laser Diode, 2019. https://www.electrontest.com/what-is-a-laser-diode/ [Accessed: Nov 18, 2021].

[12] Duinotech. Infrared Transmitter Module, 2021. https://www.jaycar.com.au/medias/sys_master/images/images/9594833895454/XC4426-dataSheetMain.pdf [Accessed: Nov 18, 2021].

[13] Jaycar. Arduino Compatible Active Buzzer Module, 2021. https://www.jaycar.com.au/arduino-compatible-active-buzzer-module/p/XC4424?pos=2&queryId=fc72bc0b3e4fb834c23d01dc893156e3&sort=relevance [Accessed: Nov 18, 2021].

[14] Jaycar. 57mm All Purpose Replacement Speaker, 2021. https://www.jaycar.com.au/57mm-all-purpose-replacement-speaker/p/AS3000?pos=8&queryId=aac3826433e0e439c387075db32e0748&sort=relevance [Accessed: Nov 18, 2021].

[15] Wikipedia. Pulse-code modulation, 2021. https://en.wikipedia.org/wiki/Pulse-code_modulation [Accessed: Nov 18, 2021].

[16] Michael Smith. 8-bit, 8000 Hz audio playback on a PC speaker, 2018. https://playground.arduino.cc/Code/PCMAudio/ [Accessed: Nov 18, 2021].

[17] E. Born and M. Wolf. *Principles of Optics*. Cambridge University Press, Cambridge, UK, 7th edition, 1999.

[18] CVI Melles Griot. Gaussian Beam Optics, 2021. http://experimentationlab.berkeley.edu/sites/default/files/MOT/Gaussian-Beam-Optics.pdf [Accessed: Nov 18, 2021].

[19] Patrick Murphy. How to reduce incidents, 2021. https://www.laserpointersafety.com/how2reduceincidents/how2reduceincidents.html [Accessed: Nov 18, 2021].

[20] Australian Government. Laser hazards and safety, 2021. https://www.arpansa.gov.au/understanding-radiation/radiation-sources/more-radiation-sources/lasers [Accessed: Nov 18, 2021].