

Midterm Report

Design 2

Team 2

MCU TNC Design

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

October 19th, 2020

Summarizing Report

Step 1:

We reviewed the requirements stated for deliverable 4 and discussed how we would handle each. We also referred to the scholarly paper template and verified ours met the criteria.

Step 2:

We double checked that our Scholarly paper to be sure that it followed the recommended template.

Step 3:

We added our appendix F from deliverable 3 to the full scholarly paper and formatted it so that it fit the criteria formatting of the paper. We also made slight adjustments so it would look more professional.

Step 4:

We created a section and heading for the test reports and added our completed reports and made it modular so we can add the rest later.

Step 5:

We re-referenced our testing tree to further discuss how we would validate and verify our design. In addition, we discussed how so far, we have been instrumenting our code so that it will be easy to test.

Step 6:

We referenced our paper to see if our current description of the file system was accurate and added a file hierarchy from our GitHub file system. This allows for easy view of all of our files and organization. Plan to add readme's when design is complete.

Step 7:

We developed a change order form and discussed any components or systems that may need a form.

Step 8:

We retraced our steps to develop a methodology for how we worked on design in each section of our development.

Step 9:

We had a meeting with our mentors to discuss what they would like us to present for our final presentation.

Step 10:

We each checked our knowledge of the project to see if we were ready to answer questions.

Step 11:

We gathered all our weekly status reports so that we may use the timesheets for our work times. We also gathered all meeting notes from meetings with our mentors.

Step 12:

We discussed on what are the main points we wish to discuss in terms of our design and progress.

Step 13:

We studied the requirements for deliverable 4 and followed the guidelines as specified.

ABET Questions

1. How did this course provide you with an ability to identify, formulate, and solve complex engineering problems by applying principles of engineering, science, and mathematics?

This course did not really teach me how to apply science and mathematics however they were used in the projects design, so we did use them in this class. The class did teach us a multitude of things about the principles of engineering and how to use different design principles to identify, formulate, and solve complex problems. We learned about flowcharts, schematics, how to identify problems in our head and create scopes of work, and how to develop an FMEA table to aid us in development to avoid errors.

RANK: 1

2. How did this course provide you with an ability to apply engineering design to produce solutions that meet specified needs with consideration of public health, safety, and welfare, as well as global, cultural, social, environmental, and economic factors?

This course required that we analyze the effect of our project on the outside world and consider many factors. However, the only one for our project was radio frequency interference and our ranges are not particularly high enough to affect any important bands. In Design II, we will be looking into environmental factors that could affect our project and work to fix and prepare for these factors.

RANK: 3

3. How did this course provide you with an ability to communicate effectively with a range of audiences?

We learned to communicate efficiently with many different types and subclasses of people. We were tasked consistently with communicating with our fellow peers and instructors as well as our mentors. In addition, solely writing the scholarly paper improved our vocabulary as engineers for professional scenarios to come.

RANK: 4

4. How did this course provide you with an ability to recognize ethical and professional responsibilities in engineering situations and make informed judgments, which must consider the impact of engineering solutions in global, economic, environmental and societal contexts?

This course provided us with the opportunity to work on a project that could benefit many others in an engineering world we were barely familiar with. We got to see the positives and negatives that come in the radio hams world and are glad we could help with the engineering problem given to us.

RANK: 2

5. How did this course provide you with an ability to function effectively on a team whose members together provide leadership, create a collaborative and inclusive environment, establish goals, plan tasks, and meet objectives?

This course provided us with a platform not only to work closely with our team members but work along side accomplished engineers and figure out how they function and lead. This course also provided us ways of planning out a project and establishing goals to finish a desired goal by a period of time. This also included splitting up tasks and using software like Gantt chart to predict our timeline.

RANK: 6

6. How did this course provide you with an ability to acquire and apply new knowledge as needed, using appropriate learning strategies?

This course allowed us to think critically and use the design tools provided to us to plan our project. We went into this project not knowing much about the subject and through research and dedication encouraged by this class we were able to learn a lot and work towards solving a problem.

RANK: 5

Table of Contents

I. INTRODUCTION	1
II. RESEARCH OF WORK DONE BY OTHERS	1
A. KISS MODE.....	1
B. HDLC (HIGH-LEVEL DATA LINK CONTROL) PROTOCOL.....	1
C. AX.25 PROTOCOL	2
D. AFSK (AUDIO FREQUENCY SHIFT KEYING).....	2
III. PROJECT ANALYSIS	2
A. PROJECT FEASIBILITY	2
B. ALTERNATIVES AND TRADEOFFS CONSIDERATIONS.....	2
C. PRELIMINARY DESIGN.....	3
D. FINAL DESIGN	4
REFERENCES	5
TABLE A-1: SPECIFICATIONS FOR DESIGN.....	6
APPENDIX A.....	6
A. SCOPE OF WORK.....	6
B. FUNCTIONAL REQUIREMENTS	6
C. OBJECTIVE TREE	7
FIGURE A-1: OBJECTIVE TREE MCU TNC	7
E. LEVEL 0 FUNCTIONAL BLOCK DIAGRAM.....	8
FIGURE A-2: LEVEL 0 DIAGRAM	8
F. LEVEL 1 FUNCTIONAL BLOCK DIAGRAM.....	8
FIGURE A-3: LEVEL 1 DIAGRAM	8
APPENDIX B.....	9
A. TECHNOLOGICAL ANALYSIS.....	9
B. TIME ANALYSIS.....	9
C. COST ANALYSIS	9
TABLE B-1: COST TABLE	9
D. REGULATORY CONSIDERATIONS	9
E. GANTT CHART.....	10
FIGURE B-1: GANTT CHART SEMESTER ONE.....	10
APPENDIX C.....	11
A. ALTERNATIVES AND TRADEOFFS ANALYSIS.....	11
1) <i>Microcontroller Comparison</i>	11
2) <i>PTT</i>	12
3) <i>ADC</i>	13
4) <i>DAC</i>	14
5) <i>Algorithms</i>	15
APPENDIX D.....	16
A. RECEIVING FLOWCHART	16
B. TRANSMITTING FLOWCHART.....	17
C. PACKET FORMATTING FLOWCHART	18
D. OSI LAYERED COMMUNICATIONS MODEL	19
E. FAILURE MODES AND EFFECT ANALYSIS (FMEA).....	20
F. WIRING SCHEMATICS	21
FIGURE D-5. FRITZING MICROCONTROLLER LAYOUT	21
FIGURE D-6. FRITZING MICROCONTROLLER CONNECTIONS.....	21
APPENDIX E.....	22

A. DOCUMENTATION NUMBERING CONVENTION AND FILE STRUCTURE	22
FIGURE E-1. FILE STRUCTURE	22
FIGURE E-2. FILE STRUCTURE CONTINUED	23
FIGURE E-3. FILE STRUCTURE CONTINUED	24
B. LEVEL 1 DESIGN DIAGRAM	25
FIGURE E-4. LEVEL 1 DESIGN DIAGRAM	25
C. CODE BREAKDOWN	26
1) <i>Kiss Packet Interpretation</i>	26
2) <i>Analog Audio Tone to Digital Data Conversion</i>	26
FIGURE E-5 THRESHOLD TRIGGER EXAMPLE	26
3) <i>Digital Data to FSK Audio Tone Conversion</i>	26
D. BILL OF MATERIALS	27
E. ASSEMBLY SPECIFICATIONS	27
FIGURE E-6. SIMPLE EXTERNAL CONNECTIONS	27
FIGURE E-5. BREADBOARD WIRING SCHEMATIC	27
FIGURE E-7. SIMPLE ICON IDENTIFIERS	28
FIGURE E-8. FOLDER NAVIGATION	28
F. OPERATIONAL GANTT CHART	29
FIGURE E-9. OPERATIONAL GANTT CHART	29
APPENDIX F	30
A. TESTING PROCESS	30
B. TESTING TREE	31
C. TEST REPORT TEMPLATE	33
D. VALIDATE AND VERIFICATION PLAN	34
1) <i>Hardware</i>	34
2) <i>Software</i>	34
E. WORKMANSHIP ON DESIGN	34
F. FINAL PRESENTATION DEMO	34
G. COMPLETED TEST REPORTS	35
APPENDIX G	39
A. CHANGE ORDER FORM TEMPLATE	39
B. COMPLETED CHANGE ORDER FORMS	39
APPENDIX I	40
A. TIME SHEETS	40
1) <i>David Cain</i>	40
2) <i>Kobe Keopraseuth</i>	41
3) <i>Kaleb Leon</i>	42

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 Microcontroller 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.

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

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.

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.

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.

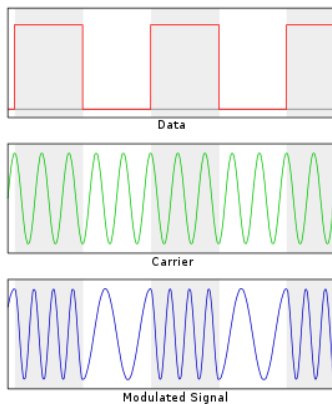


Figure 2. Audio Frequency Shift Keying

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 Chart 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 Teensy 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 number 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 since an Arduino is not needed 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 are used to test our designs as we make them. It will verify which voltage and current inputs are needed in order for our design to have the correct output. 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.

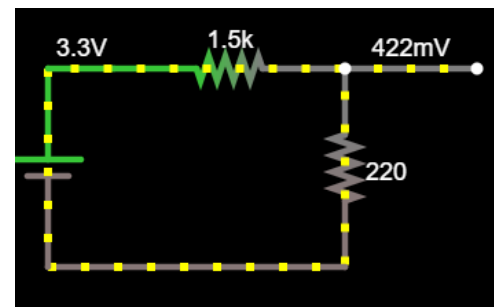


Figure 3. Simulated Voltage Divider Circuit

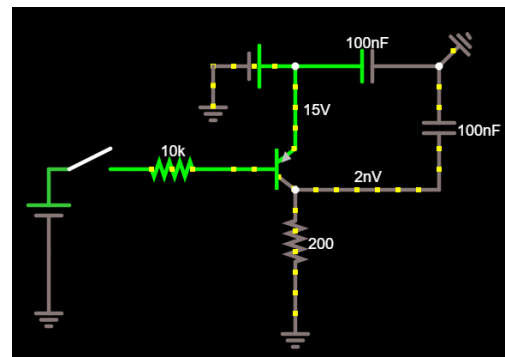


Figure 4. Simulated PTT BJT Circuit

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.

In addition to failures and potential concerns, many safety measures have to be taken into account. Since, our project can receive and transmit over the air there is potential danger of our messages interfering on different bands if we are not careful when transmitting. This is addressed by making sure our audio FM signals are modulated at the correct frequency bands. Another safety issue is meeting the voltage and current requirements of this project. We need to only use voltages up to 5.5 Volts DC with 10mA current to power the device and only output up to 400mV to the radio. These voltage requirements are meant to make sure we do not harm the devices we connect the TNC to with excess amounts of current and voltage. We handle this with the voltage divider in Figure 3. Lastly, the project needs to take into account environmental safety concerns. To be accessible and useful to many of the systems this project will be used it needs to have dimensions of less than 2inches by 2inches as well as be able to handle temperatures from -30 to 70 degrees Celsius. This will be handled by PCB design of our main hardware components as well as proper casing.

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 (frame check sequence) 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 additional 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.

D. Final Design

After review from our peers and mentors, there were not many issues to be addressed with our preliminary design. However, this version of the final design in terms of the hardware are subject to change due to optimization of components and board space. A custom PCB may be in the works for future versions of the TNC to provide an even more

compact and less power hungry set up. For the time being however, to perfect the software, the design will be consistent in using the STM as our main hardware for processing the software that mainly drives our project for the TNC system. These specifications for our software and hardware functionalities can be seen in Appendix E in greater detail. This consists of diagrams and flow charts in greater detail as well as some explanation of some of the main function blocks of our software. Appendix F covers a thorough Comprehensive Testing Plan. In Appendix F, the software will be thoroughly tested and validated over the next period of working on this project. This is done to make sure our software is fully functional to work with that necessary micro controller and components so that it is easily transferable to more optimized hardware if necessary.

The code of our project has been updated and added on to provide functions that accomplish many sections our preliminary design flowcharts. As we went through the software development many logic choices had to be made.

When receiving a KISS packet from the PC the TNC will extract the data section of the packet by removing the end flags and translating the data section from HEX to Binary. As we added this functionality, we tested whether we were getting the correct data by sending it packets that we knew what was inside before testing with an actual dumb terminal system. We would translate this Binary to ascii to make it readable again and output to serial to see if it was reading the correct data. The actual code would then just send off the binary after we verified these tests.

The binary data section would be sent off to a function block of the code where it would perform bit stuffing to produce an AX.25 packet. These packets were verified by a proprietary software provided to us and checked by our mentors. Once it was confirmed that our software was able to produce the correct AX.25 packets, these packets would be sent off to the DAC to be made into FSK audio tones.

The binary bit stream will then be sent to memory where it will be referenced and parsed for ones and zeros. When the code encounters a one the DAC will produce a sinusoidal signal at 2200 Hz and when the code encounters a zero the DAC is programmed to produce a sinusoidal signal at 1200 Hz. Each of these signals are made using an equation sent to the DAC based off the clock on the microcontroller to provide smooth signal transmission. In addition, another clock reference was made to see when to parse the next binary value and switch signals. This was based off the baud rate of 1200 baud. This rate means we will be sending values at a rate of one digit per 0.83333 milliseconds. This is the normal rate at which ones and zeros are interpreted and sent on this FSK modulation.

For receiving signals, we will be using a functionality of the microcontroller called the Schmitt Trigger. We create a variable to keep count of how many times the signal switches across the trigger. Once the timer counts up to the set period, the frequency is calculated. If the calculated frequency is 2200 Hz it will generate a one and if 1200 Hz a zero will be generated. The space between how multiple zeros or ones is calculated once again by the baud rate which is set to be analyzed based on the clock reference. The microcontroller will use the CRC calculation unit to generate a CRC value to compare with the

value in the FCS section in the packet. If these two matches the packet is valid. If they do not match the packet is discarded.

For testing and validation, we set up a small hardcoded bitstream to be sent out then we wire the output of the DAC back into the TNC to then demodulated the signal at 1200 baud and see whether we are receiving the same data we are sending out. After some adjustments and multiple tests, we were able to validate the send and receiving at the radio output.

Lastly, since this device is only to function in Half duplex, the portions where we will be receiving data to be sent to the PC will be sectioned off for one mode at one moment of time and the portions where we need to send out data will be in their own string of functions.

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," January 1987. <http://www.ax.25.net/kiss.aspx>.
- [3] Destevez. "KISS, HDLC, AX.25 and Friends." Daniel Estvez, May 6, 2018. <https://destvez.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



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.



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.

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

Table A-1: Specifications for Design

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

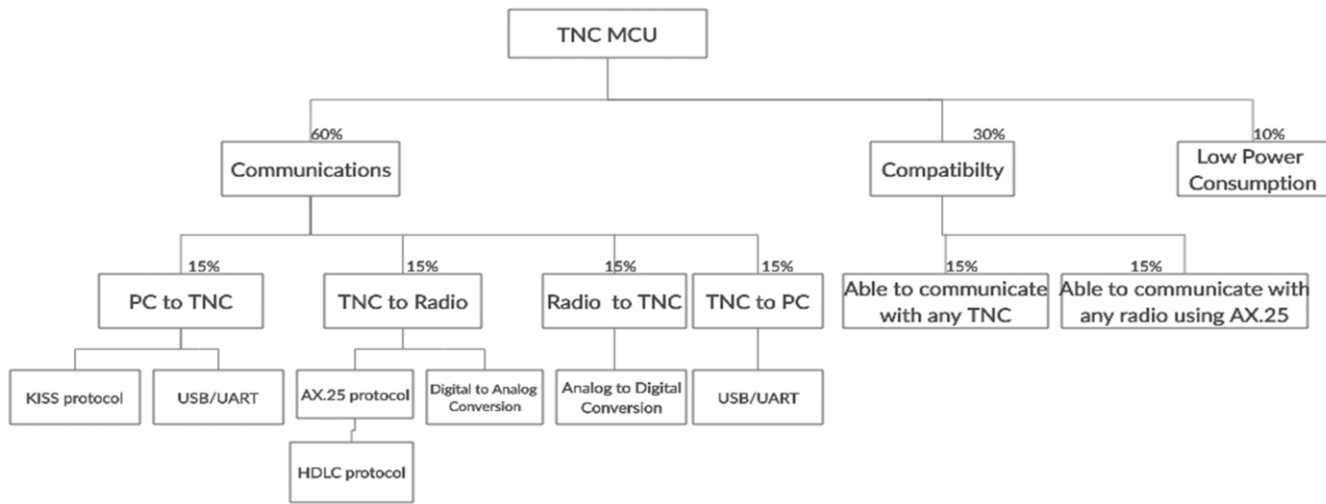


Figure A-1: Objective Tree MCU TNC

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.

E. 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.

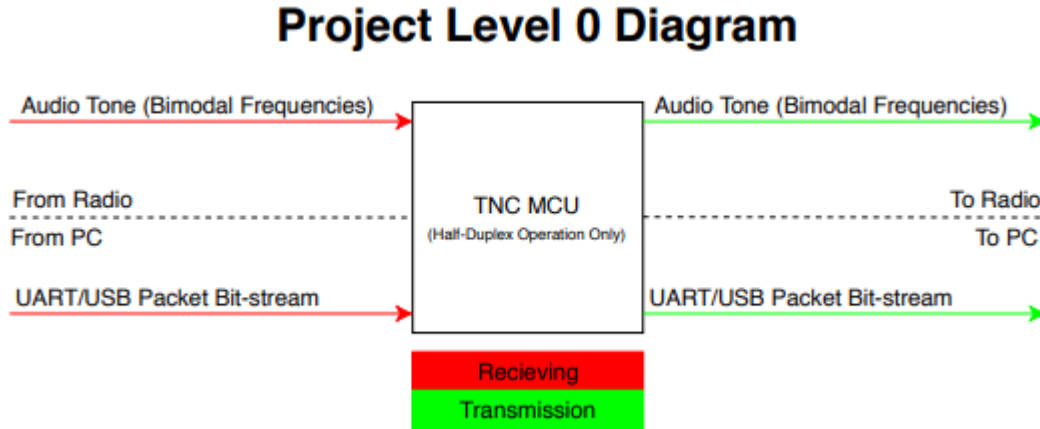


Figure A-2: Level 0 Diagram

F. 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 level 0 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. The main data processing occurs inside of our system block. 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.

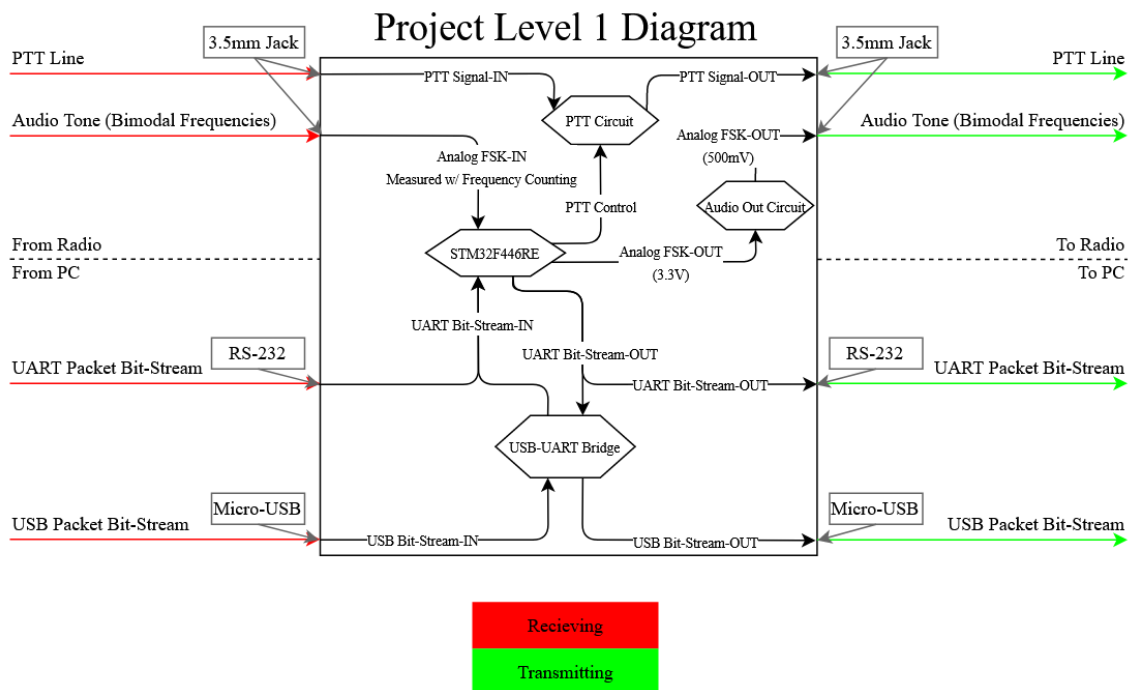


Figure A-3: Level 1 Diagram

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.

Table B-1: Cost Table

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 k Ω , 4.7 k Ω , 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

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

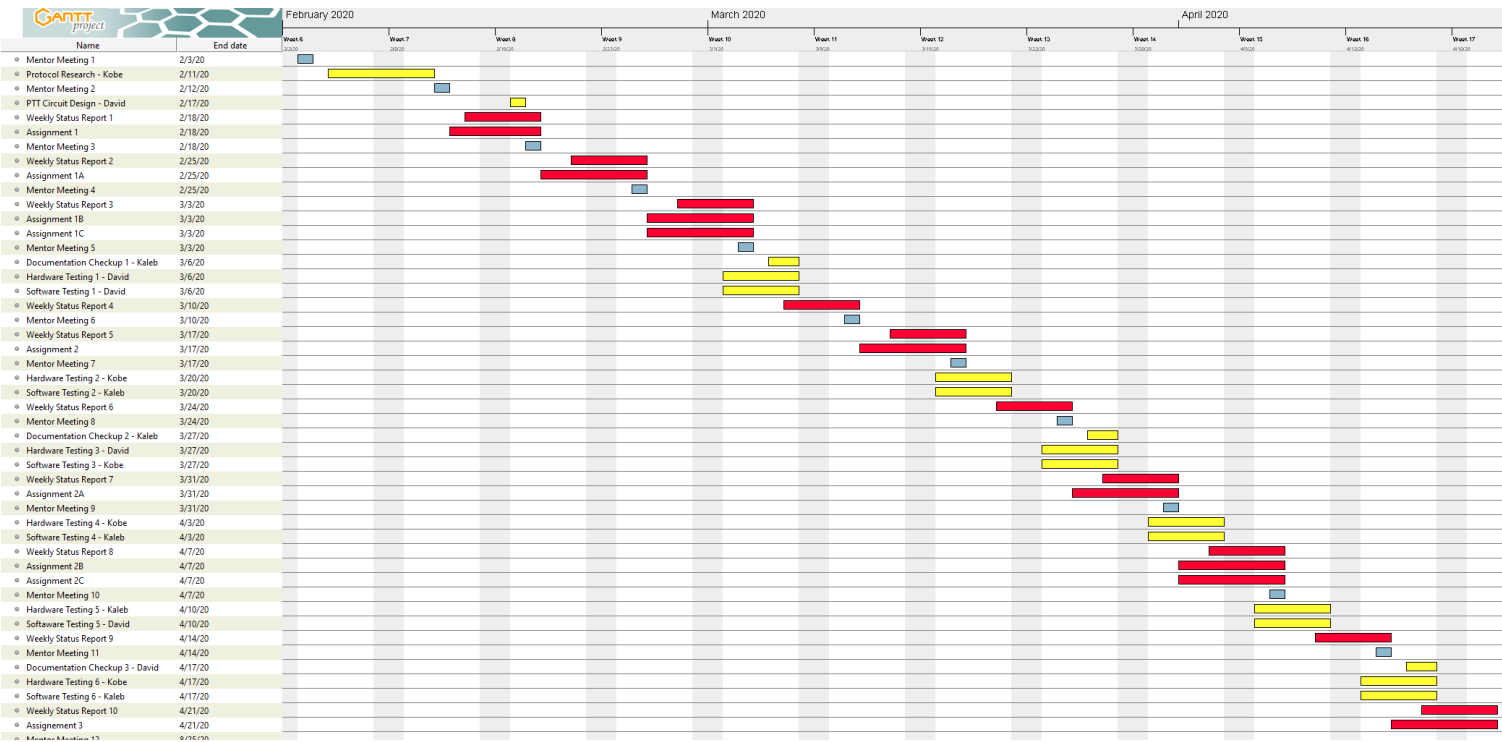





Figure B-1: Gantt Chart Semester One

APPENDIX C

A. Alternatives and Tradeoffs Analysis

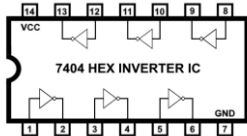

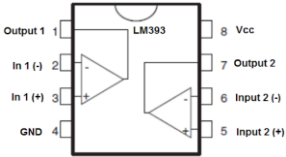
1) Microcontroller Comparison

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 Teensy board, which had very similar features to the STM32L4433, was also declined since it did not 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.

<u>MICROCONTROLLER</u>	<u>STM32L4433</u>	<u>Teensy 4.0</u>	<u>Arduino Mega</u>
			
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	<ul style="list-style-type: none"> Contains CRC calculation unit Low Cost Many GPIOs 	<ul style="list-style-type: none"> Fast clock speed Has RTC 	<ul style="list-style-type: none"> Easy to use Many GPIOs
Cons	<ul style="list-style-type: none"> Embedded C programming 	<ul style="list-style-type: none"> Highest Cost No CRC calculation unit 	<ul style="list-style-type: none"> Does not contain RTC Does not contain DACs or ADCs

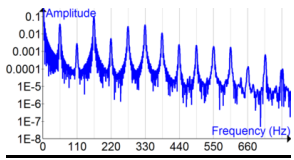
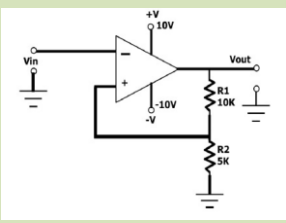
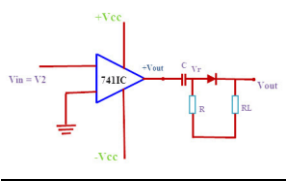
2) PTT

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.

<u>Circuit Design</u>	<u>Inverter Circuit</u>	<u>P-Channel MOSFET</u>	<u>Comparator IC</u>
			
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 $< 2 \times 2 \text{in}$	Device may not be capable of supplying needed current for the system. Device should be capable of passing ~20mA


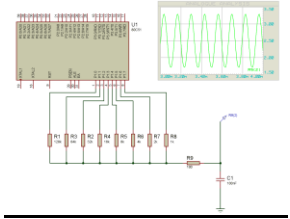

3) ADC

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.

<u>Signal Analysis Method</u>	<u>Fourier Analysis</u>	<u>Schmitt trigger</u>	<u>Zero Crossing</u>
			
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 signal	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 into our microcontroller so it would be more convenient.

<u>Circuit Design</u>	<u>Built into Controller</u>	<u>Resistor Switching Network</u>	<u>DAC IC</u>
			
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.

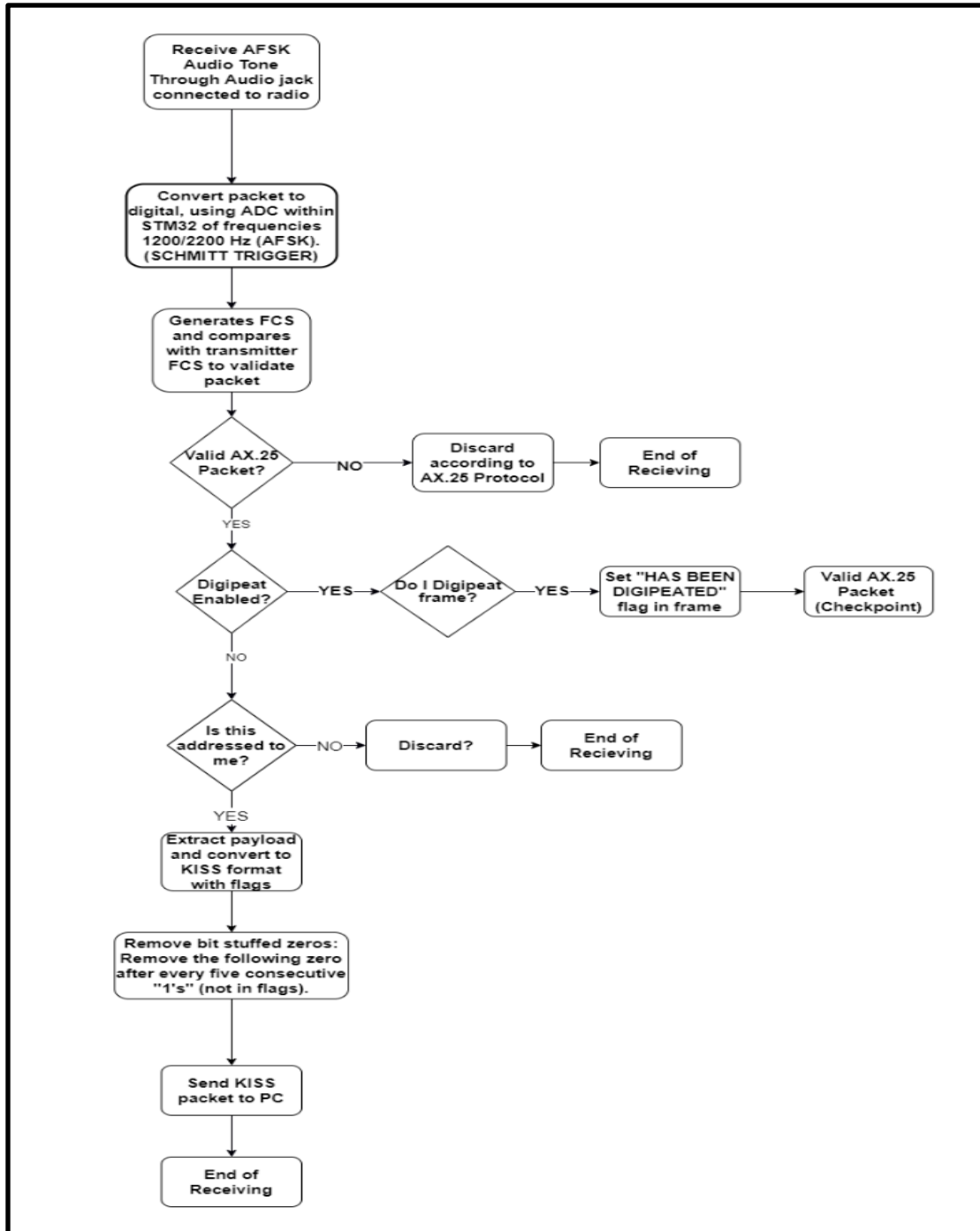


Figure D-1. Flowchart of the receiving process

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 AFSK. 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.

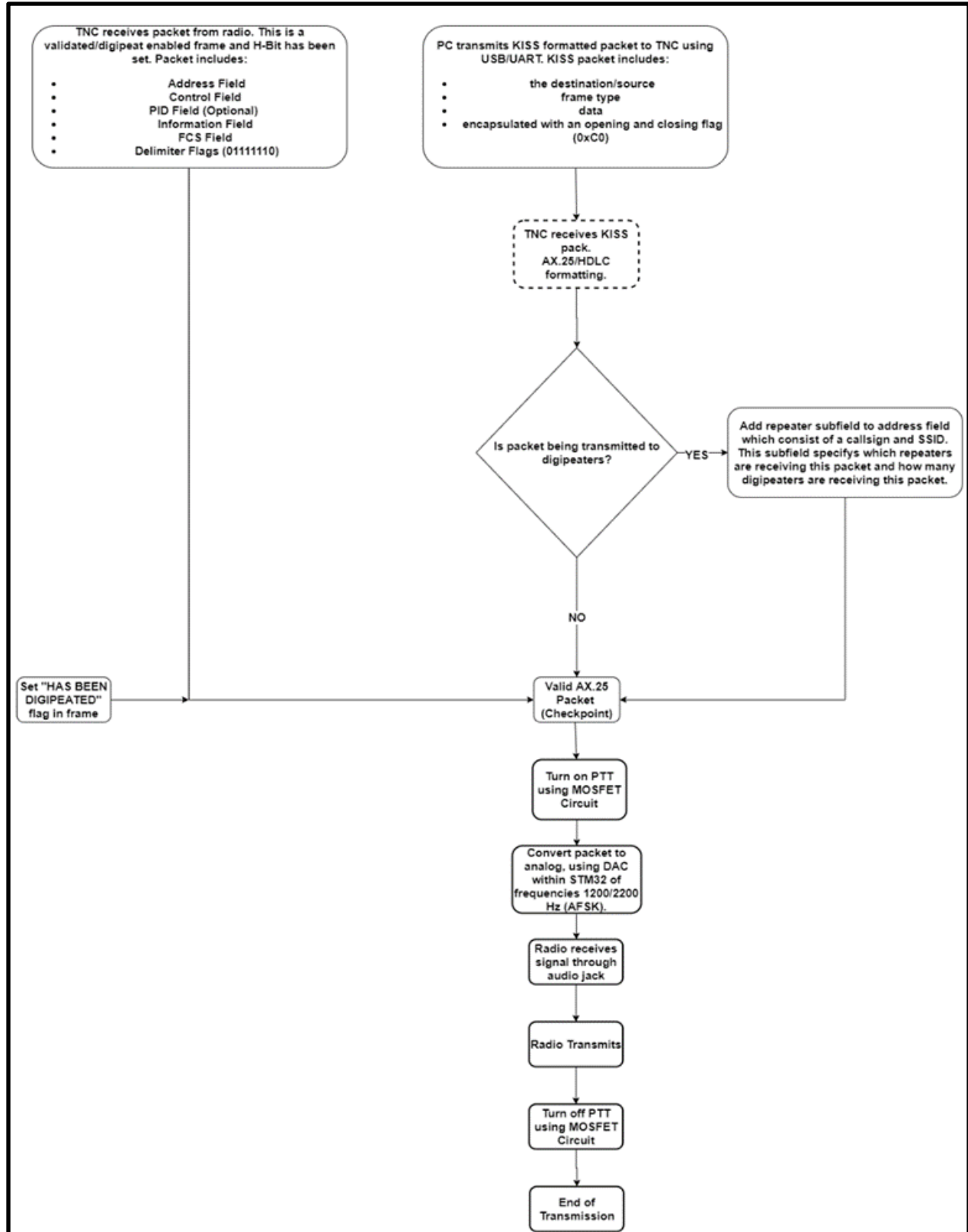


Figure D-2. Flowchart of Transmitting Process

C. Packet Formatting Flowchart

The following flowchart displays the 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.

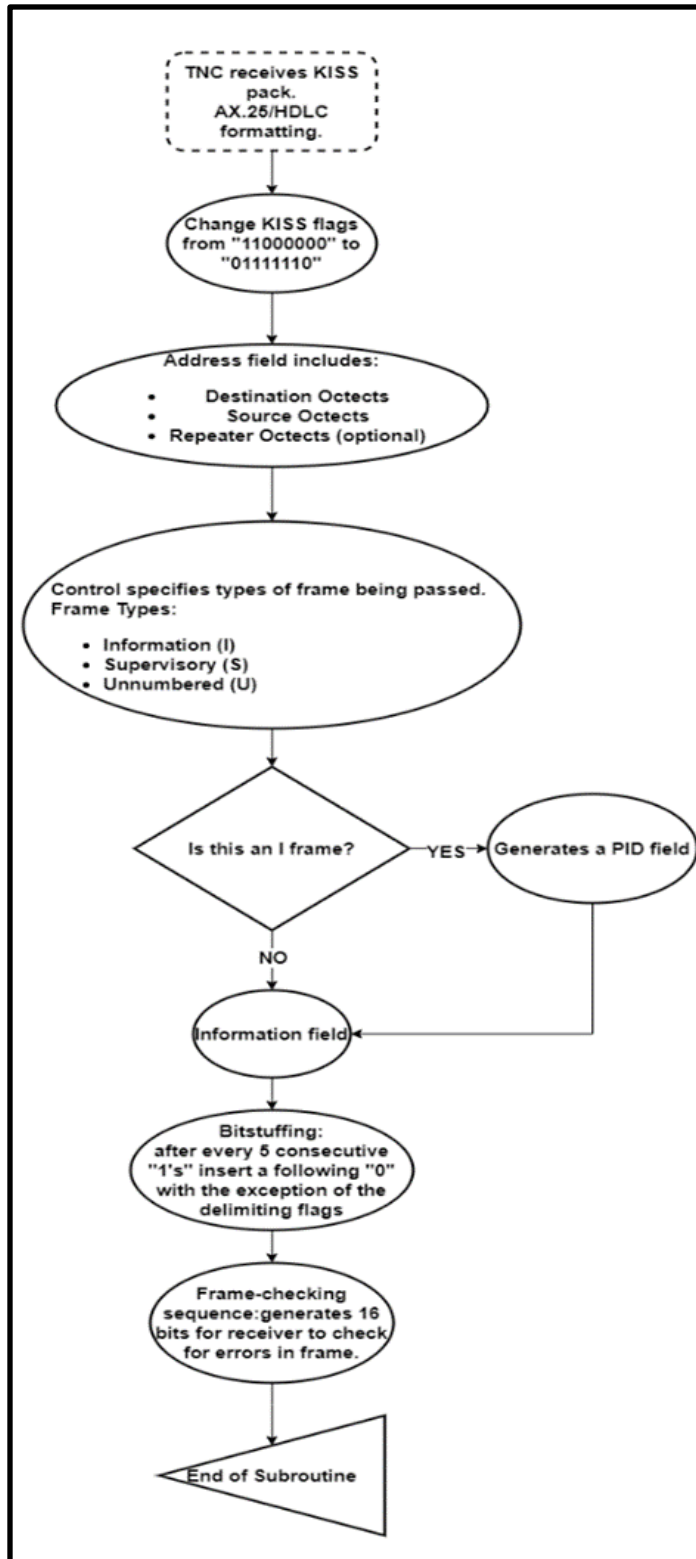


Figure D-3. Flowchart of HDLC Packet Formatting

D. OSI Layered Communications Model

An Open Systems Interconnection model is a conceptual model of our layered communications which characterizes our communication functions and computing systems without regard of the its underlying internal structure and technology. The design of the project follows a series of communication protocols that are all in the data link layer. These protocols are KISS mode and AX.25. These two protocols are related to each other in packet formatting between radio and PC. In the physical layer where external data lines are connected lie a UART/USB interface and 3.5 mm audio jack for our output FM radio signals.

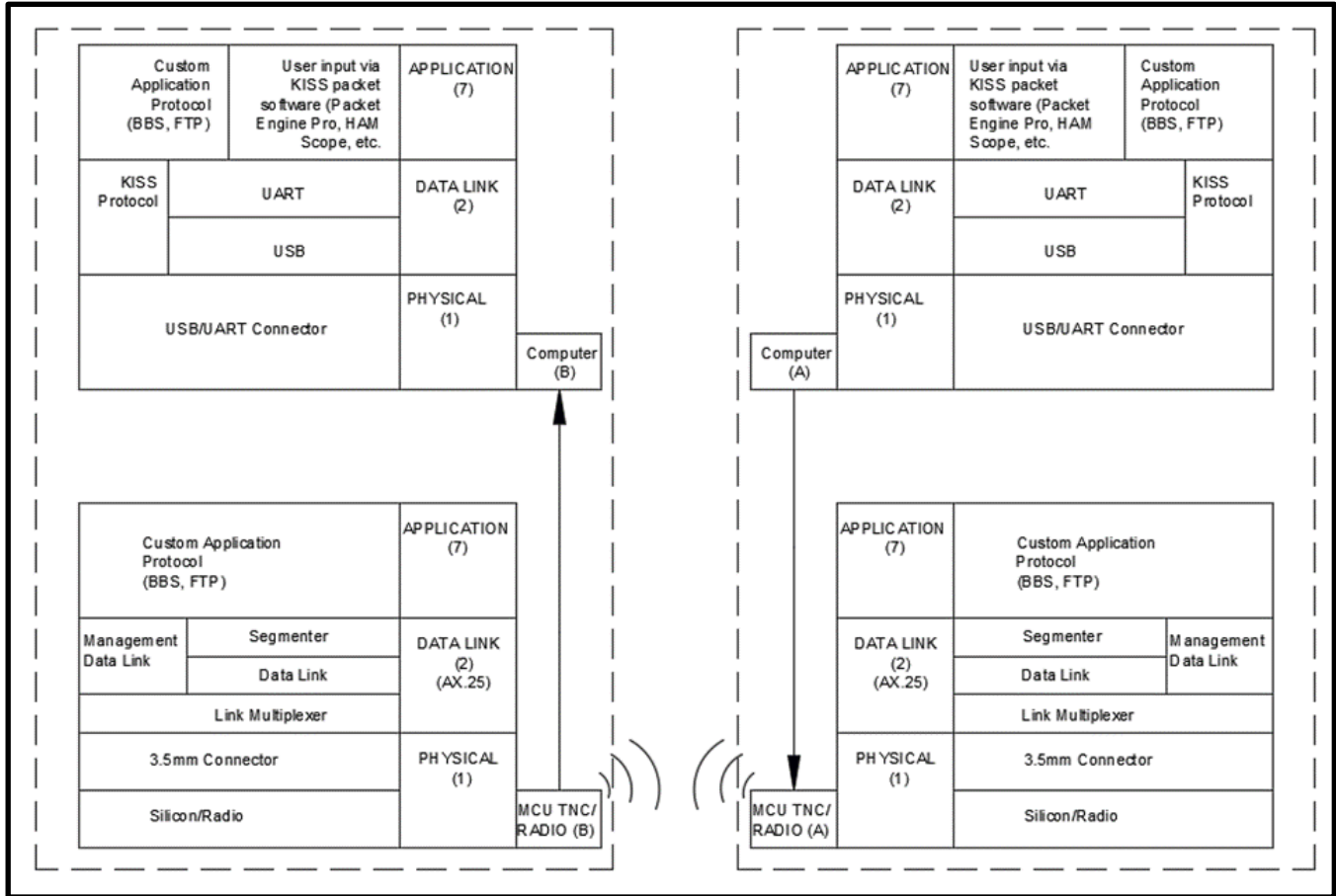


Figure D-4. OSI Layer Communication Model

E. Failure Modes and Effect Analysis (FMEA)

The FMEA 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

F. 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.)

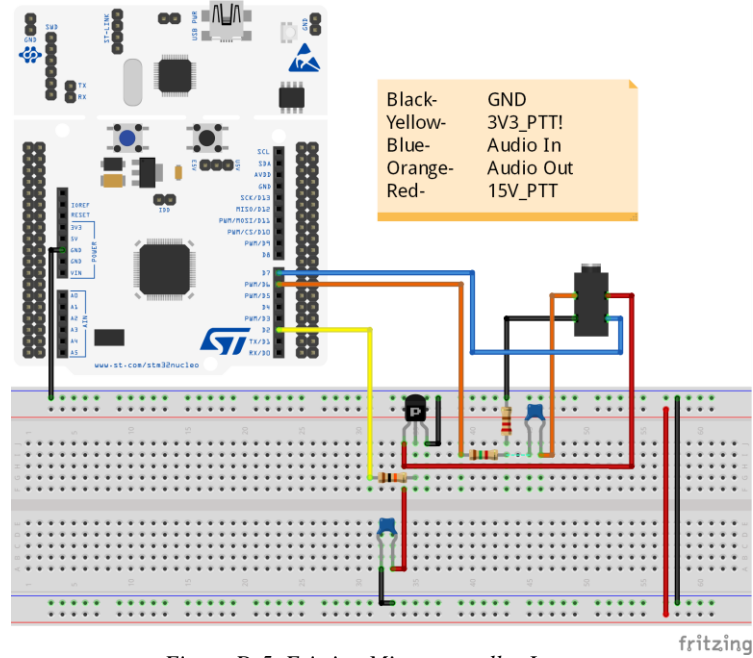


Figure D-5. Fritzing Microcontroller Layout

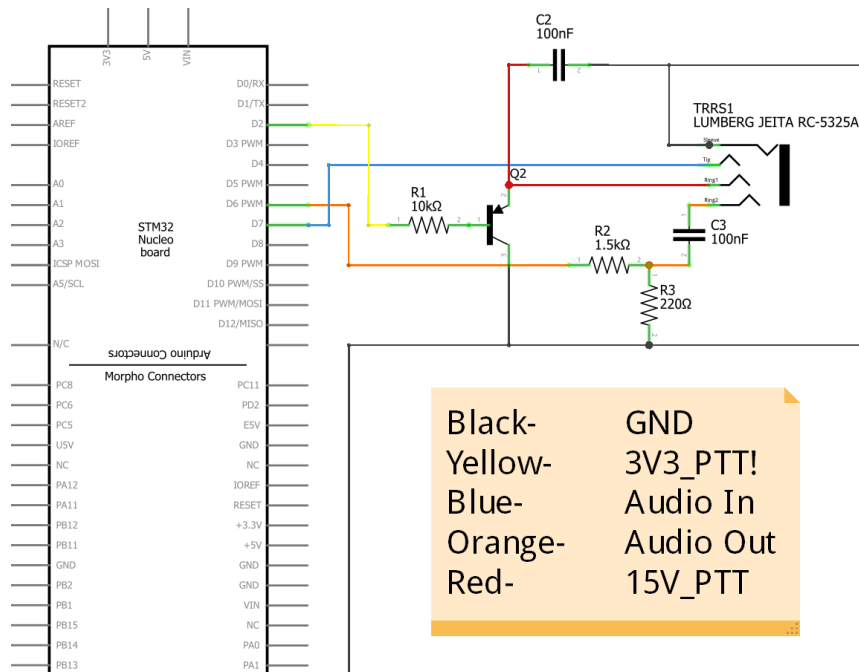


Figure D-6. Fritzing Microcontroller Connections

APPENDIX E

Final Design Details

A. Documentation Numbering Convention and File Structure

This design file system is hosted on a private GitHub repo. This repo holds all files for the project and is backed up to each member's local machines as often as possible. GitHub also supports design iteration and testing with two features. Iteration is well documented and reversible as every time a change to a file occurs, the user must input a post message and Git saves an unedited copy associated with this post. This allows for all changes to be scrolled through and previous iterations can be retrieved. With the use of branches, code testing can be done easily as each member will create their own branch to prototype code for the TNC. This code can easily traverse to other branches using merge commands. The main branch will be only be updated with code that has been thoroughly tested and proven to perform as needed.

The structure for the file system is designed to be organized and easy to find whatever is needed. The root directory will have folder that will list general areas of the project such as Design Deliverables, Presentations, Software, Schematics, or Protocol Documentation. Within these folders, the files are kept by corresponding assignment or purpose in project. The aim is to keep the system from becoming very deep so all files are easy to access in a logical manner.

This Folder: Size: 899.8 MB Allocated: 902.4 MB Percent of Drive: 0 % Files: 1,704 Folders: 604							
Name	Size	Allocated ▼	Files	Folders	% of Parent (...)	Last Modified	
GitHub\TNCMCU\	899.8 MB	902.4 MB	1,704	604	100.0 %	10/18/2020	
.git	412.3 MB	412.4 MB	327	187	45.7 %	10/18/2020	
Software	279.6 MB	281.8 MB	1,239	384	31.2 %	10/18/2020	
Code Playground	251.7 MB	253.7 MB	1,080	358	90.0 %	10/18/2020	
GenerateSine	120.3 MB	120.3 MB	45	18	47.4 %	10/18/2020	
.metadata	36.3 MB	37.4 MB	525	268	14.8 %	10/18/2020	
DAC_SINEWAVE	33.7 MB	34.0 MB	176	23	13.4 %	10/18/2020	
FreqCount_CtrTech1	32.2 MB	32.5 MB	175	22	12.8 %	9/21/2020	
Noise_signal	29.2 MB	29.4 MB	159	22	11.6 %	10/18/2020	
MCUTNC	27.3 MB	27.6 MB	154	22	9.8 %	10/11/2020	
Debug	22.5 MB	22.6 MB	79	6	82.1 %	10/11/2020	
Drivers	4.7 MB	4.8 MB	59	10	17.3 %	8/24/2020	
Core	90.5 KB	112.0 KB	10	3	0.4 %	8/24/2020	
Src	45.4 KB	60.0 KB	6	0	53.6 %	8/24/2020	
system_stm32f4xx.c	25.8 KB	28.0 KB	1	0	46.7 %	8/24/2020	
main.c	5.3 KB	8.0 KB	1	0	13.3 %	8/24/2020	
stm32f4xx_hal_msp.c	4.3 KB	8.0 KB	1	0	13.3 %	8/24/2020	
stm32f4xx_it.c	5.8 KB	8.0 KB	1	0	13.3 %	8/24/2020	
syscalls.c	2.8 KB	4.0 KB	1	0	6.7 %	8/24/2020	
system.c	1.5 KB	4.0 KB	1	0	6.7 %	8/24/2020	
Startup	24.5 KB	28.0 KB	1	0	25.0 %	8/24/2020	
Inc	20.6 KB	24.0 KB	3	0	21.4 %	8/24/2020	
[6 Files]	48.8 KB	64.0 KB	6	0	0.2 %	8/24/2020	
Riz Program	578.3 KB	580.0 KB	2	0	0.2 %	9/22/2020	
AX25 Interface.msi	578.0 KB	580.0 KB	1	0	100.0 %	9/22/2020	
Readme (Written by Kaleb).txt	288 Bytes	0 Bytes	1	0	0.0 %	9/22/2020	
[2 Files]	613 Bytes	0 Bytes	2	0	0.0 %	9/21/2020	
RemoteSystemsTempFiles	289 Bytes	0 Bytes	1	0	0.0 %	9/21/2020	
Design Deliverables (Paper Revisions)	93.1 MB	93.3 MB	87	18	10.3 %	10/18/2020	
Assignment 2A	23.8 MB	23.8 MB	3	0	25.6 %	8/24/2020	
Assignment-2A-Team2-MCU TNC Design.pdf	12.0 MB	12.0 MB	1	0	50.2 %	8/24/2020	
EECE443 Assignment 2A red marks.pdf	11.8 MB	11.8 MB	1	0	49.7 %	8/24/2020	
Assignment-2A-Team2-MCU TNC Design.docx	17.5 KB	20.0 KB	1	0	0.1 %	8/24/2020	
Assignment 3	17.6 MB	17.6 MB	13	0	18.9 %	10/18/2020	
Assignment 3 - Final Presentation - Team 2 ...	7.3 MB	7.3 MB	1	0	41.4 %	9/20/2020	
Final Report-Team2-MCU TNC Design.docx	6.1 MB	6.1 MB	1	0	34.6 %	9/20/2020	
EECE443 Design 1 Final Presentation - Tea...	1.8 MB	1.8 MB	1	0	10.2 %	9/20/2020	
EECE443 Design 1 Final Report - Team 2 M...	1.8 MB	1.8 MB	1	0	10.1 %	8/24/2020	
Schmitt Trigger.docx	494.5 KB	496.0 KB	1	0	2.7 %	8/24/2020	
OSI model.docx	40.0 KB	44.0 KB	1	0	0.2 %	9/20/2020	
Assembly Instructions.docx	29.7 KB	32.0 KB	1	0	0.2 %	9/20/2020	
Wiring Assembly.docx	24.9 KB	28.0 KB	1	0	0.2 %	9/20/2020	
Table of Contents.docx	17.0 KB	20.0 KB	1	0	0.1 %	9/20/2020	
~WRL1534.tmp	14.0 KB	16.0 KB	1	0	0.1 %	9/20/2020	
david_code_appendix.docx	14.0 KB	16.0 KB	1	0	0.1 %	9/20/2020	
david_github_appendix.docx	14.3 KB	16.0 KB	1	0	0.1 %	9/20/2020	
~\$vid_github_appendix.docx	162 Bytes	0 Bytes	1	0	0.0 %	9/20/2020	
Assignment 2C	11.1 MB	11.1 MB	4	0	11.9 %	9/20/2020	
Assignment-2C_Kobe_Edits.docx	4.9 MB	4.9 MB	1	0	44.5 %	8/24/2020	
Assignment-2C-Team2-MCU TNC Design.docx	4.9 MB	4.9 MB	1	0	44.5 %	8/24/2020	
Assignment-2C-Team2-MCU TNC Design.pdf	1.2 MB	1.2 MB	1	0	10.9 %	8/24/2020	
~\$ignment-2C-Team2-MCU TNC Design.docx	162 Bytes	0 Bytes	1	0	0.0 %	9/20/2020	
Assignment 1C	8.1 MB	8.1 MB	2	0	8.7 %	8/24/2020	
Assignment-1C-Team2-MCU TNC Design.docx	4.1 MB	4.1 MB	1	0	50.0 %	8/24/2020	
Assignment-1C-Team2-MCU TNC Design (C...	4.0 MB	4.1 MB	1	0	50.0 %	8/24/2020	

Figure E-1. File Structure

Name	Size	Allocated ▼	Files	Folders	% of Parent (...)	Last Modified
Assignment 2	6.7 MB	6.7 MB	15	1	7.2 %	8/24/2020
[7 Files]	6.2 MB	6.2 MB	7	0	92.5 %	8/24/2020
Preliminary Design	506.2 KB	516.0 KB	8	0	7.5 %	8/24/2020
Deliverable 4	6.3 MB	6.3 MB	1	0	6.8 %	10/18/2020
EECE460 Design 2 Midterm Report Project-T...	6.3 MB	6.3 MB	1	0	100.0 %	10/18/2020
[6 Files]	4.4 MB	4.4 MB	6	0	4.7 %	9/20/2020
EECE-443-Design-I- TNC-MCU-Design-Poste...	4.1 MB	4.1 MB	1	0	93.6 %	8/24/2020
Gantt Layout - Semester 2.jpg	140.1 KB	144.0 KB	1	0	3.2 %	8/24/2020
Gantt Layout - Semester 1.png	117.8 KB	120.0 KB	1	0	2.6 %	8/24/2020
Gantt Layout.gan	16.8 KB	20.0 KB	1	0	0.4 %	8/24/2020
Operational Gantt Chart.gan	7.8 KB	8.0 KB	1	0	0.2 %	9/20/2020
Presentation availability.txt	0 Bytes	0 Bytes	1	0	0.0 %	8/24/2020
Assignment 1	3.7 MB	3.7 MB	1	0	4.0 %	8/24/2020
Assignment-1-Team2-MCU TNC Design.docx	3.7 MB	3.7 MB	1	0	100.0 %	8/24/2020
Assignment 1A	3.2 MB	3.2 MB	3	0	3.4 %	8/24/2020
Assignment-1A-Team2-MCU TNC Design-me...	3.0 MB	3.1 MB	1	0	95.1 %	8/24/2020
Assignment-1A-Team2-MCU TNC Design.pdf	138.9 KB	140.0 KB	1	0	4.3 %	8/24/2020
Assignment-1A-Team2-MCU TNC Design.docx	16.8 KB	20.0 KB	1	0	0.6 %	8/24/2020
Assignment 2B	2.8 MB	2.8 MB	3	0	3.0 %	8/24/2020
EECE443 Assignment 2B Team 2 TNC MCU ...	2.7 MB	2.7 MB	1	0	96.5 %	8/24/2020
EECE443 Assignment 2B Team 2 TNC MCU ...	77.6 KB	80.0 KB	1	0	2.8 %	8/24/2020
EECE443 Assignment 2B Team 2 TNC MCU ...	18.5 KB	20.0 KB	1	0	0.7 %	8/24/2020
Testing Documents	2.7 MB	2.7 MB	11	3	2.9 %	10/18/2020
Complete Tests	2.6 MB	2.7 MB	10	2	99.4 %	10/18/2020
Software	1.3 MB	1.3 MB	7	0	50.1 %	10/18/2020
Error_Checking 2.2.1.1.pdf	661.0 KB	664.0 KB	1	0	48.5 %	10/18/2020
2.3.1.2.3.1.pdf	204.7 KB	208.0 KB	1	0	15.2 %	10/18/2020
2.3.1.2.2.1.docx	164.6 KB	168.0 KB	1	0	12.3 %	10/18/2020
2.3.1.2.2.1.pdf	117.0 KB	120.0 KB	1	0	8.8 %	10/18/2020
Error_Checking 2.2.1.1.docx	107.3 KB	108.0 KB	1	0	7.9 %	10/18/2020
2.3.1.2.3.1.docx	98.3 KB	100.0 KB	1	0	7.3 %	10/18/2020
debug.log	234 Bytes	0 Bytes	1	0	0.0 %	10/18/2020
Hardware	1.3 MB	1.3 MB	2	0	49.9 %	10/18/2020
1.3.1.1.docx	1.0 MB	1.0 MB	1	0	77.1 %	10/18/2020
1.3.1.1.pdf	311.2 KB	312.0 KB	1	0	22.9 %	10/18/2020
[1 Files]	541 Bytes	0 Bytes	1	0	0.0 %	10/18/2020
[1 Files]	13.1 KB	16.0 KB	1	0	0.6 %	10/11/2020
Deliverable 3	1.6 MB	1.6 MB	6	0	1.7 %	10/18/2020
Testing Tree SOFTWARE.png	562.6 KB	564.0 KB	1	0	34.1 %	9/20/2020
Testing Tree HARDWARE.png	463.8 KB	464.0 KB	1	0	28.0 %	9/20/2020
EECE460 Design2 - Team2 - MCU TNC Desi...	333.8 KB	336.0 KB	1	0	20.3 %	9/20/2020
Deliverable 3 - Comprehensive Testing Tree....	213.9 KB	216.0 KB	1	0	13.0 %	9/20/2020
Testing Tree TOP.png	68.5 KB	72.0 KB	1	0	4.3 %	9/20/2020
Testing Tree.drawio	3.6 KB	4.0 KB	1	0	0.2 %	9/20/2020
Diagrams Links	1.0 MB	1.0 MB	15	0	1.1 %	10/11/2020
TNCMCU Level 1 Design Diagram.png	452.3 KB	456.0 KB	1	0	42.9 %	9/20/2020
TNCMCU Flow Chart.png	368.1 KB	372.0 KB	1	0	35.0 %	8/24/2020
TNCMCU Level 2 Diagram.png	98.0 KB	100.0 KB	1	0	9.4 %	9/20/2020
TNCMCU Level 1 Diagram.png	63.7 KB	64.0 KB	1	0	6.0 %	8/24/2020
OSI Model.png	42.5 KB	44.0 KB	1	0	4.1 %	8/24/2020
TNCMCU Level 0 Diagram.png	22.6 KB	24.0 KB	1	0	2.3 %	8/24/2020
BJT Circuit.html	674 Bytes	4.0 KB	1	0	0.4 %	8/24/2020
OSI Model.html	184 Bytes	0 Bytes	1	0	0.0 %	8/24/2020
Testing Tree.html	184 Bytes	0 Bytes	1	0	0.0 %	10/11/2020
TNCMCU Flow Chart.html	184 Bytes	0 Bytes	1	0	0.0 %	8/24/2020
TNCMCU Level 0 Diagram.html	184 Bytes	0 Bytes	1	0	0.0 %	8/24/2020
TNCMCU Level 1 Design Diagram.html	184 Bytes	0 Bytes	1	0	0.0 %	9/20/2020
TNCMCU Level 1 Diagram.html	184 Bytes	0 Bytes	1	0	0.0 %	8/24/2020
TNCMCU Level 2 Diagram.html	184 Bytes	0 Bytes	1	0	0.0 %	9/20/2020
Voltage Divider.html	352 Bytes	0 Bytes	1	0	0.0 %	8/24/2020
BOM	33.3 KB	36.0 KB	3	0	0.0 %	10/11/2020
BOM 3_3_2020.xlsx	11.5 KB	12.0 KB	1	0	33.3 %	8/24/2020
BOM 9_21_2020.xlsx	11.1 KB	12.0 KB	1	0	33.3 %	10/11/2020
Eagle_BOM.xlsx	10.8 KB	12.0 KB	1	0	33.3 %	8/24/2020
Assignment 1B	23.9 KB	24.0 KB	1	0	0.0 %	8/24/2020
Assignment 1B.docx	23.9 KB	24.0 KB	1	0	100.0 %	8/24/2020
Schematic	87.0 MB	87.1 MB	22	8	9.6 %	10/18/2020

Figure E-2. File Structure Continued

Name	Size	Allocated ▼	Files	Folders	% of Parent (...)	Last Modified
↳ Ltpice	83.6 MB	83.6 MB	8	2	96.0 %	10/18/2020
↳ Rx_circuit	83.5 MB	83.6 MB	7	0	100.0 %	10/18/2020
↳ PTT_circuit	1.1 KB	4.0 KB	1	0	0.0 %	10/18/2020
↳ EAGLE	3.0 MB	3.0 MB	10	2	3.4 %	9/20/2020
↳ Libraries	2.8 MB	2.9 MB	5	0	95.8 %	8/24/2020
↳ MCUTNC	124.1 KB	128.0 KB	5	0	4.2 %	9/20/2020
↳ MCUTNC.sch	91.6 KB	92.0 KB	1	0	71.9 %	8/24/2020
↳ TNCMCU.png	15.3 KB	16.0 KB	1	0	12.5 %	8/24/2020
↳ eagle.epf	11.6 KB	12.0 KB	1	0	9.4 %	8/24/2020
↳ AUDIOCircuit.png	2.9 KB	4.0 KB	1	0	3.1 %	9/20/2020
↳ PTT_Circuit.png	2.7 KB	4.0 KB	1	0	3.1 %	9/20/2020
↳ Fritzing Schematics	512.5 KB	524.0 KB	4	1	0.6 %	8/24/2020
↳ Breadboard Schematic	509.9 KB	520.0 KB	3	0	99.2 %	8/24/2020
↳ BreadboardVer_bb.png	384.3 KB	388.0 KB	1	0	74.6 %	8/24/2020
↳ BreadboardVer_schem.png	113.0 KB	116.0 KB	1	0	22.3 %	8/24/2020
↳ BreadboardVer.fzz	12.6 KB	16.0 KB	1	0	3.1 %	8/24/2020
[1 Files]	2.6 KB	4.0 KB	1	0	0.8 %	8/24/2020
↳ Presentations	17.5 MB	17.5 MB	17	0	1.9 %	9/28/2020
↳ 2_21_20 EECE-443-Design-I- TNC-MCU-Desig...	4.1 MB	4.1 MB	1	0	23.7 %	8/24/2020
↳ 3_26_20 Assignment 2A Presentation.pptx	3.9 MB	3.9 MB	1	0	22.3 %	8/24/2020
↳ 2_18_20 Assignment 1 Presentation.pptx	1.4 MB	1.4 MB	1	0	8.2 %	8/24/2020
↳ 3_26_20 Assignment 2A Presentation.pdf	1.2 MB	1.2 MB	1	0	7.1 %	8/24/2020
↳ 2_18_20 Assignment 1 Presentation-trimmed.p...	1.2 MB	1.2 MB	1	0	7.0 %	8/24/2020
↳ 8_18_20.pptx	1.1 MB	1.1 MB	1	0	6.2 %	9/21/2020
↳ EECE460 Design2 - Kaleb Leon C00094357 - ...	831.1 KB	832.0 KB	1	0	4.6 %	9/21/2020
↳ 3_20_20 Assignment 2 Presentation.pptx	754.7 KB	756.0 KB	1	0	4.2 %	8/24/2020
↳ 4_2_20 Assignment 2C Presentation.pptx	662.7 KB	664.0 KB	1	0	3.7 %	8/24/2020
↳ 3_20_20 Assignment 2 Presentation.pdf	633.1 KB	636.0 KB	1	0	3.5 %	8/24/2020
↳ 4_2_20 Assignment 2C Presentation.pdf	566.5 KB	568.0 KB	1	0	3.2 %	8/24/2020
↳ 3_10_20 Assignment 1B Presentation.pptx	428.4 KB	432.0 KB	1	0	2.4 %	8/24/2020
↳ 4_1_20 Assignment 2B Presentation.pdf	256.8 KB	260.0 KB	1	0	1.4 %	8/24/2020
↳ 4_1_20 Assignment 2B Presentation.pptx	171.8 KB	172.0 KB	1	0	1.0 %	8/24/2020
↳ 2_12_20 Group 2 _TNC MCU Project .pptx	127.4 KB	128.0 KB	1	0	0.7 %	8/24/2020
↳ 3_10_20 Assignment 1C Presentation.pptx	78.3 KB	80.0 KB	1	0	0.4 %	8/24/2020
↳ 3_3_20 Assignment 1A Presentation.pptx	38.7 KB	40.0 KB	1	0	0.2 %	8/24/2020
↳ Protocol Documentation	7.3 MB	7.3 MB	7	0	0.8 %	10/18/2020
↳ APRS101.PDF	3.0 MB	3.0 MB	1	0	41.5 %	8/24/2020
↳ destevéz_net.pdf	1.3 MB	1.3 MB	1	0	18.0 %	8/24/2020
↳ Ahmad_2016_IOP_Conf_Ser_Mater_Sci_E...	987.8 KB	988.0 KB	1	0	13.2 %	8/24/2020
↳ AX25.2.2.pdf	928.1 KB	932.0 KB	1	0	12.5 %	8/24/2020
↳ DAC generation document.pdf	816.2 KB	820.0 KB	1	0	11.0 %	9/21/2020
↳ AX25 Amateur Packet-Radio Link-Layer Protoc...	287.8 KB	288.0 KB	1	0	3.9 %	8/24/2020
↳ debug.log	702 Bytes	0 Bytes	1	0	0.0 %	10/18/2020
↳ Hardware Specs	3.0 MB	3.0 MB	3	0	0.3 %	10/11/2020
↳ STM32L433CC Datasheet.pdf	3.0 MB	3.0 MB	1	0	97.9 %	8/24/2020
↳ graph.pdf	60.5 KB	64.0 KB	1	0	2.1 %	8/24/2020
↳ STM32-L476RG-Pinout.html	149 Bytes	0 Bytes	1	0	0.0 %	8/24/2020
[2 Files]	3.2 KB	4.0 KB	2	0	0.0 %	8/24/2020

Figure E-3. File Structure Continued

B. Level 1 Design Diagram

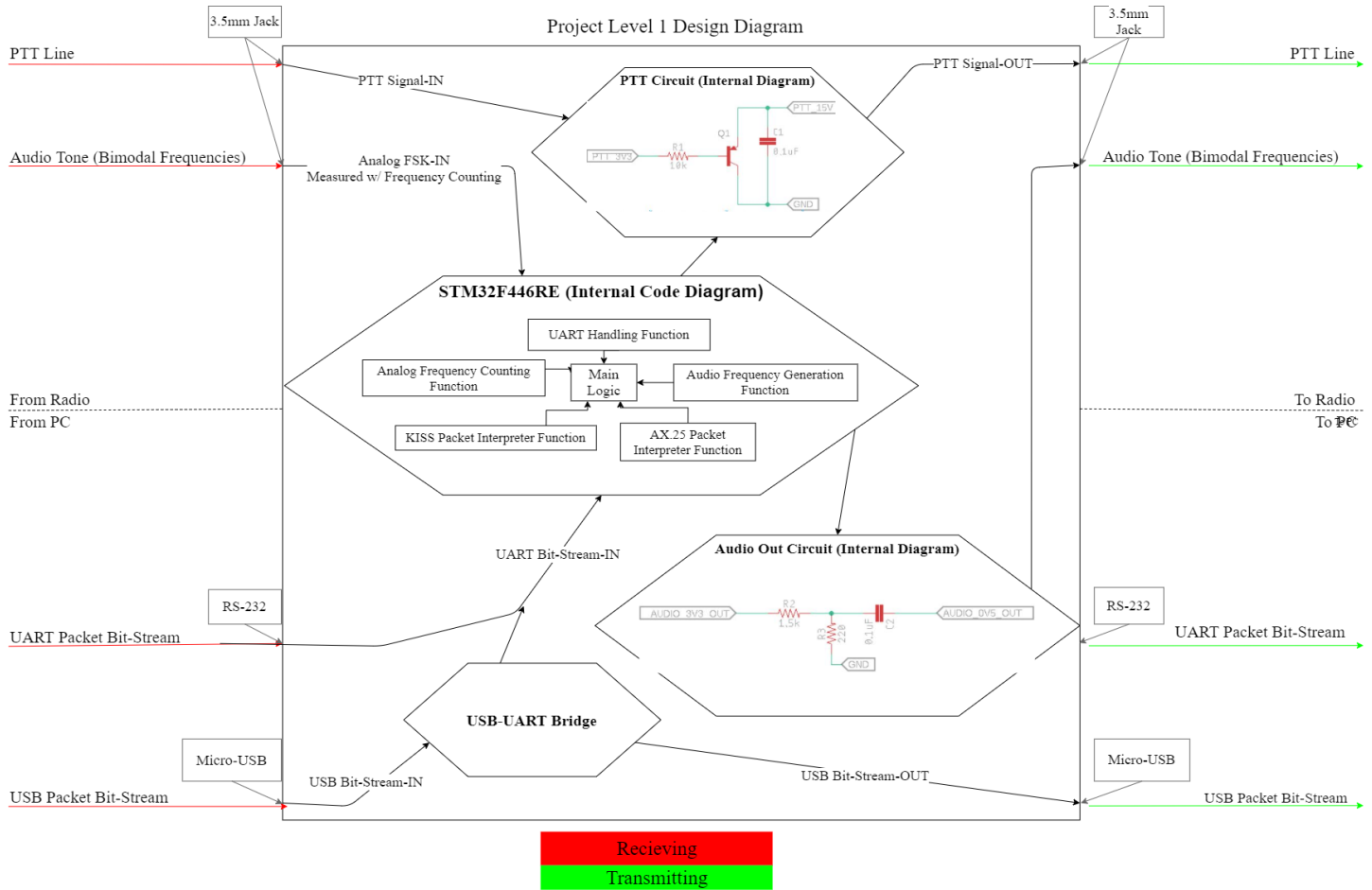


Figure E-4. Level 1 Design Diagram

In Figure E-1, is a Level 1 Design Diagram which is an updated and more detailed version of the Level 1 functional block diagram. This diagram illustrates all the hardware functionality as well as the software functionality. It shows what function code blocks are used to produce the output signals of our system. The power requirements to power this system are done in tandem with the data input at the USB port. Red arrows represent the path taken when receiving a signal from the radio or PC and the inner components show the path the data takes to for the output at the transmission side. The RS232 ports provide a serial connect to the system providing a UART communication line. The 3.5 mm jack provides as the main communication line between the TNC and radio.

C. Code Breakdown

1) Kiss Packet Interpretation

When a KISS packet is received from the PC, the TNC will remove the flags and extract the data section. This data section will then be prepped for formatting into accordance with AX.25 protocol. It will do this by looking for the start and stop flags of a KISS packet and removing those to extract what is left. In addition, when a packet is received in AX.25 protocol from the ADC, the packet will remove the bit stuffing extra zeros and begin extracting the data it needs. It will do this by parsing the messaging detecting what bits pertain to bit stuffing. It will then remove the bits as it goes and produce a data section to be added flags to and sent off to the PC.

VALIDATION/TESTING:

To validate this process, we created our own packets of which we know what the data section says (example in HEX “Hello world”). This HEX packet is sent to the TNC over the USB. It is then picked apart for its data section. This data section is then translated back into HEX then Ascii and outputted to serial. We monitor this serial to see if it is outputting the correct data.

2) Analog Audio Tone to Digital Data Conversion

The Schmitt trigger converts analog signals to digital signals by setting two different threshold voltages, an upper and lower threshold. When the analog signal is inputted, the trigger will output a high voltage, once it reads the upper threshold on input signal, and output a low voltage when it reads the lower threshold. This solution minimizes the amount of noise in the digital signal. The STM32 platform has its GPIOs equipped with Schmitt trigger inputs, with built-in upper and lower thresholds. When the GPIO reads an input above 70% of the power rail (3.3V) then that pin reads “high,” which is the upper threshold. As for the lower threshold, the GPIO will read a “low” when a voltage lower than 30% of the power rail is inputted.

We are using the Schmitt trigger in our design to read frequencies of incoming signals from the radio. Then, we use the STM32’s internal timers to count up to a set period, which is how long the input signal will be sampled and how long it would take to transfer a bit. The set period will be dependent on the baud rate (1200 bps): set period = $1/(\text{baud rate})$. We then create a variable to keep count of how many times the signal toggles from the upper to lower threshold or vice versa.

Once the timer counts up to the set period, the frequency is calculated:

$$\text{Frequency} = (\text{toggle count} * \text{Clock frequency}) / (\text{Clock Prescaler} * \text{set period}).$$

If the calculated frequency is 2200 Hz it will generate a one and if 1200 Hz a zero will be generated. The ones and zeros can then be stored in an array, with the first element being the MSB. Then the microcontroller will use the CRC calculation unit to generate a CRC value to compare with the value in the FCS section in the packet. If the CRC value matches the FCS value,

then the packet is valid and will go on with the transmission process. If it is not a valid packet, then it will not transmit the packet.

Pseudo code:

```
Initialize toggle_count = 0, frequency, set_period;
Timer.Instance->CNT = 0; //set timer to 0
while((Timer.Instance->CNT) < set_period)
{
    while ( GPIO reads low); //wait for GPIO to read high
    while (GPIO reads high); //wait for GPIO to read low
    toggle_count++;
}
Frequency = (toggle_count / (set_period));
```

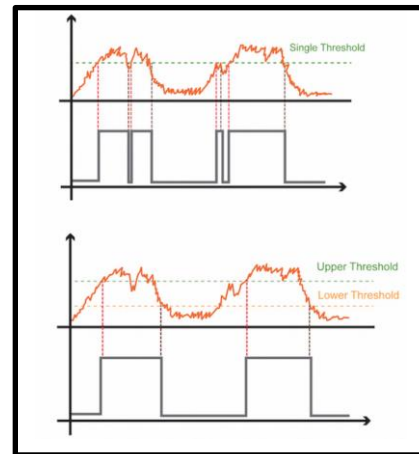


Figure E-5 Threshold Trigger Example

3) Digital Data to FSK Audio Tone Conversion

To generate audio signals for a radio to transmit, the digital to analog converter built into the F446RE will be used. This onboard DAC has a 12-bit resolution and a peak output voltage of 3.3V. This means that if the DAC is given an input value of 4095, it will output peak voltage, with a linear input-output relation.

This DAC operates off an internal clock that is pre-scaled down from 90 MHz to 1MHz while also set to count up to 10. This set the DAC input clock to pulse at a frequency of 100kHz. Next, two arrays are created to store the values of a sine wave, the size of the array will vary to achieve two distinct period lengths. The values are created using the sin function from the C math library. This sine function amplitude will be varied to achieve the necessary output for a radio. Needed Sample Count = $(\text{Input Clock Frequency}) / (\text{Output Frequency})$, producing 84 samples for the lower frequency and 46 for the higher frequency.

VALIDATION/TESTING:

A short binary bitstream is hardcoded to be sent to the DAC. The microcontroller saves this bit stream to a memory location that is reference and then parsed through as it is sent to the DAC. The DAC creates an audio signal that is sent over wire back into the TNC where it is demodulated at the ADC code block. If the data returns the same as what was sent out the process is valid.

D. Bill of Materials

Bill of Materials					
	Quantity	Package	Mfr. Part #/Vendo Link	Mouser Part #	Price
PNP BJT	1	Through-Hole	ZTX951	522-ZTX951	\$7.80
STM32F446RE Nucleo board	1	uController	NUCLEO-F446RE	511-NUCLEO-F446RE	\$42.00
3.5mm Audio Jack	1	Connector	1699	485-1699	\$0.95
10k Resistor	1	Through-Hole	MFR5-10KFI	756-MFR5-10KFI	\$0.79
.1uF Capacitor	2	Through-Hole	RDE5C1H104J2K1H03B	81-RDE5C1H104J2K1H3B	\$0.25
1.5k Resistor	1	Through-Hole	MFR4-1K5FI	756-MFR4-1K5FI	\$0.79
220 Resistor	1	Through-Hole	MFP1-220RJI	756-MFP1-220RJI	\$0.79
Total Cost					\$53.37

E. Assembly Specifications

This section shows how to assemble the hardware of this system assuming the system has not been received in custom PCB form and wish to build your own using a Microcontroller like the Nucleo Board. It will also include where to download the main software for the TNC hardware to run.

Wiring Schematic:

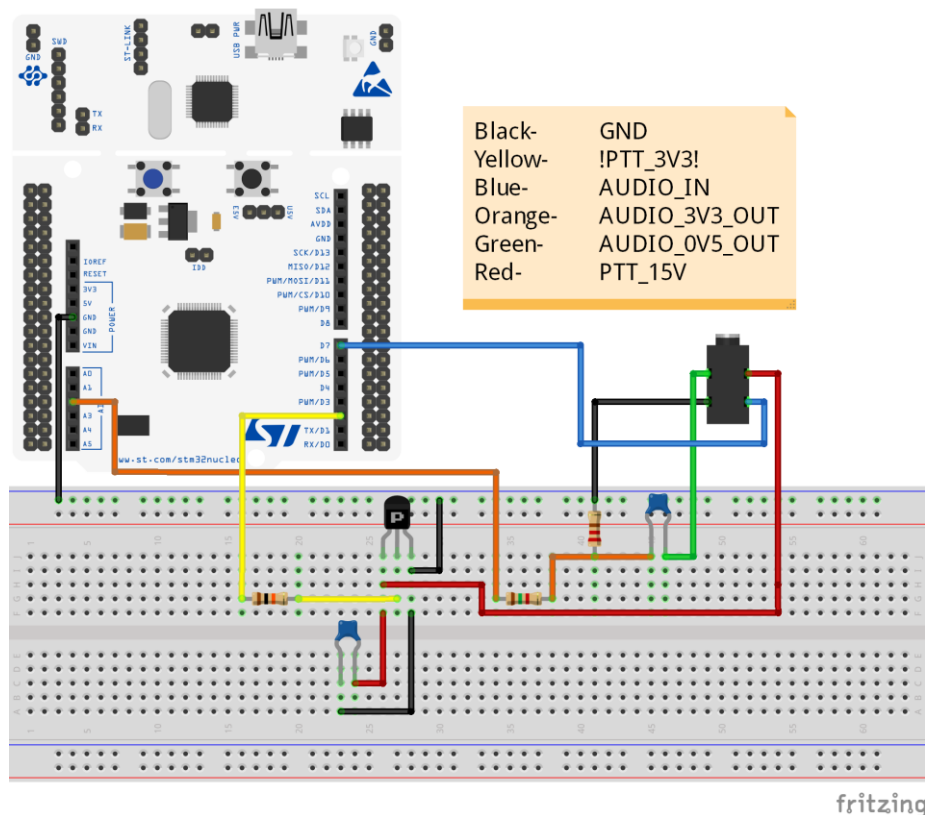


Figure E-5. Breadboard Wiring Schematic

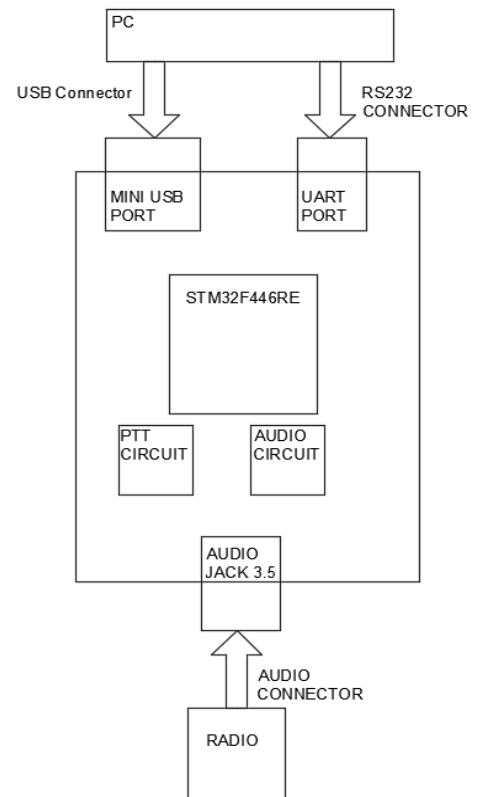


Figure E-6. Simple External Connections

Assembly Instructions:

1. Once you obtain your STM32 Nucleo board and install STM32CubeIde on your PC, gather all components, wires, and breadboard as shown in Figure D-1.
2. Connect all wires and components as shown in Figure D-1.
3. Connect mini USB connector from PC to STM32 board. If you plan on communicating with the microcontroller through UART, then to connect the RS232 connector instead of the USB.
4. Connect radio to the audio jack shown in Figure D-1, using a 3.5 mm audio cable.
5. Open STM32CubeIDE project file on GitHub repository:
<https://github.com/TNCMCU/tree/master/Software/MCUTNC>. This should open the STM32CubeIDE application on PC.

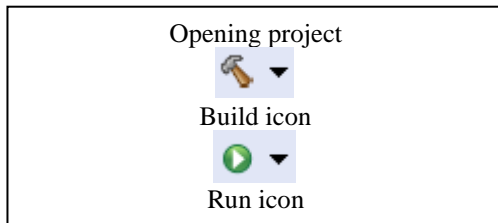


Figure E-7. Simple Icon Identifiers

6. On the left side of the IDE is where you can open the project. Open the main.c file, under MCUTNC > Core > Src > main.c, as shown in Figure E-#
7. Build and run code to STM32 by click the icons shown in Figures E-# and E-#
8. Insert message to be sent to other TNCs, with specific address and number of digipeaters.

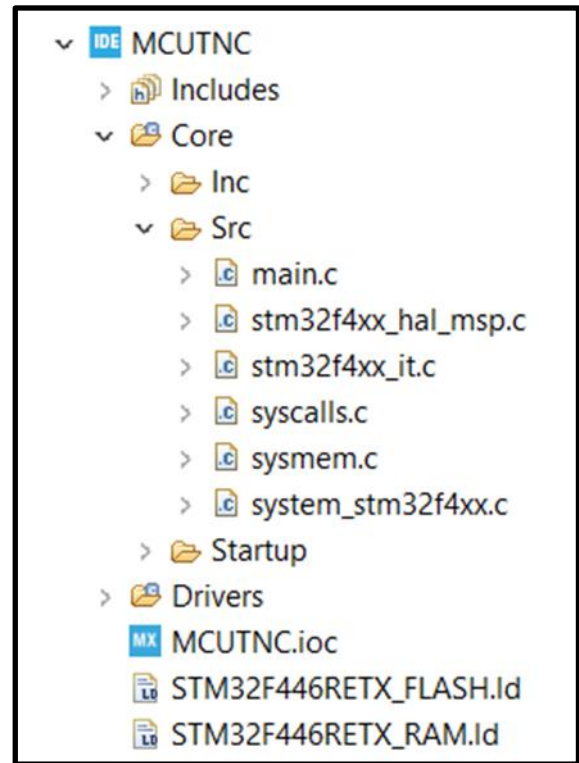


Figure E-8. Folder Navigation

F. Operational Gantt Chart

The Gantt chart shown below covers the research, planning, developing, and testing plan that is followed throughout the development of this project. Though it does not break down in complete detail what is going on it does cover the time span for each task over the course of the first half of the project. The Gantt chart also specifies our human resources divided among the three teammates on tasks based on color. This is an updated and more concise version of our Gantt chart for time management of the project.

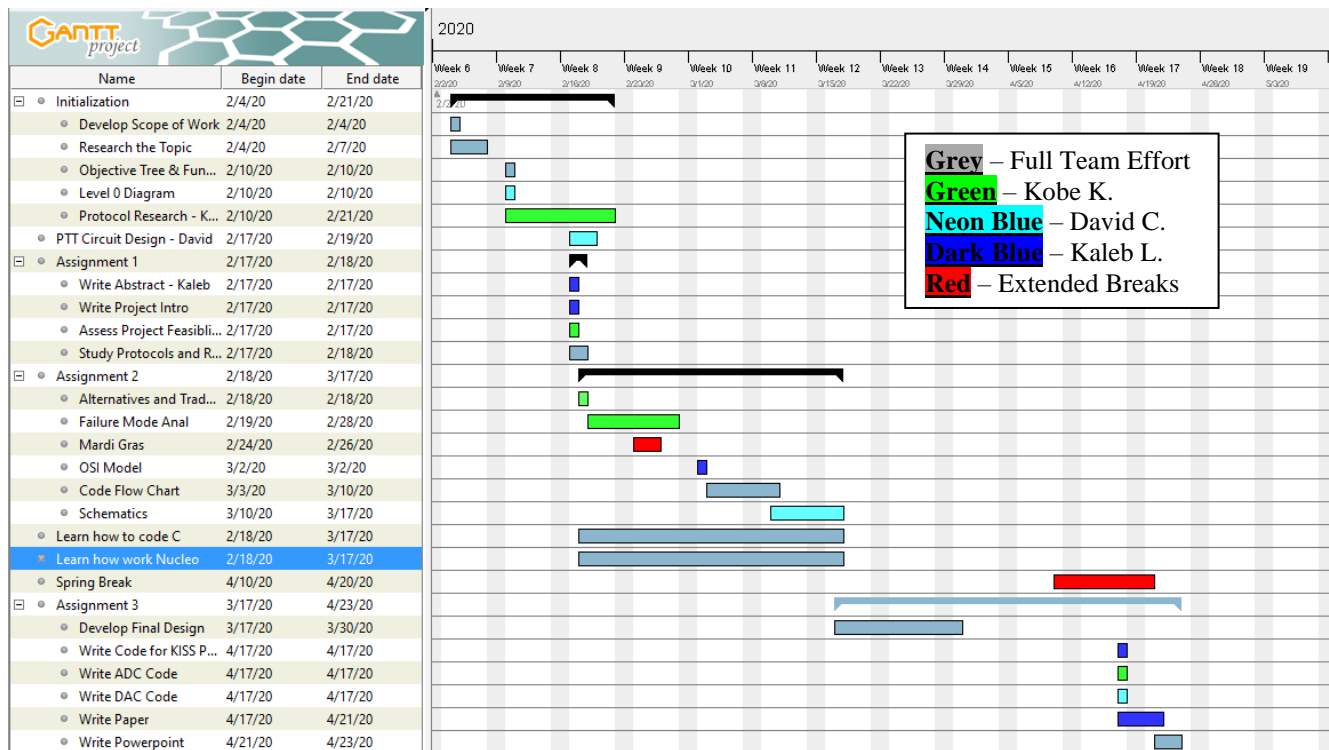


Figure E-9. Operational Gantt Chart

APPENDIX F

A. *Testing Process*

We designed our testing tree by back tracking our way through our current development process to decide which subsystems and components we need to be testing. We plan to have our Preliminary Comprehensive Modular Build and Testing Plan (PCMBTP) thoroughly assess our design's functionality. The design is statistically feasible as stated in previous appendices, so now we must test that physical and digital feasibility. This is done surgically by looking at every component and testing them then moving up to the subsystem made of these components and testing them together. Then, we finally reach the point where all the subsystem come together to test the main system as a whole. If all subsystems and components work separately and then as a subassembly to combine to a full system, the design is then proven to pass, and work as expected.

The first subassembly that needs to be tested are the components of the physical hardware system and circuits. This includes the micro controller itself and the two external circuits. According to our FMEA, if our micro controller does not function then the code has nothing to run on making the project fail before it has even begun. In terms of the micro controller we will need to test anything, and everything being used to accomplish our design. The main portions we use in terms of signal transmission and reception are mostly the cable connections between the radio to TNC and PC to TNC. These transmission lines include: the USB cable which handles serial communication between TNC to PC, the 2.5 mm Audio jack that handles analog signals between the radio to TNC and vice versa, and RS-232 connector that is a backup digital communication line. Testing the effects of the micro controller signal processing is also very important due to the design needing to meet a certain amount of specifications. We have to test to make sure all of our power consumption, latency, and voltages are under spec for them to work with our design and have it function properly. In addition to that, we need the I/O pins on the micro controller to be fully functioning so that they can communicate with the external circuits. These external circuits are also a main portion of the hardware that we need to test to assure our input signals are how we need to process them and to change modes. The Audio input circuit needs to be tested down to the component level of the amplifier and the filter to assure we are getting the correct audio signal in and out of our design. The Push To Talk circuit is used mainly to switch modes so that our TNC knows when we are transmitting or receiving. This also needs to be tested down to the component level so that we know if will function as needed to allow our design to perform its tasks.

The second subassembly that needs to be tested are the components of the digital software that controls the hardware and data processing. This subassembly is broken down into three major subsystems which all need to be tested down to the component level. They are as follows: Kiss Packet Handling, AX.25 Protocol Formatting, and Frequency Shifted Audio Tone

Handling. The Kiss Packet Handling controls how we handle inputs and outputs at the Data Link layer. This is critical for PC to TNC communication. We must take into account and test multiple components of this subsystem. The serial communication line needs to be tested and functional to make sure we can receive and transmit data across the serial bus between PC and TNC. This component needs to also be assessed on the latency of that communication line to meet our specs. In addition, the packet construction following the KISS protocol is a very important process to KISS transmission to PC. Lastly, we need to test the data extraction from this packet to make sure we are able to extract the correct data. Similarly, for the next subsystem of AX.25 Handling we also need to check whether we are extracting the correct data and formatting correctly following the protocol or when we send it off to the DAC the signal will be incorrect when being received by other radios. Lastly, that is where the last subsystem comes into testing. The Frequency Shifted Audio Tone Handling is a critical and complex section of our software, so it needs to be extensively tested. It needs to be able to handle ADC and DAC control which each entail of their own components. If these signals come in incorrectly or out incorrectly then our design is failed and not useful as a product.

It is through this methodology that our team has designed a testing tree with the described components, subsystems, and subassemblies for testing. As testing progresses, we will add more components we overlooked such that we make sure everything is functional in our design. When navigating the tree we will also write up test reports so that all our tests are well documented and noted fixes to our faults. This will ensure confidence in our design.

B. Testing Tree

As shown in our testing tree below, our design system is split into two Subsystems: software and hardware. The Software Subsystem is more complex than the Hardware Subsystem, since most of our project is mainly software based and the hardware's purpose is only to support receiving and transmitting signals from the TNC. In the Software Subassembly, it is broken into three branches that have many supporting leaves that represent different software processes as opposed to the Hardware Subassembly leaves that represent physical component testing. For each subassembly are different tests that we plan to run for each process and physical components. For example, under the hardware subsystem, under each circuit subassembly we will test and ensure each component's nominal value and desired signal output. Components such as resistors will have to be measured to ensure our circuits' outputs. The signal outputted from the physical amplifier circuit will have to be compared to the signal output from spice software. As for the software subsystem, we will validate that each process or subassembly outputs the correct bitstream and frequencies. The bitstreams outputted from the microcontroller have to be outputted in specific sequence. The microcontroller also must output specific frequencies and must be measured to ensure that there are not many errors from the output. Latency will also be tested for serial communication to ensure optimal performance.

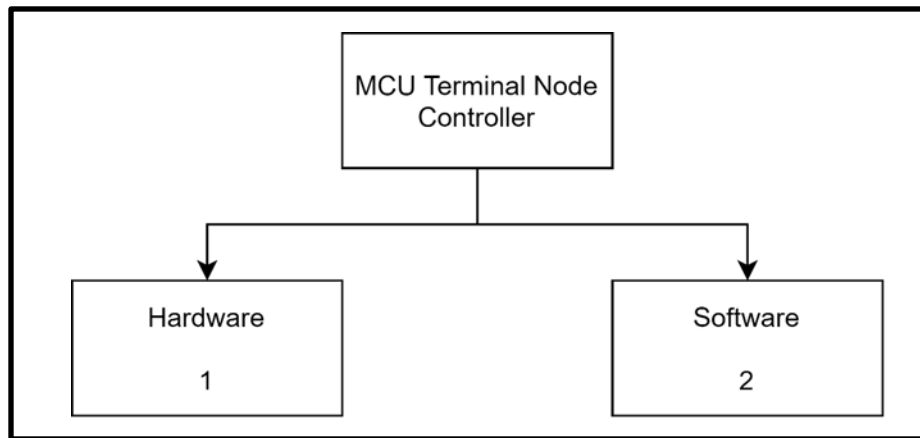


Figure F-1. High Level Comprehensive Testing Tree

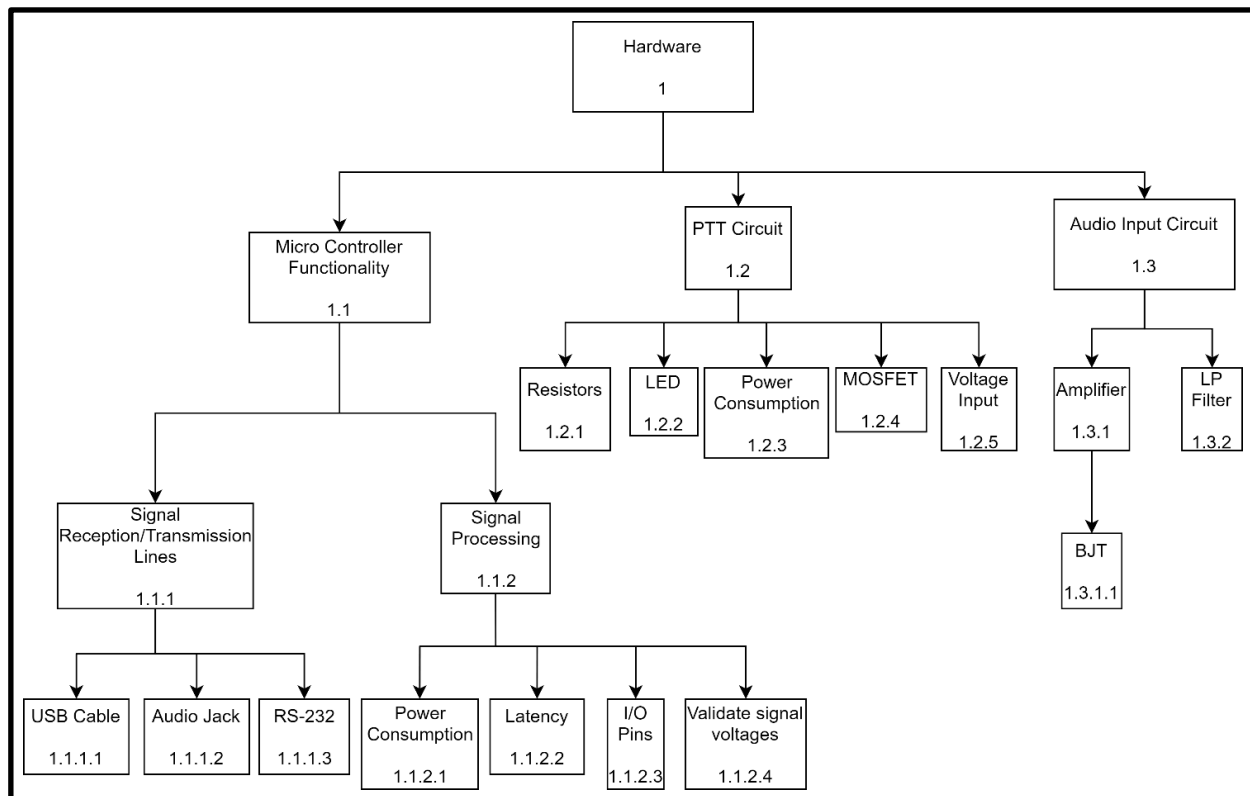


Figure F-2. Hardware Comprehensive Testing Tree

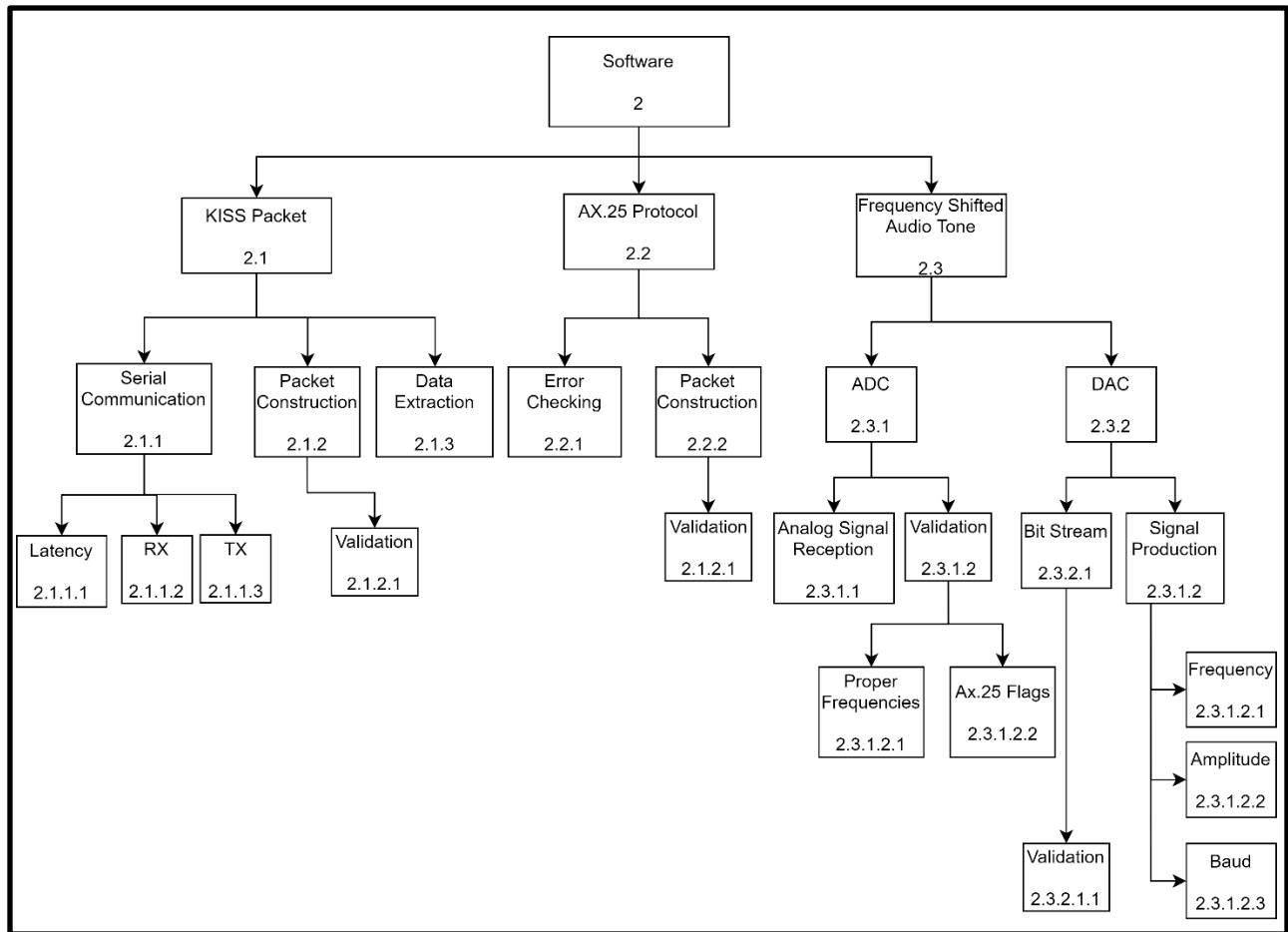


Figure F-3. Software Comprehensive Testing Tree

C. Test Report Template

TNC TESTING FORM _(REV1)	
Leaf on the Tree:	
Device Under Test (Testing Tree Number):	
Date:	
Person(s) Conducting Experiment:	
Signature:	
Experiment Purpose:	
Experiment Procedure:	
Equipment Settings/Software Settings (w Revision):	
Testing Diagram/Picture:	
Data Points:	
Pass/Fail:	
Interpreted Notes:	
Recommendations for Modification:	

D. *Validate and Verification Plan*

1) *Hardware*

The hardware for this project is split into a couple major components such as: Micro Controller, Circuits, and Connections. The microcontroller is verified as working by testing to see if it will power on and test each pin to make sure it is producing the correct output. The pin testing is done using a known to be working Analog Discovery. Next, the circuits are the PTT circuit and the amplifier circuit. They are tested using the Analog Discovery and simulation. The circuits were each designed and simulated before real life construction to be sure the theory is plausible. The constructed real-life circuit is then tested at each node to make sure we are getting proper input and output voltages/currents. We also test the individual components that make up each circuit by measuring their values and tolerances. Lastly, we can test the interconnections between the micro controller and the circuits to make sure they work in tandem. This is done by analyzing the outputs from the micro controller from the pins connected to the circuit and setting up simple code to view these values in serial to make sure they are accurate. In addition, we test the cable connections between the radio and TNC and PC and TNC. We do this by sending hardcoded packets to make sure they are properly inputted and output through serial and in analog.

2) *Software*

The software for this project is also split into many major components dedicated to different functions and protocols of our system such as: Kiss Packet Interpretation, Ax.25 Protocol Formatting, DAC, and ADC. The breakdown of how these sections function and work with one another are described in Appendix E Code Breakdown. To validate and test each of these sub processes, debug statements are spread thoroughly throughout the code and comments left to what each function does. At each step in the process we analyze a hardcoded input and validate the produced output. Each stage of the software work in tandem with each other so once each section is able to produce our desired output they are tested together in sections (two at a time). Once all would work in groups with each other and outputs measured correctly whether through serial output or analog measurement then we proceed to testing all together and send a packet through its full course. We send a hardcoded bitstream to create a FSK sine wave and see if we get the correct output KISS packet in serial on the other side. In reverse, we send a KISS

packet generated by our Mentor's software and see if it generates an accurate FSK sine wave up to spec and containing the correct data.

E. *Workmanship on Design*

Plan before going into experiment:

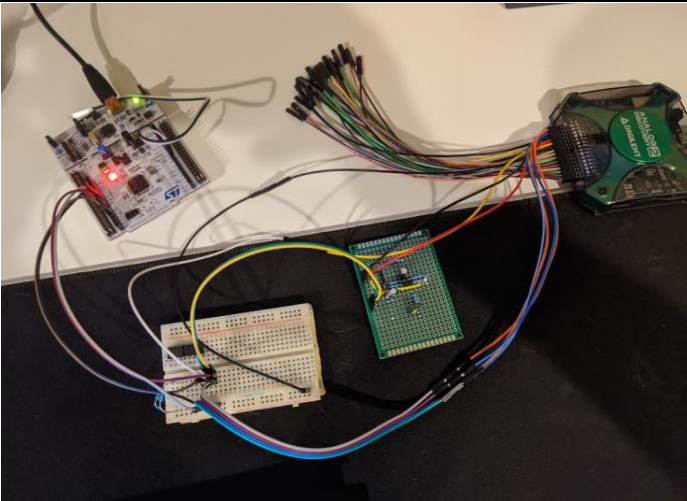
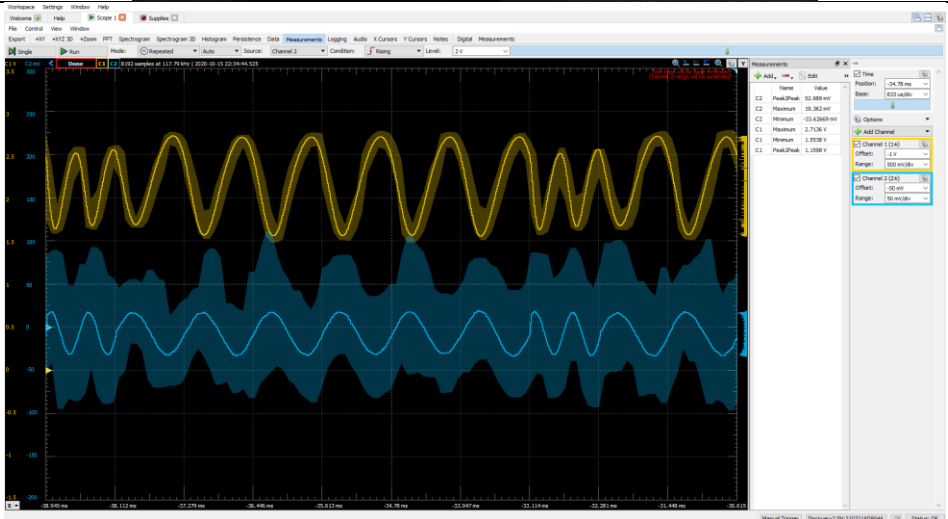
1. What is needed to be done
2. The code or circuit being worked on
3. The code functionality, problem, circuit, and output needed to be tested or created
4. The testing procedure and data collection method
5. Expected results
6. If results not met in one sitting setup a time to revisit

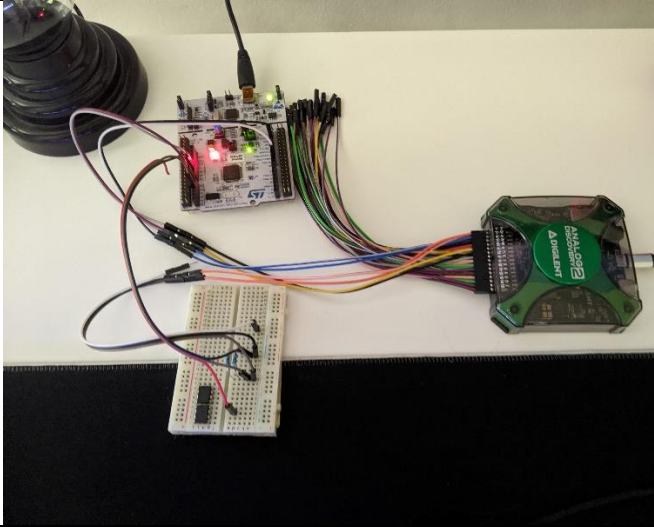
Per experiment we would perform the steps above. Each of us would take action in handling different portions of the design but we follow this same path no matter what task. We code and test things as we go to make sure we are producing accurate results. It is similar to how an AGILE workflow works. We work and test at the same time to make sure we are accurate with each step of the process. This style of work keeps the project going smoothly and meeting small milestones with each experiment completed. Once a main section of the design is accomplished, it is then documented, and all our experiment notes come together to form the paper.

F. *Final Presentation Demo*

The final presentation demo will consist of two TNC systems communicating with one another. The first system will consist of our TNC, a radio, and one of our laptops. The second system will consist of an off the shelf TNC used in CAPE currently as the satellite TNC, a similar radio, and another PC for sending KISS packets. Each PC/radio combo will have their own callsign as well. These two systems should be able to send packets between one another. For example, our laptop will use our mentor's software to generate a KISS packet and convert it to AX.25 and then to an analog signal. This packet will then be sent over the air for the other radio to receive and break down for the other PC to read. In addition, we will show it in the opposite direction as well. The CAPE TNC system will send a signal over the air and we will receive it and break it down for all PC. This will show off our packet formatting system and our ability to communicate with an already manufactured and in use TNC setup.

G. Completed Test Reports

TNC Testing Form (REV1)	
Leaf on the Tree	Amplifier
Device Under Test (Testing Tree Number):	1.3.1.1
Date:	10/15/2020
Person(s) Conducting Experiment:	David Cain, Kobe Keopraseuth
Signature:	
Experiment Purpose:	The purpose of this experiment is to ensure the amplifier output can trigger the external interrupt on the STM32 for reading incoming AFSK waveform frequency components.
Experiment Procedure:	Using waveform generator from Analog Discovery 2, input a 50mV ptp AFSK waveform to amplifier circuit, checking readings reported by microcontroller on serial port.
Equipment Settings / Software Settings (w Revision):	Micro controller is set to receiving mode Using WaveForms software(version 3.12.2), output generated AFSK signal
Testing Diagram / Picture:	
Data Points:	

TNC Testing Form (REV1)	
Leaf on the Tree	Amplitude
Device Under Test (Testing Tree Number):	2.3.1.2.2.1
Date:	10/4/2020
Person(s) Conducting Experiment:	David Cain
Signature:	
Experiment Purpose:	The purpose of this experiment is to measure waveform output voltage. Part of our specifications it to be capable of sinking 400mV(ptp) into 1k.
Experiment Procedure:	To verify amplitude, the analog output will be connected to a 1k load and measured. In this case, 2 x 2k resistors will be wired in parallel on a bread board, creating 1k of resistance across the terminals.
Equipment Settings / Software Settings (w Revision):	The Digilent will be set to record the maximum value of the waveform measured. For general insight, the RMS and minimum were also recorded
Testing Diagram / Picture:	
Data Points:	Maximum: 399.19 mV RMS: 137.11 mV Minimum: -1.43 mV
Pass / Fail:	Pass
Interpreted Notes:	The waveform satisfies the 400mV requirement. Potentially some feedback could be used to tune the output during runtime, but this is not necessarily required.
Recommendations for Modifications:	None, currently

[illegible]

APPENDIX G

A. Change Order Form Template

Change Order Form (COF) Rev 1	
Date	
Subsystem	
Component	
Change	
Reasoning	
Rationale	
Mod. Details	

B. Completed Change Order Forms

Change Order Form (COF) Rev 1	
Date	10/17/2020
Subsystem	1.3 Audio Input Circuit (Audio Input Circuit)
Component	1.3.1 Receiving Circuit (Amplifier)
Change	Need to change to resistor values and need to supply a lower DC voltage to the amplifier.
Reasoning	Before the amplifier was not connected to a low pass filter, but because of adding the low pass filter the amplifier's gain has decrease. The gain needs to be high enough for signals, coming from the radio, to be read by the STM32 microcontroller. The microcontroller reads in 70 % of 3.3 V as a high input and 30 % of 3.3 V as a low input. Unfortunately, after testing the signal, outputted from the amplifier, does not reach those amplitudes.
Rationale	A way to fix this issue is to perform a small signal analysis on the current circuit and solve for the necessary resistor values and DC supply voltage, knowing the desired gain and DC offset to achieve.
Mod. Details	Although we yet to test the circuit in the lab, after solving for the necessary resistors and correct DC supply voltage, the circuit was able to output a sine wave that reaches the necessary voltages through simulation, using LTspice.

APPENDIX I

A. Time Sheets

1) David Cain

Item	Date/Time	Description	Hours				
1	2/5/2020 5:30pm-7pm	Initial mentor meeting. Discussed plans for the project and project member were given a breakdown of what was expected of them.	1.5	1	4/18/2020 6:16pm-12:44am	Spent a while getting the DAC code written. Was struggling with pointer arrays within C. Created a formula for generating a sine wave based on variable sample rates.	5.88
2	2/11/2020 3:30pm-6pm	Met with team to discuss project after doing individual research as well create PowerPoint for mentors	2.5	2	4/19/2020 10:50 pm-12:32am	Wrote appendix sections for assignment 3 and created the design 1 diagram.	1.70
3	2/12/2020 5:30-7pm	Met with mentors to demonstrate what we found about current MCUTNC designs, provide a basic explanation of protocols involved, show a level 0 diagram, and further discuss the scope of work.	1.5	3	4/21/2020 6:10pm-11:32pm	Worked on assignment 3 with team member then began the presentations to be submitted on Sunday.	5.36
4	2/16/2020 12pm-3pm	Began preparing material to complete assignment 1 and also discussing alternatives and tradeoffs to certain aspects of the project such as PTT circuit, microcontroller, and frequency detection methods.	3	1	9/14/2020 5:30pm-7pm	Begin creating experiment sheet by overlooking sheet provided by Dr. Darby.	1.5
5	2/17/2020 6pm-1:30am	Preparing assignment 1. This involved creating a PowerPoint, the needed prospects of that such as Gantt chart, revised level 0 diagram, creating an object tree, creating alternatives and tradeoffs comparisons, and performing feasibility analysis.	7.5	2	9/14/2020 3:30pm-6pm	Use pre-written bitLoading operation to create function for converting an AFSK digital bitstream based on AX.25 flag detection.	2.5
1	2/24/2020 10am-1:30pm	Beginning to put together a data flow chart for the AX.25/KISS protocol	3.5	3	9/16/2020 5:30-7pm	Using the previous digital bitstream conversion software, implement functional logic to create an AX.25 packet.	1.5
2	2/26/2020 8pm-9:30pm	Finish flow chart with Kaleb	1.5	4	2/16/2020 12pm-3pm	Create software to output an AFSK bitstream. This was done by storing the bitstream as an array of Booleans and instruct the DAC to output a sine.	3
3	3/3/2020 6am-7am	Create BOM to present to mentors to get ahold of components	1	5	2/17/2020 12pm-1pm	Spend a bit of time looking for good prototyping boards to assemble the circuits on and connect to our microcontroller.	1
1	3/3/2020 8:30pm-11:30pm	Configuring development board with PC. The intention is to be the most informed about the development environment so team member should be able to ask me questions and I can help.	3	1	9/21/2020 5:30pm-7pm	Create BOM for prototyping components with Kobe and then send to Nolan	1.5
2	3/5/2020 8pm-10pm	Kobe ran into an issue trying to design some code to use the onboard ADC for the STM32. I helped debug the code and get the ADC working	2	2	9/22/2020 3:30pm-6pm	Worked on AX.25 handling code, implementing struct	2.5
3	3/8/2020 2pm-3pm	Review GitHub documentation and ensure it is competent and well documented.	1	3	9/23/2020 5:30-7pm	Worked on AX.25 handling code, added some packet logic based on frame received types	1.5
4	3/9/2020 10pm-12pm	Complete assignment 1B/1C with group members. I oversaw reviewing the suggested corrections with Kobe while Kaleb would adjust the document. I was also in charge of creating the PowerPoints for presentation	2	5	2/25/2020 12pm-12:30pm	Retrieved perfboard from campus	0.5
1	3/17/2020 4pm-7pm	Create audio and PTT circuit. This consisted of figuring out how these systems should I/O then design/simulate. This also consisted of presenting the circuits in a novice friendly way using the Fritzing program.	3	1	9/28/2020 5:30pm-6pm	Edited our old testing sheet to new be in word format. The previous version was okay but did not integrate into papers well.	0.5
2	3/18/2020 5pm-7pm	Recreate Level 0 then Level 1 diagram. This was done using a draw.io program. At the same time, I also corrected out file structures to have easy access to our simulation links and general structure maintenance.	2	2	9/30/2020 3:30pm-6pm	Completed testing for amplitude output of TNC. Needed to test the output with a load of 1k so I needed a simple resistor circuit.	2.5
3	3/18/2020 7pm-11pm	Work on putting together assignment 2 with team. This consisted of taking the information we gathered through the week and putting it into necessary formats for assignment 2.	4	3	10/1/2020 5:30-7pm	Completed testing for the baud rate output of TNC. Used the Digilent to measure waveform.	1.5
1	3/23/2020 5:30pm-7:44pm	Started calibrating/testing the Analog Discovery 2 and the new STM32-F446RE. Got PWM working on STM32-F446RE and being read with Analog Discovery 2.	2.2	5	10/3/2020 12pm-6:30pm	Worked on packet structure code.	6.5
2	3/24/2020 5pm-7pm	Helped Kobe get the Serial Echo program that I wrote working for his STM32-F446RE. Also spent some time with him going over how pointers, structs, and other C datatypes work in C as my program was utilizing this functionality.	2				
3	3/25/2020 6:30pm-11pm	Worked on assignment 2A with group members. We all gave comments on the paper individually then I was in charge of putting together the assignment 2A presentation.	4.5				
1	3/28/2020 5pm-7pm	Spent some time trying to find a packet program. Ended up finding something called packet engine pro that should do the job for us.	2				
2	4/1/2020 6:57pm-9:50pm	Worked on assignment 2B/2C with group members. For this session, Kobe and I came up with the OSI model shown in our paper. We also put together the presentations	2.88				

3) Kaleb Leon

Item	Date/Time	Description	Hours
1	2/5/2020 5:30 – 7:00 PM	First Meeting with Mentors	1.5
2	2/12/2020 5:30 – 7:00 PM	Second meeting with Mentors	1.5
3	2/5/2020 – 2/12/2020	Studying Communication Protocols and developing presentation for mentors.	5
4	2/14/2020 – 2/17/2020	Writing Portions of the Paper for Assignment A	6
1	2/18/2020 3:30 – 5:00 PM	Meeting with Mentors	1.5
2	2/19/2020 – 2/27/2020	Worked on Code Flow Chart	5
3	2/25/2020 – 2/28/2020	Worked on Poster for E&T	3
4	2/29/2020 – 3/2/2020	Peer Reviewed Paper for Assignment 1A	4
5	2/24/2020 – 3/3/2020	Studied how to program the microcontroller (Nucleo Board)	4
1	3/2/2020 3:00 – 5:00 PM	Finished Writing Assignment 1A	2
2	3/3/2020 – 3/9/2020	Studied Nucleo Programming tutorials and guides	5
3	3/3/2020 3:30 – 5:00 PM	Meeting with Mentors	1.5
4	3/5/2020 – 3/9/2020	Wrote Assignment 1B and 1C	6
1	3/10/2020 5:30 – 7:00 PM	Meeting with Mentors	1.5
2	3/10/2020 – 3/16/2020	Get Nucleo to transmit and receive UART Data	4
3	3/16/2020 – 3/18/2020	Finish Assignment 2	4
4	3/13/2020	Worked on Level 1 Diagram	2
1	3/26/2020 – 2/28/2020	Added on the Nucleo UART Echo code	4
2	4/1/2020 6:30 – 12:00	Added Safety concerns, fixed errors, and made OSI Model	5.5
3	3/26/2020 – 4/2/2020	Messed around on Nucleo C environment	4
1	9/14/2020 5:00 – 6:00 PM	Met with Mentors to discuss progress	1
2	9/15/2020, 9/17/2020, 9/18/2020 6:00 – 8:00 PM	Developed Testing plan	6
3	9/19/2020 4:00 – 8:00 PM	Made Testing Tree	4
4	9/17/2020 5:00 – 8:00 PM	Tested ADC and DAC code	3