

Final Report
Engineering Project III (EECE72405)

Project 2: Coded Messaging System

Group # 4 - JAM

December 6th 2018

Justin Turcotte, 7860406

Daniel Dreise, 7996630

Ramtin Alikhani, 8185704

Table of Contents

Introduction	2
Background	2
System Description	3
Software	3
Hardware.....	8
Design and Test Methods.....	13
Software	13
Hardware.....	13
Project Management	16
Conclusion and Recommendations.....	17
Appendix	17

Introduction

The purpose of this project was to develop the Coded Messaging System which involves the development of a PC-based texting and paging intercom system for offices. The applications of this device includes prison inmate cells, the hallways outside of a room of college professor' cubicles or a digital answering machine outside of each room of an institution to allow outsiders to communicate with the people inside of the building, cell or room.

The chosen solution consisted of a text-based user interface to maintain simplicity. The interface would allow the user to input their own "address", such as their name, and thus any messages sent from that device while the program was running would be "stamped" with the user's address. It would also restrict the device from only receiving messages that were specifically addressed to their address. A novel aspect of the program is that it allows users to send and receive multiple text messages. Therefore, a user on one end could send a string of multiple texts, and the user on the other end would be able to receive them all from the order they were sent. Another unique aspect of the project is the implementation of a global error function. Every function used in the program is initially called from the error function, which runs diagnostic tests to ensure that the function is working properly, memory is allocated properly, user input errors are detected, etc. This would enable the easy creation of a general error code database where users could easily search why an error may have arisen, and also adds a dynamic aspect to the function because it allows the developer to implement new error codes should the need arise.

Background

From the onset of the project, a Google Drive was created to organize the group efforts and sync them to the cloud in a way that could be accessible to all group members involved. As a part of the Project Schedule and Plan task for the 1st Task and Evaluation (T&E or TE, for short), each week was given a folder from Weeks 1 through 7 of the project. From then onwards, all content related to or involving the deliverable for the weeks were uploaded onto their respective weeks. More information is shared in the project management section of the project on the way the group decided to approach the problem.

The team assessed the strengths and weaknesses of each person in the group. Then, from the gathered information, divided weekly roles into areas where each could excel, but also had members working in areas where they needed to grow so they could improve their skill.

Regarding software, a highly modular design was chosen. This was because the team wanted freedom to reuse functions for multiple implementations. With the addition of modular functions, the project became much more streamlined and functions could be used multiple times in different scenarios. The team designed an error function, by which all other functions would be passed. Then from there, designed a new solution (complete with source file and header) for each major section of the project (example: ECHO_COMPRESS.H, ECHO_COMPRESS.CPP, ECHO_RECEIVE_MESSAGE.H, and ECHO_RECEIVE_MESSAGE.CPP).

Regarding hardware, the designs for the modulator, RS232 to TTL IC, and demodulator were provided. However, the voltage divider, by which the voltage delivered by the RS232 would be sent to

the modulator and determine the output frequency, needed to be designed. The team chose the simplest and most streamlined circuit, consisting of two potentiometers and a signal diode. The design was chosen because of its simplicity, low cost, and reliability.

System Description

Software

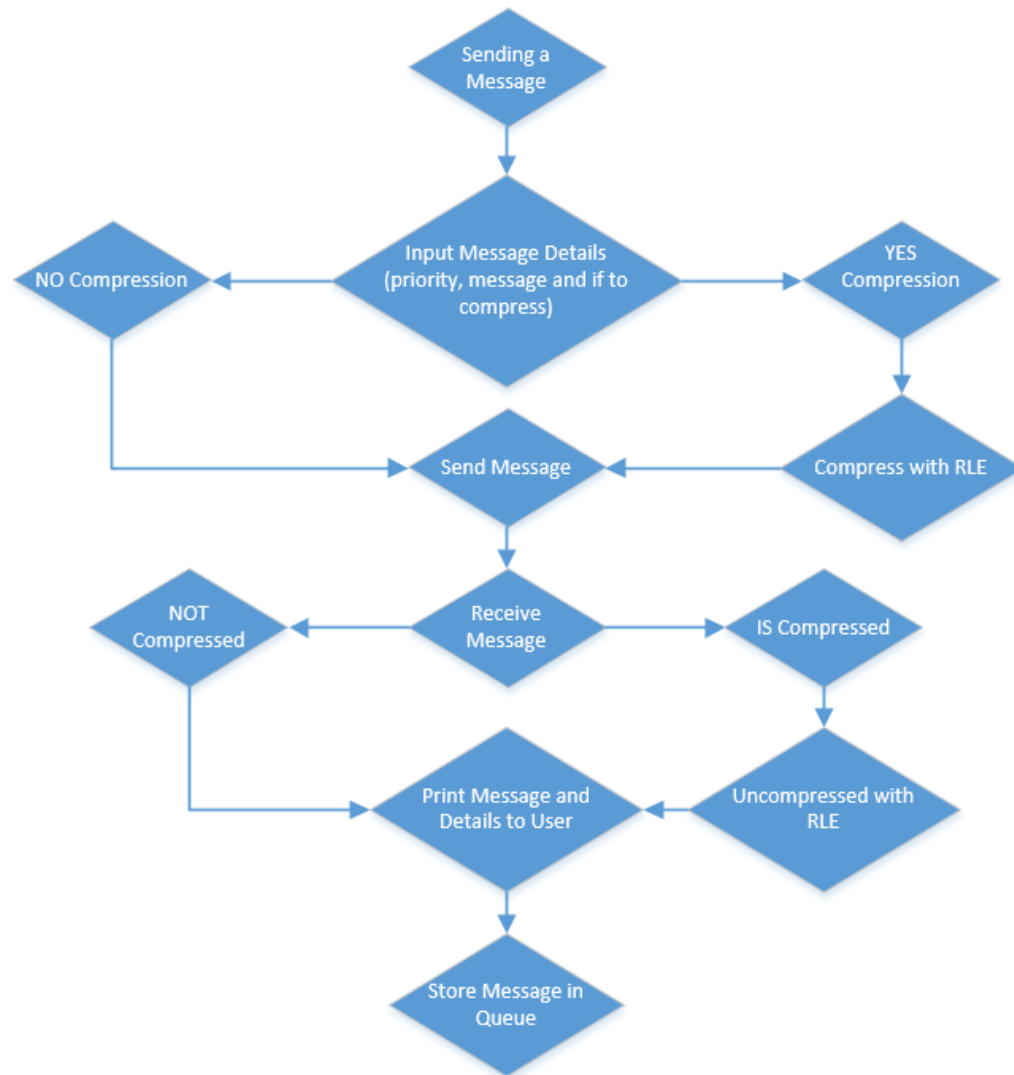


Figure 1 - Block Diagram of Message Transmission and Reception

Our program is extremely modular, with approximately 15 .cpp and .h combinations plus the source code. Our aim was to use GitHub and this required a modular design to keep conflict issues to a minimum.

```
//wrapper for sending multiple texts (Echo_Frame.cpp)
int frameSendMultiple(void)
{
    //send multiple texts or quotes
    int i;
    int quotes = 0, texts = 0; //hold how many messages and/or random quotes

    printf("How many texts would you like to send?\n");
    scanf_s("%d", &texts);
    getchar();

    printf("\nHow many quotes would you like to send?\n");
    scanf_s("%d", &quotes);
    getchar();

    for (i = 0; i < texts; i++)
    {
        error(frameSendText);
    }

    for (i = 0; i < quotes; i++)
    {
        error(frameSendQuote);
    }

    return SUCCESS;
}
```

Figure 2 - Sending Multiple Messages

```
int frameReceiveMultipleText(void)
{
    int i;

    int count = 0;

    printf("\nPlease enter how many messages you are expecting\n");
    scanf_s("%d", &count);
    getchar();

    for (i = 0; i < count; i++)
    {
        error(frameReceiveText);
    }

    printf("\nFinished Receiving!\n");

    system("PAUSE");
    system("CLS");
    return SUCCESS;
}
```

Figure 3 - Receiving Multiple Messages

We also implemented a function to send and receive multiple quotes or messages and add them to a queue on the receiving end (as seen in the two figures above). This is a fairly unique feature that was able to effectively receive multiple messages using a long timeout for the RS232 communications.

```

int sendAudio(void)
{
    //MUSTS:
    //→ get audio the user wants to send
    //→ get senderID they want to send it to
    //→ compressed with RLE/Huffman
    //→ transmit the message
    //COULDS:
    //→ load saved audio and send it

    long audioSize;
    int status = 0; //status of audio, 0 = no issue, 1 = issue

    //set up recording settings
    audioSetup();

    //send audio size
    audioSize = getAudioSize();
    //printf("audio size before compression: %d\n", audioSize); //TEST
    outputToPort(&audioSize, sizeof(audioSize));

    //receive 'ok' to send audio
    inputFromPort(&status, sizeof(status));
    switch(status)
    {
        case 0:
            //record audio
            audioRecord();

            //compress audio
            //audioSize = compressAudio((unsigned char*)getAudio(), getAudioSize());

            //transmit audio
            //outputToPort(getCompressedAudio(), audioSize);
            outputToPort(getAudio(), getAudioSize());
            printf("\nRecording is doneeee0\n");
            break;
        case 1:
            printf("\nHandshake failed! Unable to send audio of previous size! Going to previous menu...\n");
            break;
    }

    system("PAUSE"); //temp
    return SUCCESS;
}

```

Figure 4 - Sending Audio

```

//wrapper to receive audio
int receiveAudio(void)
{
    //MUSTS:
    //→ receive audio and audio size
    //→ play audio to user
    //COULD:
    //→ store audio for later

    long audioSize; //size of audio
    short* audio; //audio data
    int status = 0; //status for receiving audio, 0 is no issue, 1 is issue

    audioSetup(); //temp

    //receive size of audio
    inputFromPort(&audioSize, sizeof(audioSize));

    //send 'ok' if space is allocated for audio
    audio = (short*)malloc(audioSize);
    if (audio == NULL)
    {
        printf("\nERROR: Unable to allocate space required for audio buffer!\n");
        status = 1; //update status and send error
        outputToPort(&status, sizeof(status));
        return -1; //temp
    }

    //save audio to buffer
    setAudio(audio);
    setAudioSize(audioSize);

    //output 'ok' for receiving

    outputToPort(&status, sizeof(status));
    //receive audio
    inputFromPort(audio, audioSize);

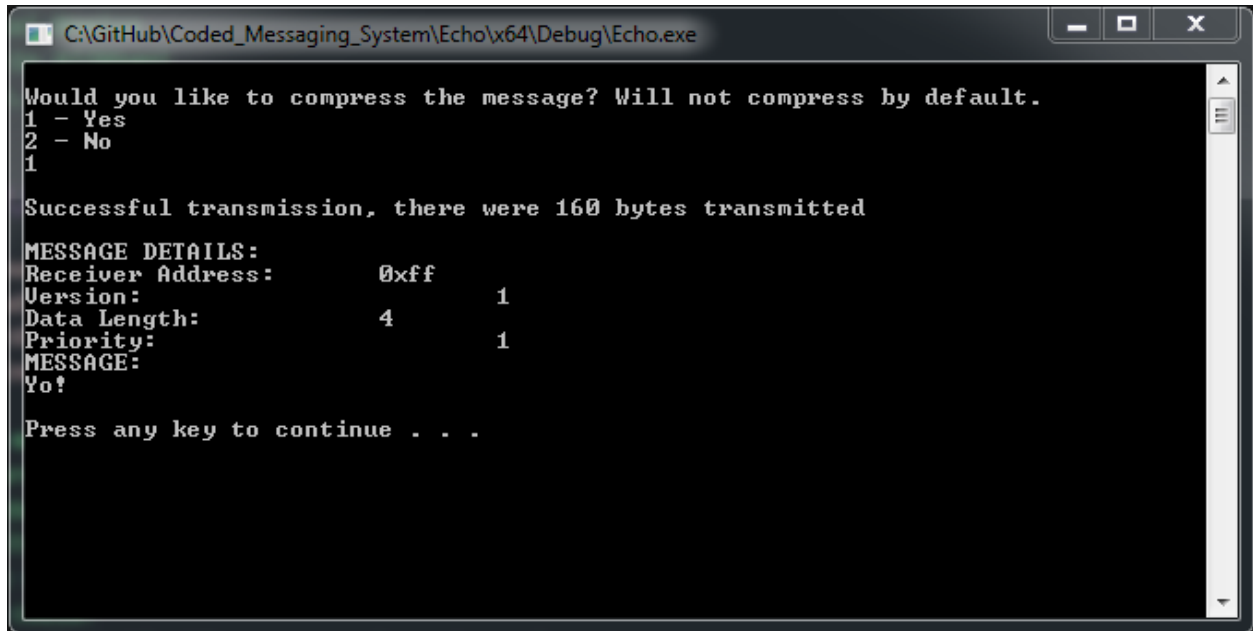
    //play audio
    audioPlayback();

    system("PAUSE");
    return SUCCESS;
}

```

Figure 5 - Receiving Audio

Our audio function also sends a header ahead of time with the information of the audio (its size) so that dynamic audio lengths can be sent.



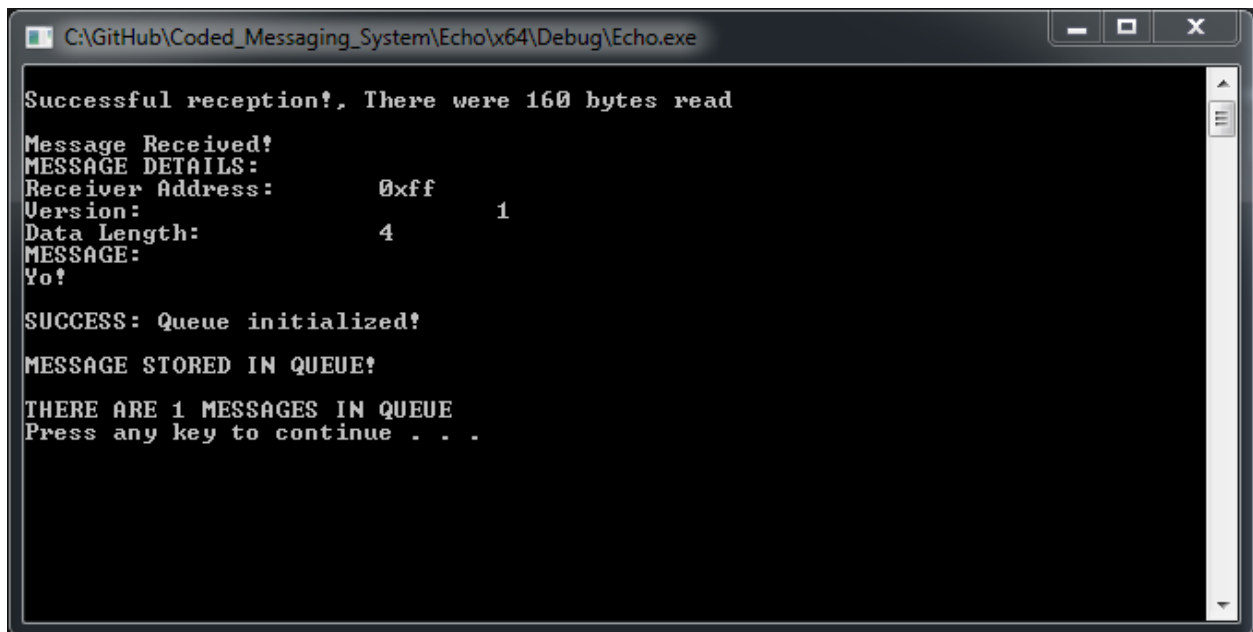
```
C:\GitHub\Coded_Messaging_System\Echo\x64\Debug\Echo.exe

Would you like to compress the message? Will not compress by default.
1 - Yes
2 - No
1

Successful transmission, there were 160 bytes transmitted

MESSAGE DETAILS:
Receiver Address:      0xff
Version:               1
Data Length:          4
Priority:              1
MESSAGE:
Yo!

Press any key to continue . . .
```

Figure 6 - Sending a Message with Compression

```
C:\GitHub\Coded_Messaging_System\Echo\x64\Debug\Echo.exe

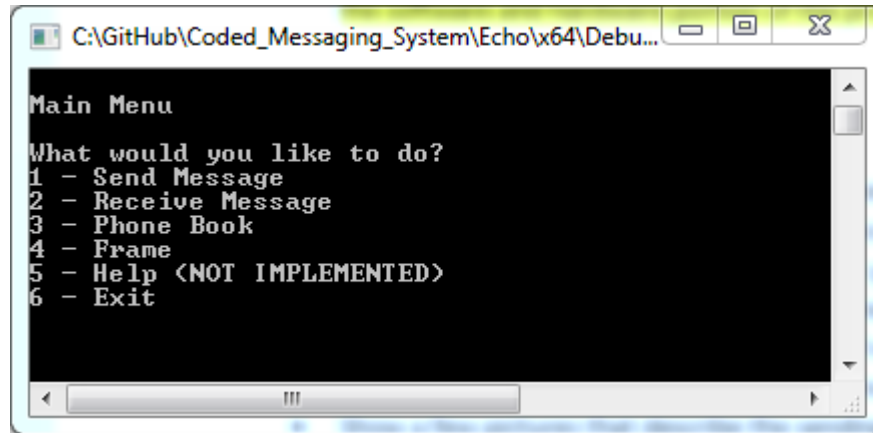
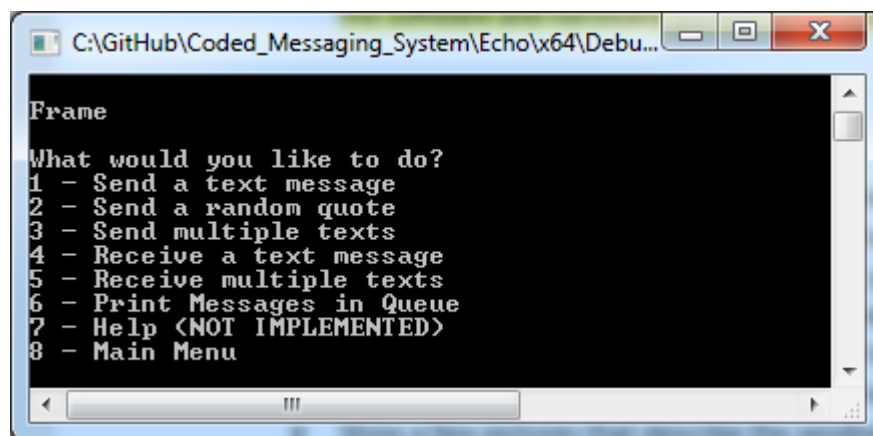
Successful reception!, There were 160 bytes read

Message Received!
MESSAGE DETAILS:
Receiver Address:      0xff
Version:               1
Data Length:          4
MESSAGE:
Yo!

SUCCESS: Queue initialized!
MESSAGE STORED IN QUEUE!
THERE ARE 1 MESSAGES IN QUEUE
Press any key to continue . . .
```

Figure 7 - Receiving a Compressed Message

The sending, receiving and queuing of messages in our program, works like so: the message is sent to the receiver, in the header of the message is the sender/receiver id, size of the message, settings for compression and encryption. The message is printed on both sides so the sender receiver can both see the message and the receiver also gets the message added to the queue which can be traversed as needed.

*Figure 8 - Main Menu**Figure 9 - Frame Menu*

The menu design used is more like a tree approach. We wanted to avoid swamping the user with commands so we built a multi-layer menu system that would cleanly display the commands the user wants to use. It was also designed so that on startup, the user id and com port to be used would be inputted right away for the functioning of the program. The menu shown in Figure 4 is the frame which sends a header with the message (described below figure 2).

During the project, we used GitHub to manage our code. This worked exceptionally well for us at first. Allowing us to share code and edit “simultaneously”. Near the end of the project, we started getting merge conflict errors. These should be simple to solve; approve the code that’s right or accept both. However, it was not simple and GitHub’s conflict resolution tool would not work correctly.

Hardware

The hardware of the project included assembling a modem with an FSK modulator and demodulator, DB9 cables (to connect to computer’s communication ports), MAX232 IC, and low quality wiring on a breadboard. The design was so that it could be probed with an oscilloscope to measure the output waveform being transmitted by the intercom and received by the receiver.

The Setup of the modem was emulated by the design of the 4th and 5th lab of the Telecommunications course. Frequency shift keying is a type of frequency modulation in which digital

information is transmitted through changes of a carrier signal. An FSK modulator modulates the transmitting signal and the demodulator demodulates it for the receiver's station to enable the receiver to understand the message that was sent. The FSK modulator used was the 74LS628 IC and the demodulator used was the XR2211 IC. Their corresponding datasheets can be found in the Appendix section. The FSK modulator was set up on a breadboard like so below:

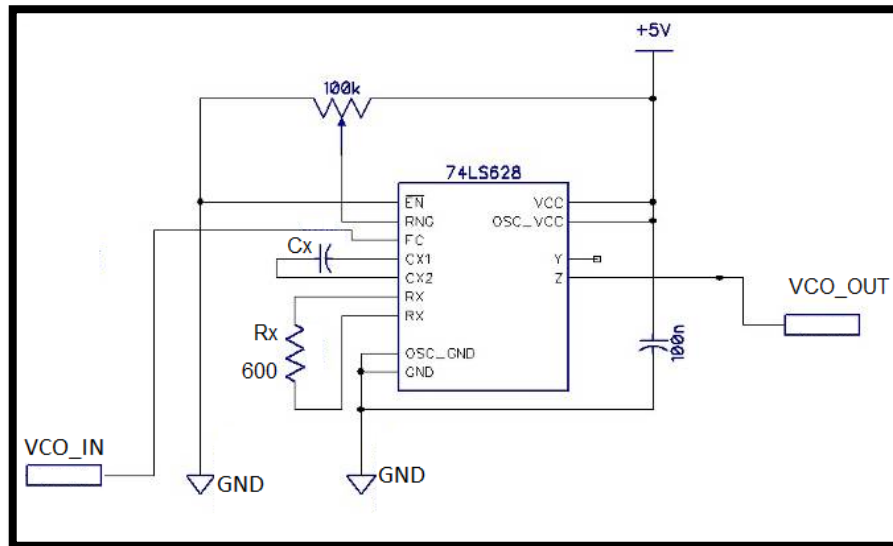


Figure 10 FSK Modulator Setup

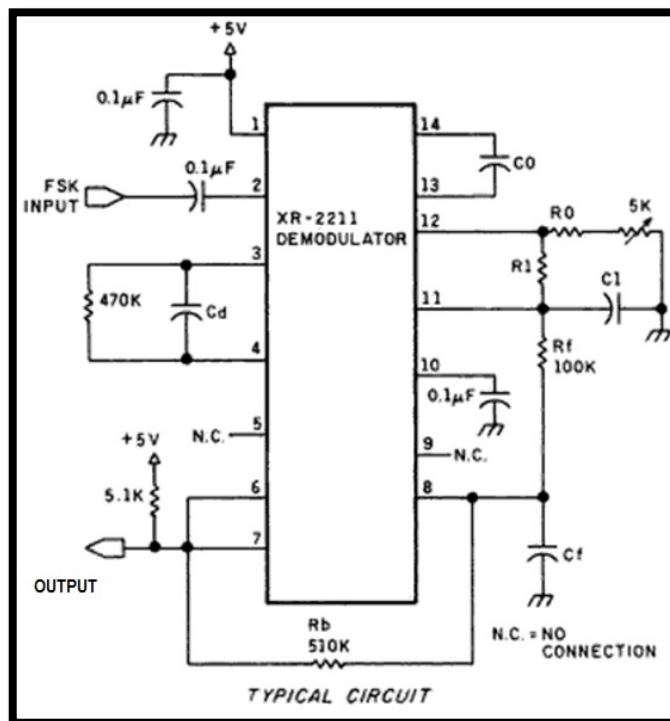


Figure 11 FSK Demodulator Setup

The FSK system was adapted onto the ESE year 2 laboratory computers with an RS-232 communications serial cable attached to the DB9 serial ports. The RS-232 (Recommended Standard - 232) is a standard for the serial communication of data between a computer and the modem. Via COM ports that were selected by the transmitting and receiving stations, the DB-9 cables were attached to a 9-pin female D-subminiature connectors provided in the 2nd year kits. The RS-232 uses Baud rates to communicate information between two ports. A baud rate is a unit of measurement of sending signals from one port to another using a certain frequency range, usually in bits per second. A typical baud rate of 9600 bps is usually used to communicate with the RS-232 standard. Two were used; one for the transmitting side and one for the transmitting side.

The system involved in the Hardware is outlined below. The components involved are for inputs and outputs. The inputs are in yellow and the outputs are in red.

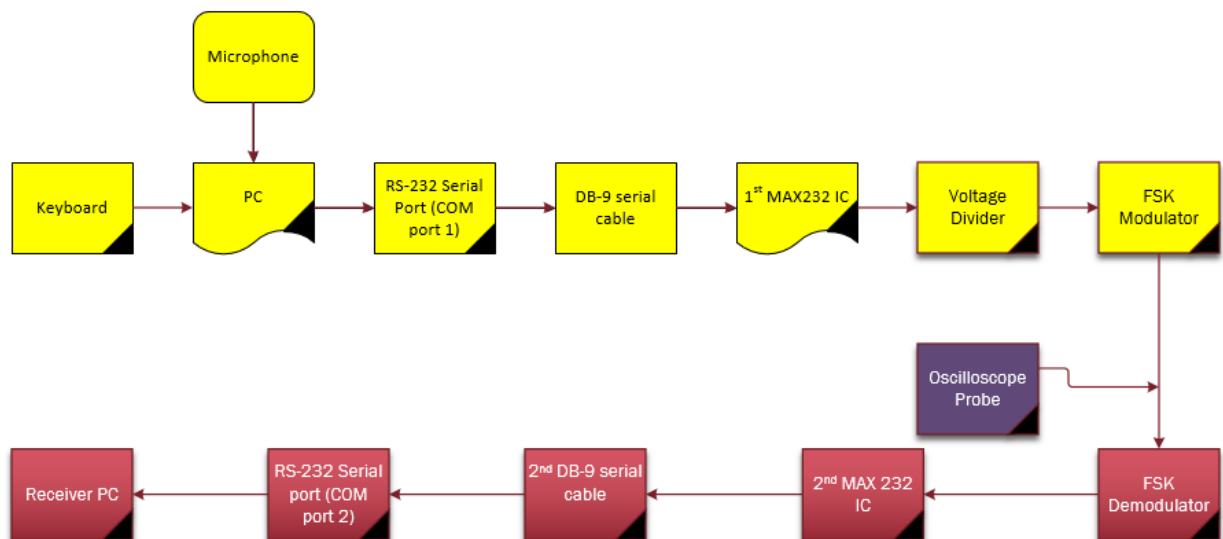


Figure 12 Flowchart of the hardware system

Hence the interface of the intercom involves sending text messages or audio, the inputs of the system start with a microphone and a keyboard into the transmitting PC. The PC in use must have a sound card capable of recording from the microphone jack, otherwise this system will not work. Once the text message or the audio has been recorded, the data is sent to the DB-9 Serial cable via the com port (RS-232 serial port) selected by the transmitting user on their interface. The outgoing signal enters the MAX232 chip (used HIN232 IC) which takes the transmitting and receiving signal and converts them into analog signals, with an input square wave of -12 Volts and +12 Volts and output square wave of 0 and 5 Volts.

The pinouts of the DB-9 cable were mapped and soldered by low quality wiring onto the breadboard to be integrated into the hardware system on the breadboard. The pinouts are indicated by the figures below:

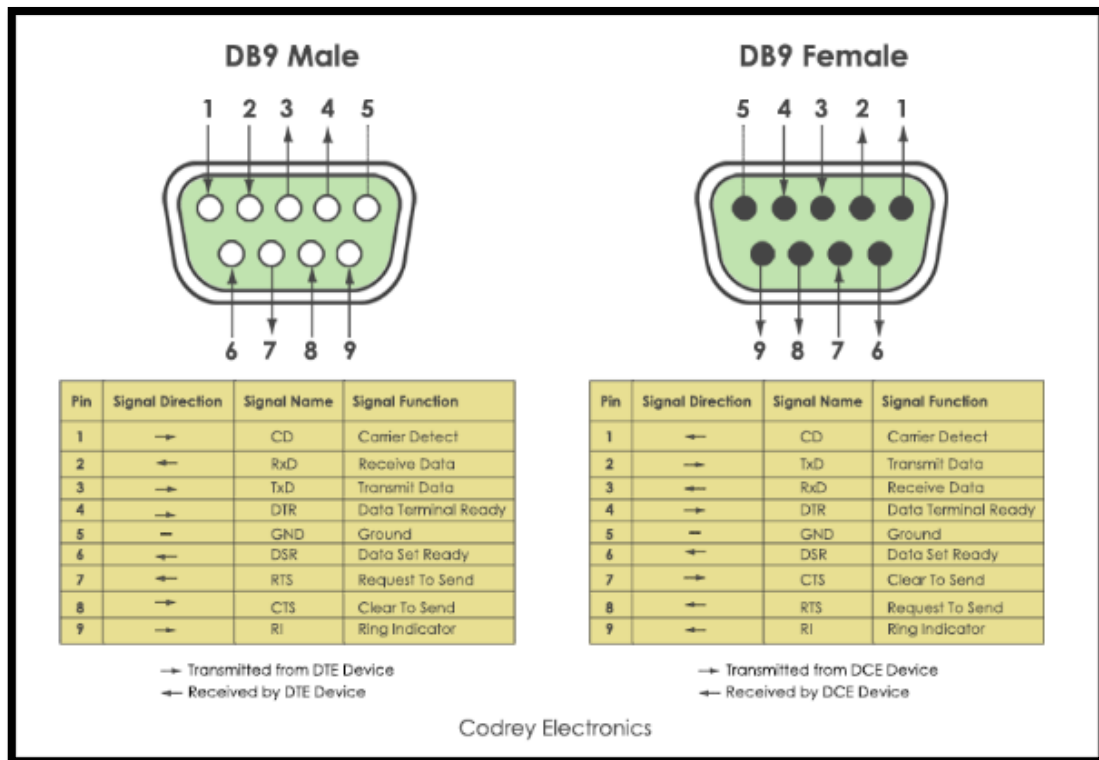


Figure 4 Pinouts of the DB-9 serial cable

Functional Description

Apart from the electrical characteristics, RS232 defined the functions of signals that are used in the serial interface. Some of them are common ground, Data, control and timing signals. Here is a list of signals used in RS232 pinout.

Signal Name	Function
Protective Ground	This signal is connected to chassis ground of metallic connector.
Common Ground	Zero reference voltage level for all the control signals.
TxD (Transmit Pin)	To transmit data from DTE to DCE.
RxD (Receive Pin)	Sends data from DCE to DTE.
DTR (Data Terminal Ready)	DTE is ready to accept request.
DCD (Data carrier Detect)	DCE accepts a carrier from a DTE located at remote location.
DSR (Data Set Ready)	DCE is prepared to send and receive the information.
RI (Ring Indicator)	Detects the incoming ring tone on the telephone line.
RTS (Request to Send)	DTE call for DCE to send the data.
RTR (Ready to Receive)	DTE is geared up to receive data coming from DCE.
CTS (Clear To Send)	DCE is in a ready state to accept data coming from DTE.

Other than above signals, (primary signals) RS232 provides secondary signals such as secondary DTE, secondary RTS, secondary DCD, secondary TxD and secondary RxD for optional connection of DTE and DCE.

Figure 5 Pinout descriptions of the DB-9 pinout mapping in figure 7

Once the signal leaves the voltage divider, it enters the FSK modulator and then can be probed to check the signal being transmitted onto the receiver side of the system. The signal now enters the output phase. It is inputted into the FSK demodulator and then onto a different MAX232 chip and then back to a secondary serial com port and DB-9 cable back to the receiving PC.

Design and Test Methods

Software

In this project, we aimed to design a tree menu system. Each menu has sub-menus that eventually run the intended program. This was to eliminate clutter and improve user experience.

Code was modularized in a way that allowed each individual piece to be tested separately. Message sending, receiving compressing, etc. were all wrapped and used as black boxes. This allowed each piece to be tested separately as necessary and changed without issue.

Frequently, we would test our code with the goal of breaking it. We would test inputs that it should protect against as well as testing unexpected variables. For example, setting the bit rate to an odd rate. This caused a crash because only 4 and 8 bit rate are allowed. So to fix it we made the user pick between the two.

Many, but not all bugs were captured during testing. We would frequently test small chunks of code to make sure what we are working on is working up to this point. Once individual pieces were completed and deemed safe, we would merge them together and iron out any additional bugs that appeared.

Hardware

Testing the hardware of the system was done discretely. Through the telecommunications labs 4 and 5, the modulator and the demodulator was tested individually. The rest of the assembled components of the hardware were installed and tested individually as well, including the DB-9 wiring to the COM ports, the MAX232 chip and the voltage divider subsystem.

The FSK modulator only works with a square wave of 2.4 – 2.6 Volts, it cannot work with the outputted square signal of 0 and 5 Volts of the HIN232 IC, therefore, the team was tasked with designing their own subsystem to lower the input voltage to the range of 2.4 – 2.6 Volts. This was the first time that the group had to come up with the design of curating their own solution. Initially, an idea was offered to take reference of Joshua's work from a different group; a voltage divider setup by 2 potentiometers a diode and a transistor. Then, Ramtin made a schematic of a voltage divider that only used 2 diodes and only resorted to using a transistor as a method of last resort. After a bit of simulation testing, Ramtin came up with a rudimentary design that involved 2 diodes and a 2 potentiometers. Justin was offered to work with Ramtin's design and modified it to include only a single diode and then Daniel took the design and modified and refined it further.

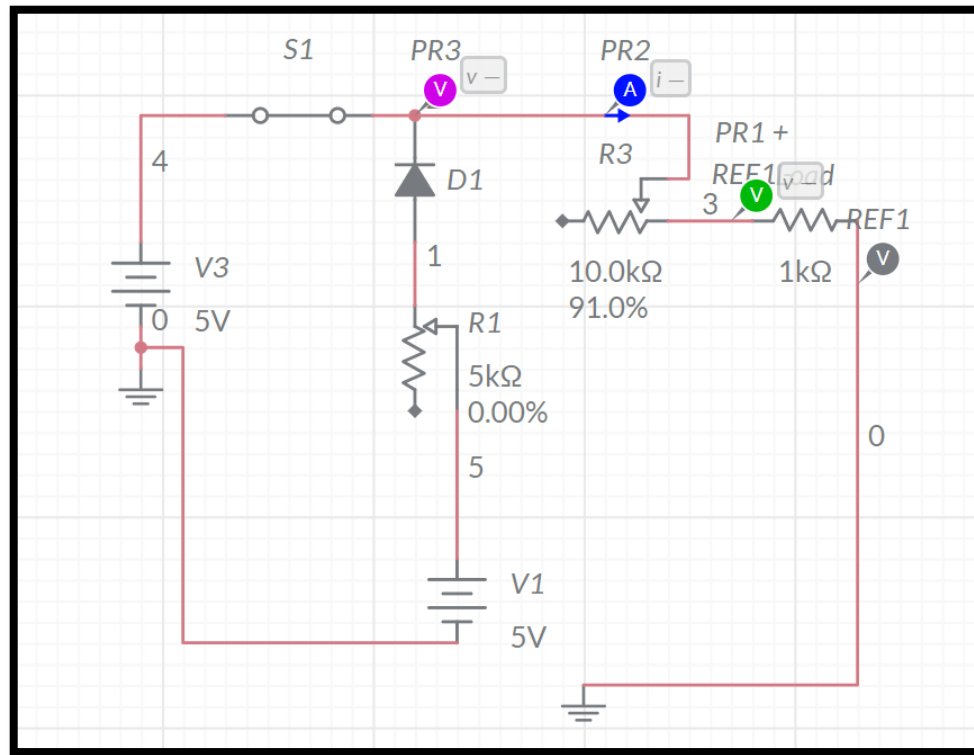


Figure 13 Initial design by Ramtin, refined by Justin

Ultimately, the group setup a voltage divider with 1 diode and 2 potentiometers to lower the voltage from 0 - 5 Volts down to 2.4 - 2.6 Volts. This was the group's second most difficult part of the project, but with all of the group members working towards a solution, the challenge was quickly overcome within a day. The schematic of the working voltage divider is shown below:

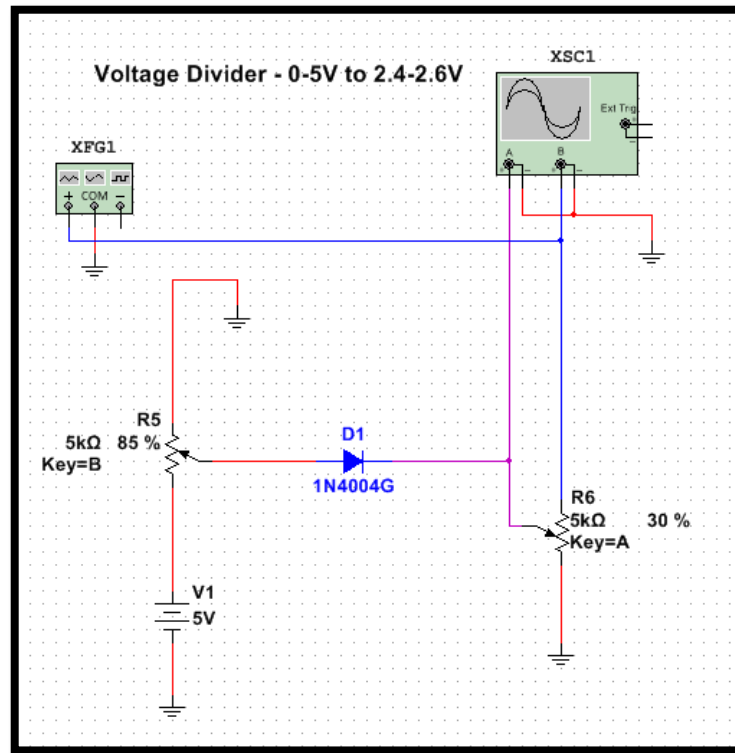


Figure 7 The final design of the Voltage divider by Daniel, inspired by the previous design

Next, the parts were put together and tested. The method of testing was performed by sending a message or an audio and by using the oscilloscope, probing the input and the output of the MAX 232 and the input and output of the voltage divider through the 4 channels of the oscilloscope simultaneously. All tests resulted in a success; the transmitter from the DB-9 cable was outputting a square signal of -12 and +12 volts, the output of the MAX232 was a square signal of 0 and 5 volts and the output of the voltage divider was a square signal of 2.4 to 2.6 volts.

Then we probed the input and the output of the FSK modulator and the input and the output of the FSK demodulator. We encountered a major problem when testing this feature of the hardware. It was the most difficult problem ever encountered for the project. From the output of the demodulator, the signal was often initially a square wave, as was to be expected. However, the output would oftentimes be very unstable and the wave would widen and narrow up in terms of frequency and eventually merge into a high DC output wave, then go back to an unstable square wave. Daniel and Ramtin troubleshooted the problem. At first, pinpointing the issue was found to be very difficult. Daniel suspected that the voltage divider might be too rudimentary for such a complex design and advised to create a new and more sophisticated voltage divider with more diodes and a transistor. However, hence we tested the voltage divider individually and in the system it did not possess any problems, we soon abandoned the idea to create a new voltage divider and Dan put away some valuable time to work on troubleshooting the issue. Hence all the other components were working correctly, the issue was narrowed down to the voltage divider and the potentiometers were adjusted slightly to check the output waveform going into the FSK modulator and the modulated signal was observed. The voltages of the output were slightly changed and the modulated signal became stable, but the frequency of the modulated signal had been affected. Therefore, to solve this secondary problem, the baud rate of the

RS-232 was changed from the transmitting and receiving PC's down to 1200 bps. The solution had worked; the modulated signal was a stable square wave that was transmitted onto the demodulator. From the receiving side, the input and the output of the demodulator and the input and the output of the 2nd MAX 232 chip was observed and no issues were detected.

Ultimately, the entire hardware system was integrated with the software system and tested and a text message was transmitted and received successfully.

Project Management

The Master Plan and scheduler that was initiated after the inception of the project was shared with all group members on Google Drive. It helped manage the group in their weekly deliverables and progress. It is linked in the appendix section and can be accessed via the web.

A step was taken to help the team understand what each group member's objective strengths and weaknesses were in each skill; hardware, software and soft. This extra step helped the group on deciding how to divide the tasks for each deliverable. It was called the Project Schedule Strategy.

<u>PROJECT SCHEDULE STRATEGY</u>		
Following is a list of strengths and weaknesses of each member in the group:		
	Strengths	Areas to improve
Ramtin	<ul style="list-style-type: none"> - Memorization - Consistency - Hardware - Public speaking 	<ul style="list-style-type: none"> - Programming in C - Math related concepts - Slow learner
Justin	<ul style="list-style-type: none"> - Programming in C - Time management 	<ul style="list-style-type: none"> - Hardware concepts
Daniel	<ul style="list-style-type: none"> - Management / Delegation - Hardware - Decision making 	<ul style="list-style-type: none"> - Programming in C

Therefore, we have delegated tasks in the project schedule accordingly based on the above strengths and areas to improve. Daniel has been given the heavier hardware and management roles, but has been given small programming tasks in order to increase skill. Justin has been assigned to the more complicated programming roles, but has also been given small hardware roles to improve skill. Ramtin has good memory, hardware, and is good at public speaking, therefore he will be given more roles in communication with professors and for demonstrations, while wanting to get more comfortable with programming in C, will be given small tasks catered to software improvement.

Figure 14 This scheduler enabled the team to divide tasks up according to their strengths and weaknesses, weekly

Once established, Software was mostly divided up between Daniel and Justin, while the hardware was divided up between Ramtin and Dan. Ramtin was tasked with the paperwork of the project. The Project Planning and Scheduling was done on a Google Sheets application on the Google Drive folder where all of the TE from weeks 1 to 7 were outlined, tasked, set the difficulty to and progress level of, as well as checkbox to indicate that it had been completed. Ramtin and Daniel curated the Master Schedule and Plan, and took turns between the weeks to supervise the completion of tasks for each deliverable on the master schedule.

Conclusion and Recommendations

In conclusion, besides the software issues that plagued the final demonstration, the hardware performed flawlessly as it was designed to. The software performed as it was designed to before the compression was attempted to be implemented by the group. The working components before the compression were even demonstrated to the instructors before the final presentation as a proof of concept. During the trials before the final demonstrations, the integrated Software and Hardware showed no problems and worked as it was designed.

Since the team consisted of members with different skill levels in different areas, there was something to be learned by everybody. The team learned how to practically integrate software with hardware through the FSK system. They also learned how valuable it is to create a modular program to allow extra freedom when needing to use code multiple times in different scenarios. The team had troubles dealing with multiple file programs, especially because it was a fairly new environment for everybody so there were a few issues along the way.

For the future, it is recommended that some of the software requirements for the project be removed so that more time can be spent crafting a well-functioning program. The number of requirements deterred the team from honing our understanding and craftsmanship of the software, which left some members of the team feeling like they did not fully grasp many of the core concepts or unable to practically write good code. Regardless, the project was exciting, daunting, and rewarding. Thank you for the opportunity.

Appendix

- [Code Base](#)
- [All Datasheets used in this project are listed here](#)
- [Master Scheduler and Plan used by the group](#)
- [Diagnostics for Testing the software](#)