Assignment 2 Report

EECE 443

Team 2: MCU TNC Design

Kaleb Leon – C00094357
Kobe Keopraseuth – C00092349
David Cain – C00043561

March 19th, 2020

MCU TNC Design

Kaleb P Leon, *Member, IEEE*, Kobe Keopraseuth, *Member, IEEE* and David Cain, *Member, IEEE*

***Abstract*—Developed by Kaleb Leon, Kobe Keopraseuth, and David Cain with All Rights Reserved. The objective of this paper is to document and describe the design process of developing a software based Terminal Node Controller (TNC). The TNC will be able to perform all the basic functions of a hardware TNC but with little to no hardware necessary. It will be capable of receiving an audio tone FM modulated signal, convert it into binary, gather the payload, check for bit errors and send the payload to the PC via KISS (keep it simple, stupid) mode. It will also be able to receive a packet via KISS from the PC, follow the AX.25 protocol to form it into a valid data packet for radio communication, translate it into an FM modulated audio signal tone and send the tone to the radio. This paper has multiple parts that show the design process: the scope of work, objective tree, feasibility analysis, functional specifications and design alternatives.**

***Index Terms*— controller, payload, packet, modulation, radio communication**

I. Introduction

THE MCU TNC Design team is in the process of designing a software based TNC. This project's purpose is to make an easy to use and more compact version of a TNC. Hardware TNCs are usually very bulky electronics and come with a hefty price tag. With the use of a microcontroller, the design of a TNC can be condensed down to around four inch surface area at the price of a standard microcontroller. The team will implement code in C on a microcontroller that will represent all the analog functions of a hardware TNC in the form of software. The software's job will be to process all the incoming data into packet form suitable for the PC when receiving and the radio when transmitting. When in transmitting mode, the microcontroller will receive a KISS packet from the PC and translate the payload. It will then take the payload and transfer it into the AX.25 format using bit stuffing techniques which will then be modulated onto a signal in the form of frequency modulation. The frequency modulation will be done with different frequencies representing a zero or a one. When in receiving mode, the microcontroller will receive a FM modulated audio tone and translate it down into a KISS packet to be sent to the PC. These processes done normally in the form of hardware can be done all in code. The future sections of this paper include research done that has relevance to the protocols and systems that will be used in this design and a detailed project analysis which will contain a feasibility analysis in addition to design alternatives and tradeoffs. Following the main sections of this paper, are a set of appendices A, B, and C. Appendix A contains a more streamline scope of work as well as functional requirements. It will also contain a level 0 diagram and an objective tree to visually describe our designs basic functionality. Appendix B will contain a feasibility analysis in which we discuss whether the project is technologically, timely, and economically feasible in addition to our plan for the incoming year. Appendix C will show our analysis on our alternatives and tradeoffs to the systems and protocols used in this design.

II. Research of work done by others

In the world of radio communication, there are many projects involving TNC design but most of them involve hardware redesign or optimization. Our design takes the analog systems of the previous age and takes them into the digital modern age of software. However, there has been one other project that has accomplished this task of a software TNC and they call their software TNC, Direwolf [4]. We hope that our research and design will help improve on their design and make it more compact and efficient.

A. KISS Mode

One of the first obstacles to overcome is how to send data from the PC to the TNC. According to our project mentors [Mr. Nolan Edwards, Mr. James Palmer, Mr. Nick Pugh, and Mr. Rizwan Merchant], the most common protocol used to communicate is the KISS protocol. According to Chepponis and Karn presentation [2], the KISS protocol provides direct computer to TNC communication. It provides the host software of the PC with the ability to control all the TNC functions. It allows the PC to send a packet with a payload controlled by stop and start flags. These flags are represented by a hex value of C0. The data in the payload can be as large as 1024 bytes but this size can also increase based on the TNC's specs. Our project will use this protocol to send and receive data from the PC. However, we will be converting our data into the KISS form using software on the microcontroller.

This paper was submitted for review February 18, 2020. The project's mentors are Mr. Nolan Edwards, Mr. James Palmer, Mr. Nick Pugh, and Mr. Rizwan Merchant.

K. P. Leon is with Electrical and Computer Engineering Department, University of Louisiana at Lafayette, Lafayette, LA 70504 USA (email: C00094357@louisiana.edu)

K. T. Keopraseuth is with Electrical and Computer Engineering Department, University of Louisiana at Lafayette, Lafayette, LA 70504 USA (email: C00092349@louisiana.edu)

D. L. Cain is with Electrical and Computer Engineering Department, University of Louisiana at Lafayette, Lafayette, LA 70504 USA (email:C00043561@louisiana.edu)

B. HDLC (High-level Data Link Control) Protocol

After the KISS packet is received by the TNC it will have to translate it into another packet format to prepare it for transmission. This was the next obstacle. This protocol, HDLC, is a data link layer described by Estevez [3] in combination with AX.25 to solve this packet formation that works mainly with KISS. The HDLC protocol is made from the payload sent from KISS, start and stop flags, a control frame, and an error checking frame. Estevez states that HDLC is NRZ-I encoded meaning that a logical bit 0 is marked by a change in state and a logical bit 1 is marked by no change in state. Similar to KISS, it also has an end flag or maker which is represented by a Hex 7E. To identify the difference between these end flags and the rest of the message, HDLC forbids more than 5 consecutive 1s. To do this they use a technique called bit stuffing which means when they see 5 consecutive 1s a 0 bit is added in and this zero is ignored. The last main function that Estevez mentions is that it has a form of error checking. It has a frame for a 16 bit check sum which is compared at the transmitter and receiver. This checksum is computed using CRC-16CCITT. If this frame does not match at the receiver the packet is dropped and a retransmission is requested. A visual diagram of an HDLC packet is shown in Figure 1.

Flag	Address	Control	Information	FCS	Flag
8 bits	8 or more bits	8 or 16 bits	Variable length, 8×*n* bits	16 or 32 bits	8 bits

Figure 1. HDLC Packet Format

C. AX.25 Protocol

The AX.25 protocol since it is also a Data Link Layer protocol defines a specific format for how certain HDLC frames are to be set up and what data goes in these frames. According to the spec sheet of AX.25 by Beech, Nielson, and Taylor [1], the frames handled by the AX.25 protocol in the HDLC packets are the control and the address frames.. In reference to the address frame, it contains the source address, destination address and one or more repeater addresses up to eight. The control frame describes what the data is: information, numbered, unnumbered, or supervisory. This protocol provides a more specific packet formatting as an implementation of HDLC.

D. AFSK (Audio Frequency Shift Keying)

Now that these frames are formed it is important to know how the packets will be translated into audio signals and transmitted over the air. Bits are to be represented by two separate frequencies. Based on Estevez's documentation [2], FM AFSK normally uses a baud rate of 1200 on a frequency shifts between tones of 1200 Hz and 2200 Hz. To translate these signals when receiving, frequency counting could be used as well as a comparator or Schmitt trigger.

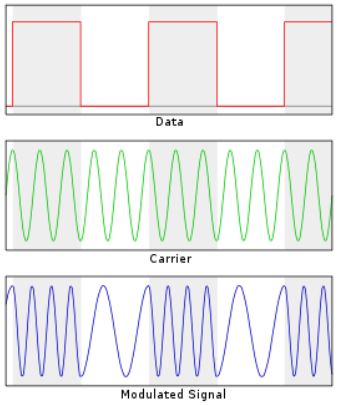


Figure 2. Audio Frequency Shift Keying

III. Project Analysis

A. Project Feasibility

The scope of work and required functional specifications are feasible.

The main goal of the project is aimed at replacing the hardware that was originally developed in the early 1980s, this means the technical requirements are very minimal. In addition, a previous design has been successfully achieved by a project called Direwolf [4] meaning the design is feasible on a technological standpoint. Also, the hardware and protocols we plan on using are well documented and are already implemented in hardware so it is very feasible to make the shift to software related logic.

Given the simplicity of packet management required to have a functioning TNC it seems it will be feasible to have close to a working design in the first half year. In addition, the second half year could be used to develop our own more optimized microcontroller board as well as add some features. According to the Gantt Chat referenced in Appendix B, our planned schedule projects the project's feasibility in the time span of one year. In the Gantt Chart we estimated the time that each part of the project would take and tried to fit it into one year. We analyzed the times and dates estimated along the critical path and found that even with some delay we will have a functioning project by one year.

In the terms of cost, the design required little to no physical devices or hardware so the price will be relatively low considering most of the work will be done using free software written by us. In addition, we currently possess many of the necessary components personally and from our mentors so the total projected cost is less than one hundred dollars making the project economically feasible.

B. Alternatives and Tradeoffs Considerations

Hardware

With a project based mostly around software there are small amounts of alternatives and tradeoffs to consider. In Appendix C, Tables are shown referencing our comparison of different alternatives to many parts of our design and our decisions. The

alternatives are mostly based around specs given to us by our mentors and we chose the ones that would best meet those specs. Our first decision to make was on microcontrollers. This would play a major role in our overall design and would have to meet many of our required specs. Our three choices were the STM32L4433, the Teensey 4.0 board, and an Arduino Mega. The next decision was how would we implement and build our PTT (push to talk). There are a number of ways this logic could be implemented like using an Inverter Circuit, a P-Channel MOSFET, or a Comparator IC. Also, we had to make a decision on what we would use for our signal conversion methods when receiving and transmitting. For receiving we will need to take in an analog signal and convert it to digital binary so we looked into different analog to digital converters such as: using Fourier analysis, Schmitt Trigger, or Zero Crossing. Lastly, we had to look into transmitting by translating a binary packet into an FM audio tone. This can be accomplished using the STM's built in digital to analog controller, a resistor switching network, or a Digital to analog integrated circuit. Most of the decisions were made based on whether they were already on the micro controller and power consumption. Ultimately, for microcontroller choice we chose STM32L4433 to use due to its already integrated DACs and ADCs as well as its CRC calculation unit which can detect bit errors. However, the downside is we are not familiar with C and will have to learn. The STM32 fills all the alternatives so it makes it the obvious choice.

Software

Being that software is a large portion of this project, many alternatives and tradeoffs needed to be considered. The environment the software will be programmed in is a factor. We looked at Python, C, Arduino IDE. Python was a good option because it is easy to pick up and we already knew how to code in that language. In addition, it had a large amount of useful libraries. However, Python would be overkill due to the system we would have to use to code it on like a Raspberry Pi. When looking into microcontrollers, we looked into the Arduino which was programmed through the Arduino IDE but this was also ruled out due to us not needed to use an Arduino for our hardware. Lastly, we looked into C and decided to use that because the microcontroller we would be using is programmed using this language. It also provides lower level hardware control making it more efficient for our design. However, we lack experience in this language so some researching will have to be performed. In addition to an environment, we have some tradeoffs on how we will be coding this. Essentially, what algorithms we will be using. One set of algorithms we could use would be what is called Greedy Algorithms. These algorithms focus on local steps that reach an optimal solution. In terms of our code it would be doing all the formatting and conversions in one block of code. This is very inefficient and may be hard to follow. Another way to approach the code, is by using algorithms that would divide and conquer. This form of coding takes tasks and splits them up into functions to solve those individual functions and takes their outputs back the main code. This is a very efficient way to code because it has a well-defined path to follow and errors can be easily seen through monitoring outputs of those functions. Lastly, we could use a form of programming using dynamic algorithms, that divide a main problem into smaller overlapping sub-problems. This is a more efficient way to code than divide and conquer due to the fact that functional outputs are fed into the next functions instead of having to reference back to main. This provides a more streamline tree like code which can still be monitored for errors at separate functional jumps. We plan on choosing the dynamic algorithms to make a chain of sub problems to produce our correct audio tone to be fed into the radio.

C. Preliminary Design

To set our project design and development in the right direction, a thorough preliminary design is required. The tools used to analyze our current project design for this section of the paper are: Flowcharts, schematics, diagrams, and Failure modes and Effect Analysis (FMEA). Appendix D consists of all the analytical tools used and descriptions of what they entail.

The preliminary design flowcharts serve as a way to subdivide our functions of the project into subfunctions that can each be described. Our design as of right now includes a microcontroller STM32 that is our main hardware for data processing. The flowcharts describe what this microcontroller will do in its half-duplex operation modes of receiving and transmitting. On the receiving side, the micro controller will receive an FM modulated audio tone from the radio and will then be demodulated and process down into a readable data stream packet to be sent to the PC. On the transmitting side, the micro controller will receive a bit stream from the PC and format it by following protocols. This formatted data packet will then be FM modulated onto a carrier audio tone and be sent off to the radio for transmission. The flowcharts provide us with a path to follow when coding this microcontroller to perform these formatting subfunctions. This is an efficient and effective way of software development.

To begin designing our hardware to be small and streamline we used schematics and simulations. The simulations were used to test our designs as we make them. To verify with correct voltage and current inputs we would be able to have the correct outputs from our design. The simulation software we used are called Fritzing for micro controller wiring as a whole and a website called Falstad for more intricate circuit component design like the Push to Talk (PTT) function. This software actually showed us that when using a P-channel MOSFET, the driving voltage needed to be 15 volts to produce the 15-volt output when the switch is pushed. This cannot be done because the max the micro controller without an amplifier can produce is 3.3 volts. So, we changed and now are looking more into a BJT for the PTT. This also assisted in our design of the voltage divider to reduce the voltage of our output signal from 3.3 volts to 500 Millivolts. Having the components and circuits laid out separately and being able to test and simulate them provides us with insight and confirmation that our design will work and how it will work.

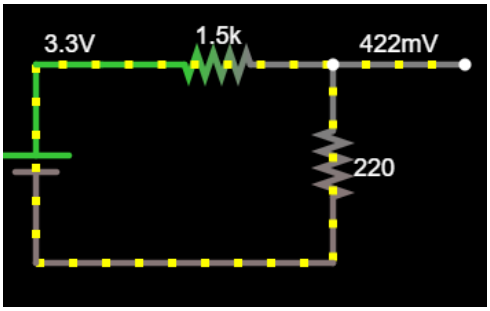


Figure 3. Simulated Voltage Divider Circuit

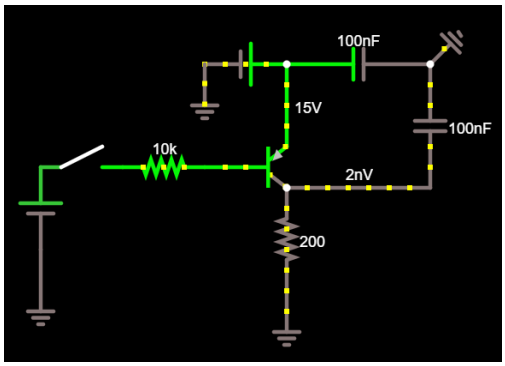


Figure 4. Simulated PTT BJT Circuit

With a preliminary design comes some potential concerns or failures that need to be addressed. The process of FMEA allows for us to analyze these potential failures and concerns and address them before they occur with proper solutions. We created a table that describes the potential failures along with the solutions we think would solve these failures. We also describe what could cause these failures and why. This analysis helps us prep for any early potential disasters, as well as provide us with insight on methods that could be used to deal with these issues if they were to come up in development process.

Now that all these tools have been used and the data gathered has been analyzed we can predict the success of our design as well as be prepared for any failures in the future. Our preliminary system is as such: on the receiving end we will receive an FM modulated audio tone from the radio through our 2.5 mm audio jack, the radio will know when to turn on and give us this data from our push to talk circuit shown in Figure 4. After the audio tone is received at the micro controller the software flowchart will come into play. We will demodulate the signal and translate it down into a bit stream which should be in AX.25 protocol format. This packet will then be analyzed for validity using the FCS bits. If the packet is valid the packet will be received and translated into a KISS packet and sent to the PC through RS232/USB interface. On the transmitting end, we will receive a packet from the PC through the USB interface to the STM32. The STM32 will then follow the flowchart and gather the payload from that packet. It will then take that payload as well as some addition specifications and format it into the AX.25 protocol format. In this process, we will generate the FCS bits for error checking. Lastly, the packet will be translated into an FM audio tone and transmitted to the radio to be sent off. The components and strategies to perform these tasks are subject to change past this preliminary design point throughout the testing process.

REFERENCES

[1] Beech, W. A., Nielsen, D. E., & Taylor, J. AX.25 Link Access Protocol for Amateur Packet Radio. (1997). PDF. Tucson .

[2] Chepponis, Mike, and Phil Karn. "The KISS TNC: A Simple Host-to-TNC-Communications-Protocol,"January1987. http://www.ax.25.net/kiss.aspx.

[3] Destevez. "KISS, HDLC, AX.25 and Friends." Daniel Estvez, May 6, 2018.https://destevez.net/2016/06/kiss-hdlc-ax-25-and-friends/.

[4] Langner,John. "Dire Wolf Software TNC." March 2018. https://microhams.blob.core.windows.net/content/2018/03/MHDC2018-WB2OSZ.pdf . PDF. MHDC

Kaleb P. Leon was born in New Iberia, Louisiana in 1998. Attended New Iberia Senior High for high school and graduated Valedictorian. Currently a senior attending University of Louisiana at Lafayette concentrating in Computer Engineering and minoring Computer Science. Mr. Leon is also part of a team with a publication in on Cyber Physical Security of Electric Vehicles and was part of a team awarded Louisiana Clean Fuel Leader Award for most innovative project of the year in 2017. He anticipates graduating in Fall 2020 and is looking for government jobs in Cyber Security or Security Engineering.

Kobe T. Keopraseuth was born in New Iberia, Louisiana in 1998 and attended Westgate High School. After graduating he enrolled in the University of Louisiana at Lafayette in the fall of 2016. Since 2018, he's been a member of IEEE. At the time of writing this paper he is a senior pursuing a Bachelor of Science in Electrical Engineering. He is expecting to graduate in the Fall of 2020.

David L. Cain was born in Hallsville, Texas in 1997 and attended New Iberia High School. Currently a senior at the University of Louisiana at Lafayette, majoring in Electrical Engineering and minoring in Computer Science. Mr. Cain has been a Member of IEEE since 2017. Expected to graduate in the Fall 2020.

APPENDIX A

A. Scope of Work

The purpose of this project is to design a TNC that has its basic functions implemented mostly if not solely in the form of software. The TNC must be able to receive a FM audio tone signal and translate it into binary (HDLC packet). Then, take that binary and perform an error check to see whether the binary packet is valid. If the packet is valid it must extract the payload and form the packet into another format (KISS) to send to the PC. The TNC must also be able to receive a packet in the form of KISS from the PC and form the appropriate HDLC packet. It will also need to form the error checking bits and translate the whole package into a FM audio tone to be sent over the air. This must be done primarily using software with little to no hardware. This design should be easily used by any experience hardware TNC user and provide convenience in its size and lack of power consumption.

B. Functional Requirements

Receiving

1. Receive Audio Tone Signal
2. Analog to digital convert into binary
3. Gather payload
4. Check for Errors
5. Send payload via KISS to PC

Transmitting

1. Receive KISS formatted data from PC
2. Follow AX.25 protocol to form data packet
3. Translate into analog audio tone signal
4. Send Audio tone to radio

Category	Requirements
Power	3.3 VDC to 5.5 VDC 10 mA
Environmental	-30 to 70 deg C <2x2 Inches
Audio Input	50 mV into 1K ohms BER 10-3 @ 6db snr
Audio Output	400 mV into 1K ohms
Protocol	KISS Mode AX.25 HDLC
Digital Input	3 Volt UART
Digital Output	3 Volt UART
LED Indicators	PTT indicator RX good Packet Energy in Audio Passband
PTT (Push to Talk)	Active High & Low Supply and Sink 20 mA Accept 3 to 15 Volts

C. Objective Tree

We have created an objective tree shown in Figure A-1 below. This is used to visualize our main priorities in our design. Communications is massive part of this design. This is due to the TNC needing to communicate with two different systems. The TNC receiving packets from PC, transmitting to radios, receiving from radios, and transmitting to PC each contribute a quarter of communications. Each subfield under these communication blocks, specify how the communication between these systems are possible. The compatibility is second but not to be neglected because the TNC needs to be able to communicate with other TNCs and radios to be properly functional, which each contribute a half of compatibility. Lastly, due to specifications from the sponsors low power consumption is desired but not required for total TNC functionality.

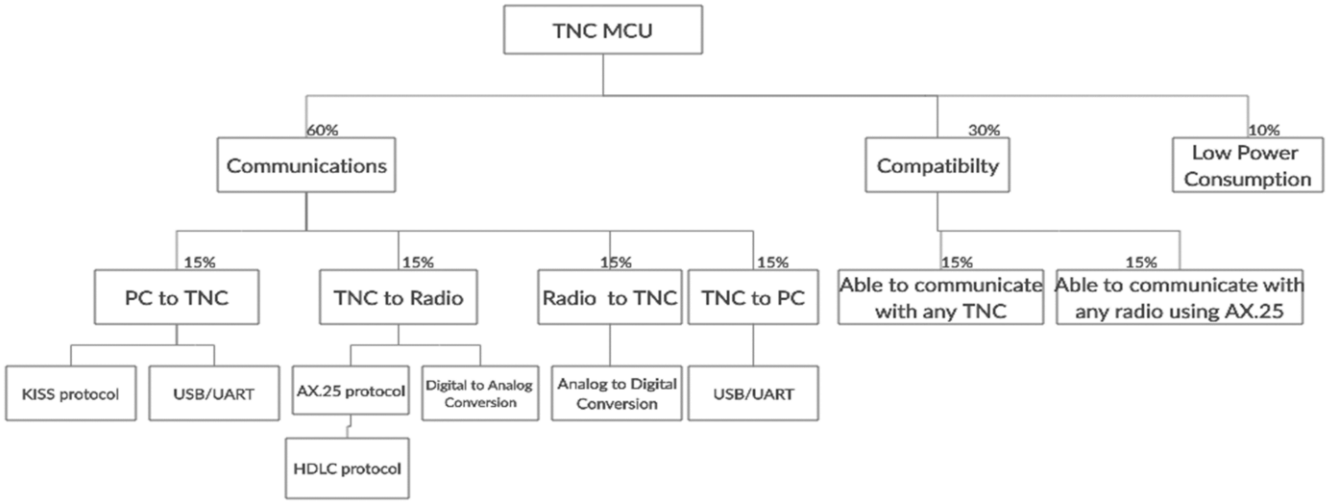


Figure A-1: Objective Tree MCU TNC

D. Level 0 Functional Block Diagram

Shown below is our Level 0 Functional Block Diagram, Figure A-2. This diagram shows the inputs and outputs in our base design. As shown our design functions in half duplex meaning it can only receive at one time and transmit at a separate time. Never at the same time. When receiving different tasks are performed as well as when transmitting.

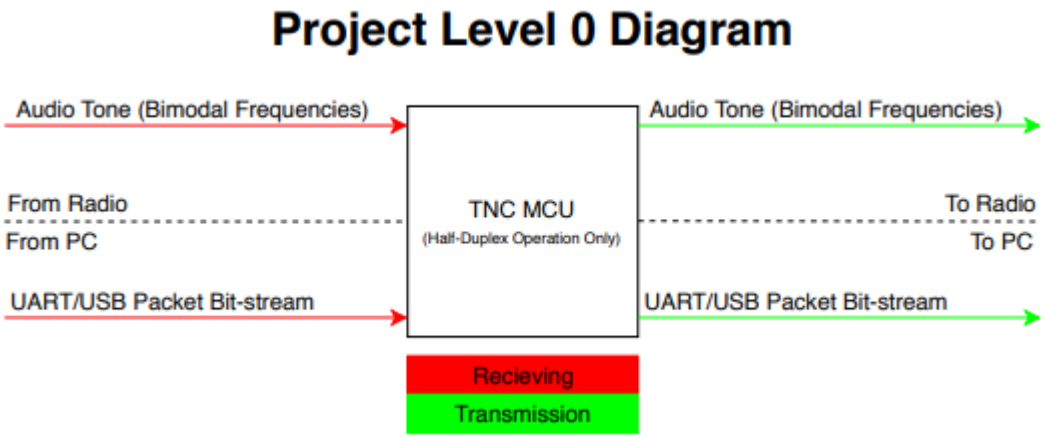


Figure A-2: Level 0 Diagram

E. Level 1 Functional Block Diagram

Our level 1 functional block diagram can be seen below in Figure A-3. This diagram is an extension upon our basic level diagram. The level 1 diagram provides greater detail on the signals coming into our system and how they are handled and processed to produce our desired outputs. Our diagram is split into two part: Receiving and Transmitting. These two parts DO NOT function at the same time making our design half duplex. These communications consist of two interfaces: a 3.5 mm audio jack of which we will receive and transmit our FM modulated audio tones through the radio and a RS 232/USB interface to gather bit stream binary data to be formatted to and from the PC. Inside of our system block, the main data processing occurs. Audio tones and bit streams travel into our STM32 micro controller to be formatted or unformatted for transmission and receiving. The PTT circuit is used to signal the radio when to turn on and off so that we may receive signals or transmit them.

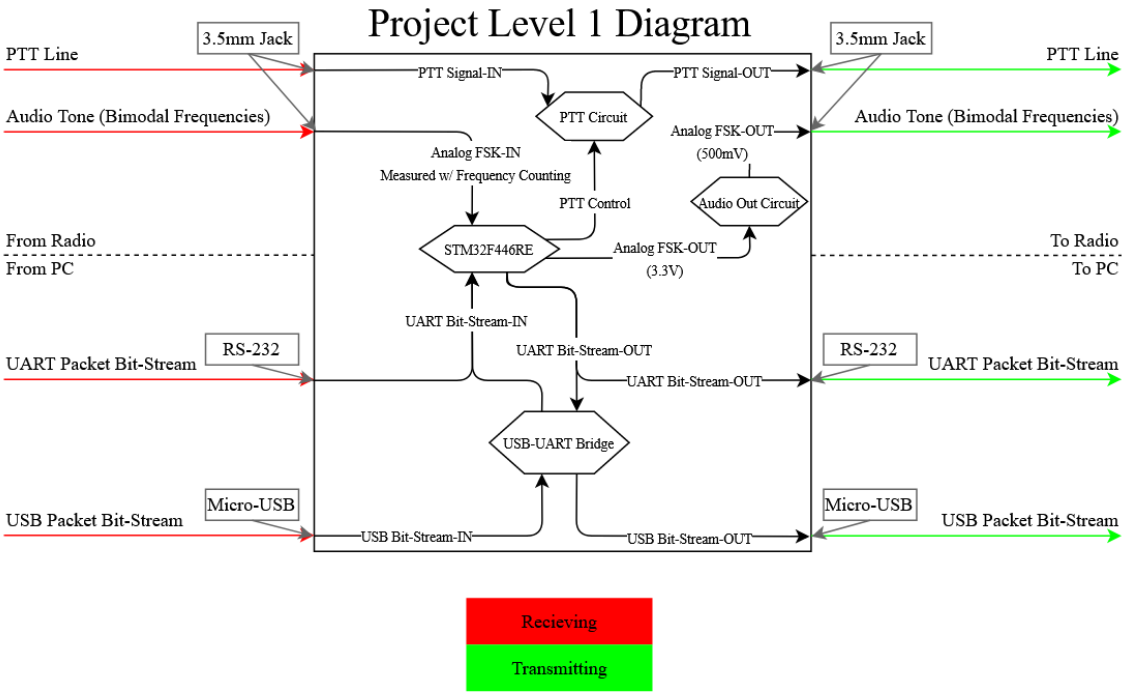


Figure A-3: Level 1 Diagram

APPENDIX B

A. Technological Analysis

The goal of the project aiming to replace hardware that was originally developed in the early 1980s, this means the technical requirements are minimal. The project is stated in a fashion that allows us to get the hardware in working order, then begin adding useful features and ensuring documentation is sufficient/competent for successive developers to branch off. The hardware for the project has only a few requirements:

- Interface with a terminal program via UART/USB
- Packetize data into AX.25 Protocol requirements
- Detect the frequency of an incoming audio signal
- Output an audio signal representing the packet bit stream
- Or output the packet bit stream via UART/USB

Tasks such as interfacing with UART or detecting the frequency of an audio signal not only have many options but also are very well documented. The process of packetizing the data will be an exercise of reading the documentation on how AX.25 packets are formed. The software environment on the Nucleo board is programmed in using C. We do not have any previous knowledge in C programming but we can learn fairly quickly thanks to guides and tutorials. The algorithms used to structure our code and produce our data are feasible to implement due to our previous knowledge of coding structure and data processing. Lastly, due to the system only needing to process at 1200 baud, which is fairly slow, efficiency in code should not be an issue to meet this spec.

B. Time Analysis

To assess the feasibility of time, a rough Gantt Chart has been attached to outline the goals of the team at various points of the design process. It is important to emphasize the design occurs over two semesters. For our project, the physical designing and testing will begin after a month of research on AX.25 and the accomplishments of other research groups. A goal of our project is to ensure documentation is sufficient for future groups to have a strong understanding of our system; to ensure this outcome, every three weeks we will dedicate time to ensuring the documentation is thorough and informative. With all of this, in addition to human resource/personal days on weekends and holidays, the critical path shows us finishing in one year with some spare time if we run into any issues that would directly delay the critical path.

C. Cost Analysis

Table A displays our list of components, along with their prices, and if we currently possess them. We possess most of the components personally and from our mentors, so the total cost of this project will be under $100, which is very feasible for our circumstances.

Item	Rented/Posses	Cost
STM32F446RE Nucleo board	No	$14.90
Arduino Uno	Yes	$0.00
LEDs	Yes	$0.00
MOSFET	No	$2.95
Resistors(22 kΩ, 4.7 kΩ, 470 Ω)	Yes	$0.00
Capacitor (100 nF)	Yes	$0.00
USB logic analyzer	Yes	$0.00
breadboard	Yes	$0.00
3.5 mm/2.5 mm jack	Yes	$2.09/$2.99
jumper wires(M-M, M-F,F-F)	Yes	$0.00
RS232 Cable	No	2 X ($4.29)
Total		$31.51

Table B-1: Cost Table

D. Regulatory Considerations

Our design will be using an audio jack to communicate with radios, which will be tested with ham and handheld radios. This is so that there is no interference between the TNC and radio's frequencies and outside frequencies.

E. Gantt Chart

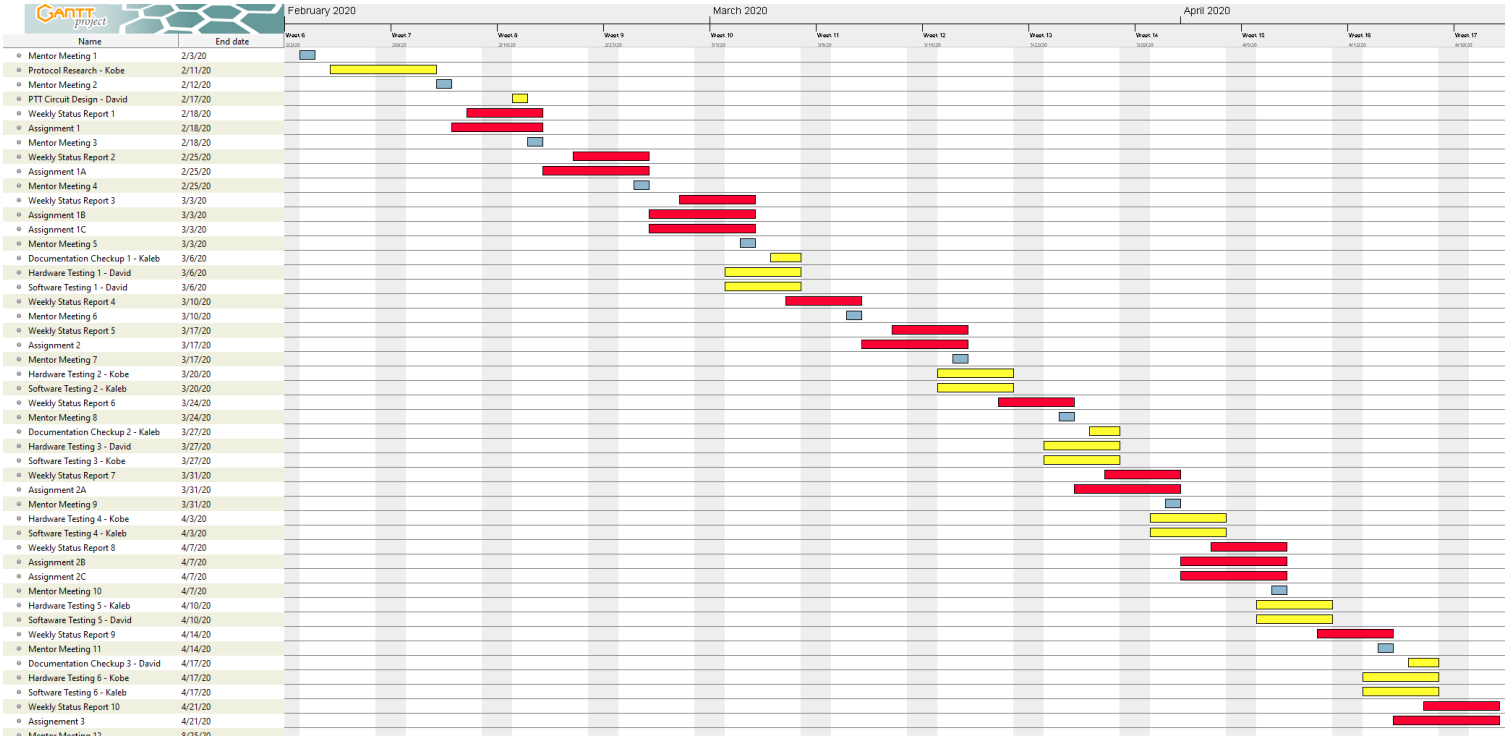


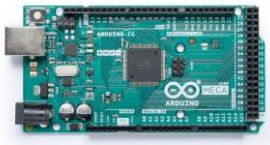
Figure B-1: Gantt Chart Semester One

APPENDIX C

A. *Alternatives and Tradeoffs Analysis*

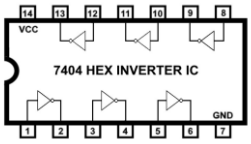
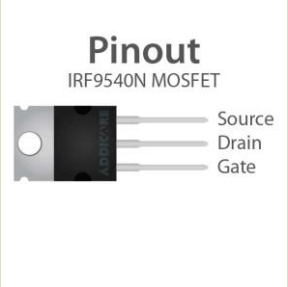
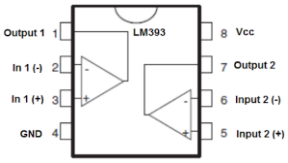
1) Microcontroller Comparison

After comparing the options listed below, the team decided on the STM32L4433 Nucleo board for the microcontroller that would format the receiving and transmitting data. Although we are all with the utilization of the Arduino Mega, it does not contain most of the functionalities we need to transmit and receive analog/digital data. We would have to implement more external components, which could possibly increase the design's size. The Teensey board, which had very similar features to the STM32L4433, was also declined since it did contain the CRC calculation unit. The STM32L4433 is the perfect fit for our design since it has DACs and ADCs for transmitting and receiving data. It also contains a CRC calculation unit which would help detect errors, if any, when transmitting or receiving. The only downside is that we are not familiar with coding in embedded C, but tutorials and guides can aid us in that aspect.

MICROCONTROLLER	**STM32L4433**	**Teensey 4.0**	**Arduino Mega**
			
Description	This microcontroller contains 16 external ADC channels, 1 12-bit ADC, 2 12-bit DAC output channels, an on board RTC, 2 CAN buses, 2 ultra-low-power comparators, CRC calculation unit, and a Schmitt trigger I/O.	This microcontroller contains 40 digital pins (all interrupt capable), 14 analog pins, 2 ADCs on chip, a RTC for date/time, an ARM Cortex-M7 at 600 MHz, 1024K RAM (512K is tightly coupled), and a 2048K Flash (64K reserved for recovery & EEPROM emulation).	This microcontroller contains 16 Analog read pins, 53 Digital pins,and 6 interrupt pins.
Cost	14.90	19.99	18.99
Pros	• Contains CRC calculation unit • Low Cost • Many GPIOs	• Fast clock speed • Has RTC	• Easy to use • Many GPIOs
Cons	• Embedded C programming	• Highest Cost • No CRC calculation unit	• Does not contain RTC • Does not contain DACs or ADCs

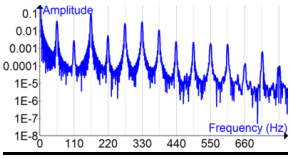
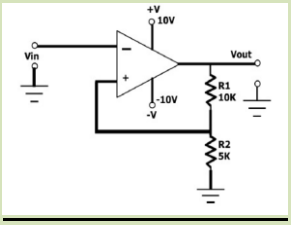
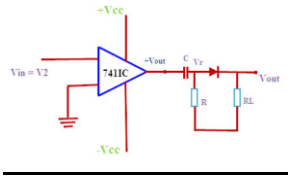
2) PTT

The radios that we are testing with will be outputting fifteen volts, and we need our automatic push to talk to be able input that voltage and send low signal to activate. The team decided to use a MOSFET to switch the mode on, since it is flexible in the voltage it takes in. Although the inverter IC and comparator IC are very simple to use, they may not be able to pass in twenty milliamps. The MOSFET on the other hand definitely meets our specifications.

Circuit Design	**Inverter Circuit**	**P-Channel MOSFET**	**Comparator IC**
			
Description	This IC outputs an inverted signal of the input.	Simple transistor gate to create the desired active low needed for radio circuits.	Many of these ICs feature several outputs: A<B, A>B and A=B. Using the greater than output, this would be an easy way to generate an inverting signal.
Pros	Since the device has built in digital logic, this means setting up a circuit for it to operate in would be much simpler than the MOSFET.	Using a MOSFET allows for flexible input voltage and does not add to power constraints.	Since the device has built in digital logic, this means setting up a circuit for it to operate in would be much simpler than the MOSFET.
Cons	Device may not be capable of supplying needed current for the system. Device should be capable of passing ~20mA	MOSFETs are capable of generating excess heat that may cause issues when design is compressed to <2x2in	Device may not be capable of supplying needed current for the system. Device should be capable of passing ~20mA

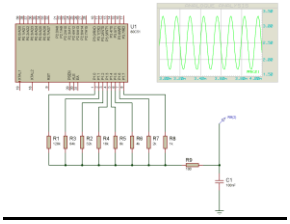
3) ADC

Since we are going for the STM32L4433 microcontroller we are using the Schmitt trigger as our ADC. The zero crossing implementation would be a simple circuit to use, but it would be an extra component we would have to purchase and it would take up extra space on our final board design. Although applying Fourier analysis would not involve using any hardware, applying it would be too much for the scope of this design. The TNC is only working with two different frequencies. The Schmitt trigger would save us money and space on our final design.

Signal Analysis Method	**Fourier Analysis**	**Schmitt trigger**	**Zero Crossing**
			
Description	In our case, this project would use this method to add multiple analog signals of different frequencies to generate digital signals	Simple transistor gate to create the desired active low needed for radio circuits.	Uses comparator to output a toggling logic signal when analog voltage goes to zero.
Pros	It doesn't involve any hardware.	Built in on STM 32 boards, great for filtering out oscillation in digital signal, when noise is in audio/analog signa	Easy implementation with a comparator
Cons	Not efficient because we would need multiple waves of different frequencies to generate a digital signal when we are only working with two different frequencies	If not built on microcontroller we would have to buy a comparator IC	Device may not be capable of supplying needed current for the system. Device should be capable of passing ~20mA

4) DAC

After comparing the options for the DACs, we decided to use the DAC built into our microcontroller. The resistor switching network would be more components we need to implement on our final design and we would have to spend more time coding the DAC. The DAC IC would be very simple to use in our design, but it would take more space in final design and add more to our cost. The DAC is already built in to our microcontroller so it would be more convenient.

Circuit Design	**Built into Controller**	**Resistor Switching Network**	**DAC IC**
			
Description	If the design were to include any of the STM32 line, the MCUs have built in DACs.	Would only consist of using ~4-6 GPIO, connected to different resistor values to represent variable step voltage output. This output would be passed through an LPF to generate a smooth sinusoid.	This would be using a dedicated High-Speed DAC ICs (such as DAC38RF82) that only requires digital input translated to an analog wave for us.
Pros	Similarly, to the dedicated IC, the benefit is there will only be a need to generate digital values.	Simplicity and lack of components needed to generate waveform at low power cost.	Ease of use, only needing to generate digital values that will quickly be converted to sinusoidal waveform.
Cons	Often built in DACs are slow and this may not work within the strict timing constraints of AX.25	Requirement to create code to drive a resistor network meaning more time would be spent on the DAC	With the dedicated silicon, this will raise the price and power consumption of the board.

5) Algorithms

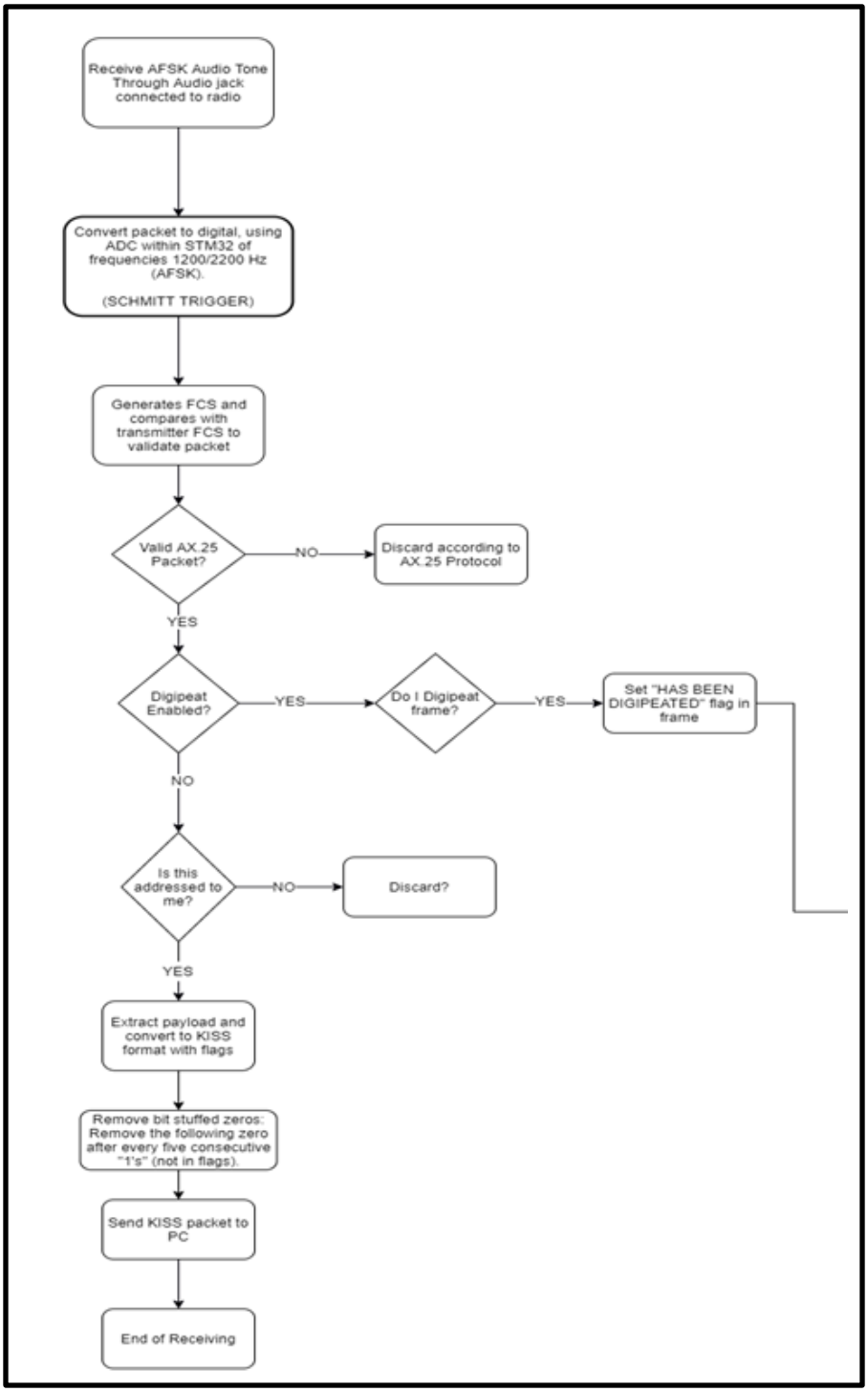
Since the entire project is software based, different algorithms used to develop our code to produce our output signal need to be considered. We took categories of algorithms and compared those. The algorithms discussed are: Greedy Algorithms, Divide and Conquer Algorithms, and Dynamic Programming. Greedy Algorithms focus on local steps that reach an optimal solution. In terms of our code it would be doing all the formatting and conversions in one block of code. This is very inefficient and may be hard to follow. Divide and conquer form of coding takes tasks and splits them up into functions to solve those individual functions and takes their outputs back the main code. This is a very efficient way to code because it has a well-defined path to follow and errors can be easily seen through monitoring outputs of those functions. Dynamic algorithms divide a main problem into smaller overlapping sub-problems. This is basically functions inside of functions that share results into the next steps. This is a more efficient way to code than divide and conquer due to the fact that functional outputs are fed into the next functions instead of having to reference back to main. This provides a more streamline recursion tree like code which can still be monitored for errors at separate functional jumps. We plan on choosing the dynamic algorithms to make a chain of sub problems to produce our correct audio tone to be fed into the radio.

APPENDIX D

Preliminary Design

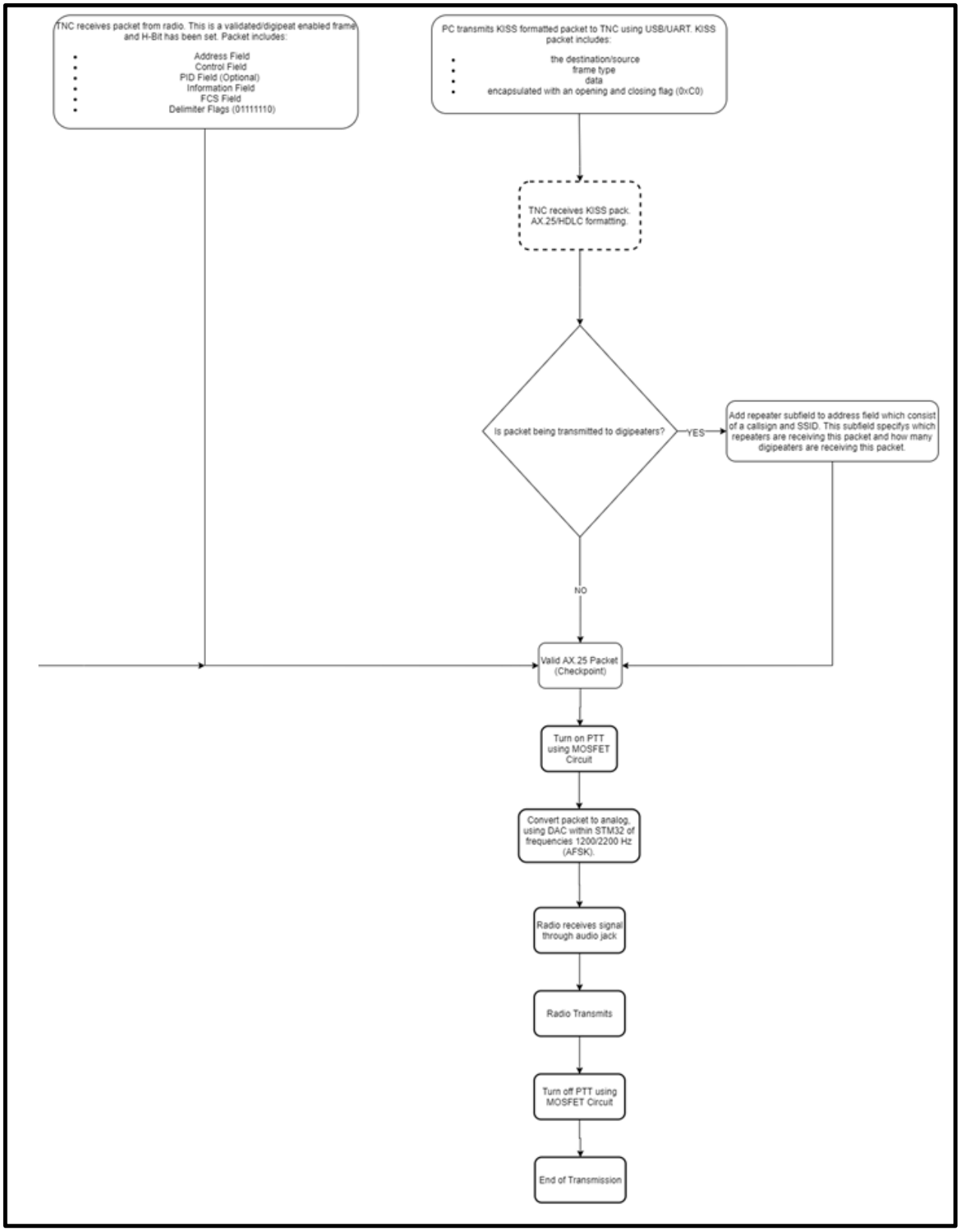
A. Receiving Flowchart

The following flowchart displays the process for receiving an audio tone from the radio. The TNC will first convert the received packet to digital, then check for any bit errors by comparing FCS fields between transmitter and receiver. If it is a valid packet, then it will check if digipeat is enabled. If digipeat is enabled, then it points to the "valid AX.25 Packet (checkpoint)" bubble in the transmitting flowchart. If digipeat is disabled, it will then check if the TNC should be receiving the packet and convert it to a KISS packet after removing the bit stuffed zeros if any.



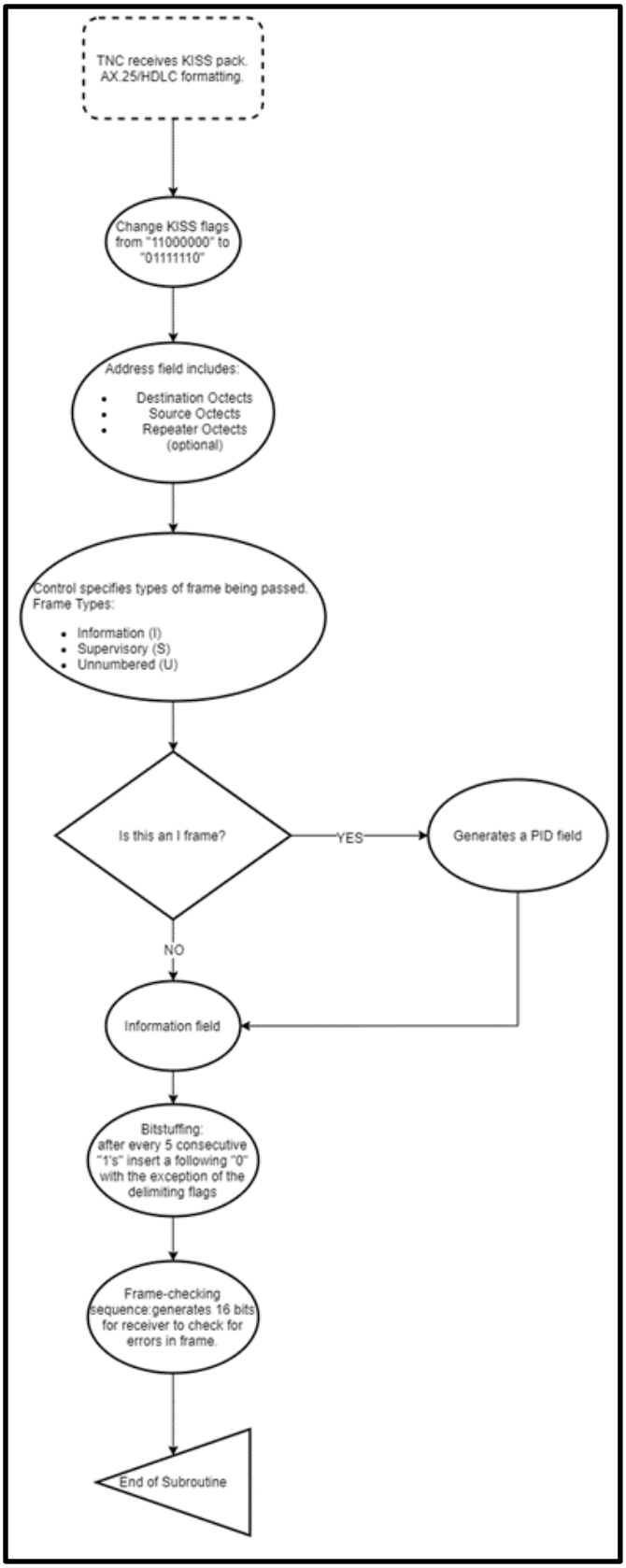
B. Transmitting Flowchart

The following flowchart displays the process for transmitting an audio tone to the radio. The PC first sends the KISS formatted packet to the TNC, through USB/UART, with the specified subfields in the packet. Then, it puts the packet in AX.25/HDLC format as shown in "AX.25/HDLC Formatting Flowchart." If the packet needs to be transmitted to any digipeaters it will then require a repeater subfield in the address field. When the packet is valid, it will then start the transmitting process. It first turns on the push to talk button, then converts the HDLC packet to an analog signal using ASFK. Once the radio receives the signal, the push to talk button then turns off. This flowchart also includes the case where the TNC is receiving from a radio or if digipeat is enabled.



C. Packet Formatting Flowchart

The following flowchart displays process for formatting a KISS packet into an HDLC packet. It first changes the KISS flags from “11000000” to “01111110.” Then an address field is created, using address bits from payload, which includes the packet’s destination, source and repeater (if any). Then the packet includes what type of frame it is and if it is an I frame, then it will generate a Protocol Identification field. Then if there are any five consecutive “1’s” in any field other than the flags, it will add bit stuffed “0’s.” Then an FCS field will be generated to check for any errors between transmitter and receiver.



D. Failure Modes and Effect Analysis (FMEA)

The FEMA table below displays the potential failures that can disrupt the functionality of our project. It will help guide us to make sure our project is fool proof. The first column displays what part of the project is not functioning correctly. The second column displays the potential problem that can happen with respect to the input. The third column displays what negative effects the potential failure can cause. The fourth column displays what can cause the potential failures. The last column displays what actions to take in order to fix the issues. Since our project is mainly based on software, it is important our code and configuration for our STM32 microcontroller is properly done. Faulty code/configuration can result in issues for the transmission and receiving of our TNC. Packet formatting is also important, because invalid formats can result in misinterpretation from the systems receiving from our TNC. Also, since the "push to talk" is the only circuit, besides our microcontroller, it can possibly fail if the components used are not sufficient.

Process Step/Input	Potential Failure Mode	Potential Failure Effects	Potential Failure Causes	Action Recommended
Bits in packets	Receiving TNC/Computer mistakes bits for flags	-Misinterpretation of information from receiving end -Disposal of packet due to invalid size	-Bits, anywhere in payload of KISS packet, are arranged as "11000000" -Bits, anywhere not in flags of HDLC packet, are arranged as "01111110"	**Bit stuffing:** -In KISS mode, a "1" is added after every "00000" arrangement in payload. Receiving TNC removes added "1" after every "00000" -In a HDLC, a "0" is added after every "11111" arrangement. Receiving TNC removes added "0" after every "11111"
Packet format	-Invalid Packet Format: -Less than 136 bits in frame -Not bounded by opening and closing flags -Octect not aligned	-Inaccurate information received	-Code failure -Excess noise on the received audio to digital conversion	-Receiving TNC disposes packet -Rewrite Code
Transmitter	-Transmitter is kept on for an extensive amount of time	-Receiver is polling for an extensive amount of time for frames to be sent	-Delay in frames being sent	**-Inter-Frame Time Fill:** when necessary for a TNC to keep transmitter on while not sending frames, flags should be sent to fill in time between frames being sent
Microcontroller	**1.)** Transmits audio signals with improper frequencies **2.)** Receives audio signal with noise	**1.)** -Bit errors -Receiving TNC misinterprets data **2.)** -Bit errors in packets sent	**1.)** -Incorrect code/configuration **2.)** -Noisy environment	**1.)** -Reconfigure microcontroller or rewrite code **2.)** -check for good connections -Move to a less noisy environment
Push-to-talk (PTT)	**1.)** LED burns out **2.)** MOSFET gets too hot	**1.)** -User cannot tell if TNC is sending audio signal to radio. **2.)** -Can damage components near MOSFET -MOSFET can burn out and TNC cannot perform audio transmission	**1.)** -LED used is old **2.)** -MOSFET is consuming too much power -Insufficient MOSFET used to handle required Power -Improper capacitors and resistors used in PTT circuit	**1.)** -Replace old LED **2.)** -Add heat sink to MOSFET -Replace MOSFET with a better one -Reconsider using different resistors/capacitors in circuit

E. Wiring Schematics

Figure D-1 is a layout of our intended breadboard wiring schematic, figure D-2 is a schematic representation. As shown, it features our chosen STM32F446RE, and a simple PNP based amplifying circuit. A GPIO of the STM board will be used to drive the gate of the transistor; this will be used to generate the active-low, push-to-talk signal for the radio. There is a 100nF decoupling capacitor across the push-to-talk line, this is to help reduce RF noise that may affect the circuit performance. A PWM ready GPIO is used to output an analog waveform. This output is passed through a simple voltage divider to lower the peak voltage from 3.3V to 500mV; 500mV is the expected input of most radios. After the analog waveform voltage level has been reduced, it is passed through a 100nF coupling capacitor to remove DC from the audio tone. (Note: Testing can be done on a breadboard with neglect of the imperfections associated with this circuit construction method. This is due to the low frequencies involved with the intended signals.)

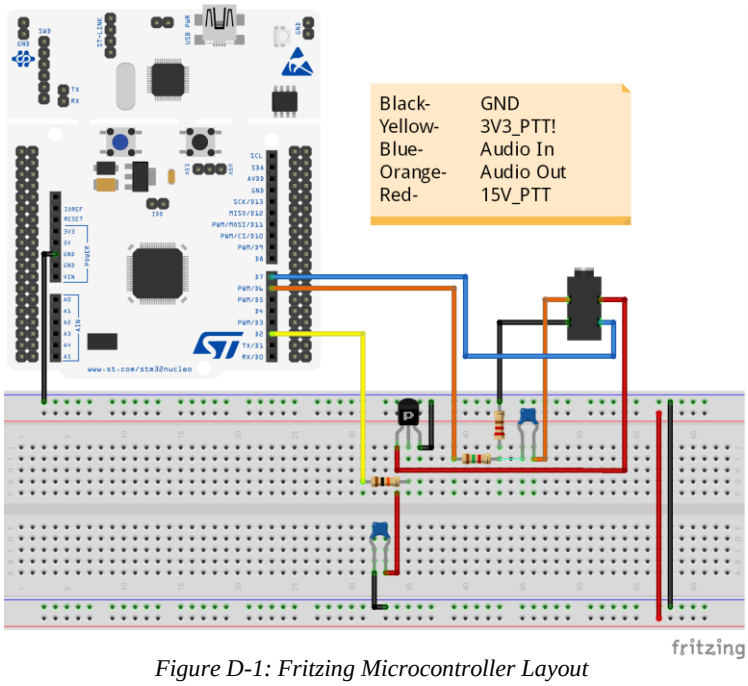


Figure D-1: Fritzing Microcontroller Layout

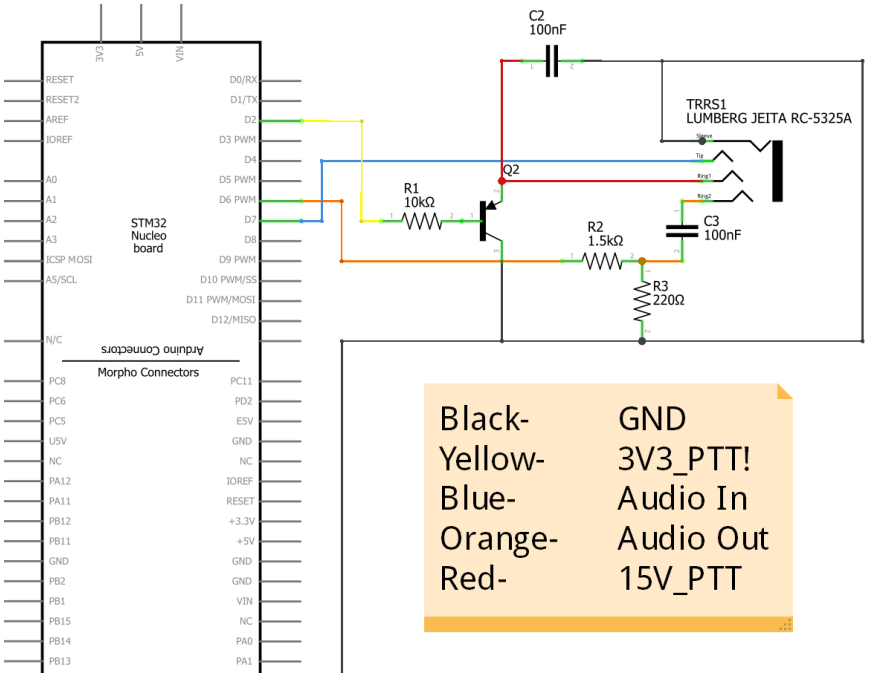


Figure D-2: Fritzing Microcontroller Connections