Final Report

Design 2

Team 2

MCU TNC Design

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

November 13th, 2020

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 develop and conduct appropriate experimentation, analyze and interpret data, and use engineering judgment to draw conclusions?*

   This course provided us with the ability to assess others information and form our own judgements based on known data. We made many changes to our code and adapted our design based on analysis and tests. We used our engineering judgement to make important design choices. **RANK: 5**

7. *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

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, TNC, Nucleo, KISS, AX.25, HDLC, NRZ, NRZI, FSK,**

I. Introduction

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

II. Research of work done by others

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

A. KISS Mode

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

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

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

retransmission is requested. A visual diagram of an HDLC packet is shown in Figure 1.

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

Figure 1. HDLC Packet Format

C. AX.25 Protocol

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

D. AFSK (Audio Frequency Shift Keying)

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

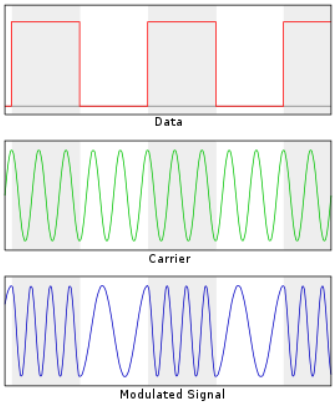


Figure 2. Audio Frequency Shift Keying

III. Project Analysis

A. Project Feasibility

The scope of work and required functional specifications are feasible.

The main goal of the project is aimed at replacing the hardware that was originally developed in the early 1980s, this means the technical requirements are very minimal. In addition, a previous design has been successfully achieved by a project called Direwolf [4] meaning the design is feasible on a technological standpoint. Also, the hardware and protocols we plan on using are well documented and are already implemented in hardware, so it is very feasible to make the shift to software related logic. Given the simplicity of packet management required to have a functioning TNC it seems it will be feasible to have close to a working design in the first half year. In addition, the second half year could be used to develop our own more optimized microcontroller board as well as add some features. According to the Gantt Chat referenced in Appendix B, our planned schedule projects the project's feasibility in the time span of one year. In the Gantt Chart we estimated the time that each part of the project would take and tried to fit it into one year. We analyzed the times and dates estimated along the critical path and found that even with some delay we will have a functioning project by one year.

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

B. Alternatives and Tradeoffs Considerations

Hardware

With a project based mostly around software there are small amounts of alternatives and tradeoffs to consider. In Appendix C, Tables are shown referencing our comparison of different alternatives to many parts of our design and our decisions. The alternatives are mostly based around specs given to us by our mentors and we chose the ones that would best meet those specs. Our first decision to make was on microcontrollers. This would play a major role in our overall design and would have to meet many of our required specs. Our three choices were the STM32L4433, the 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.

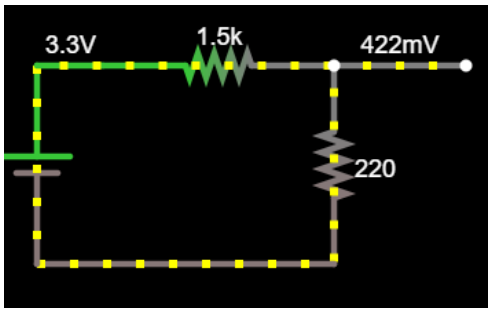


Figure 3. Simulated Voltage Divider Circuit

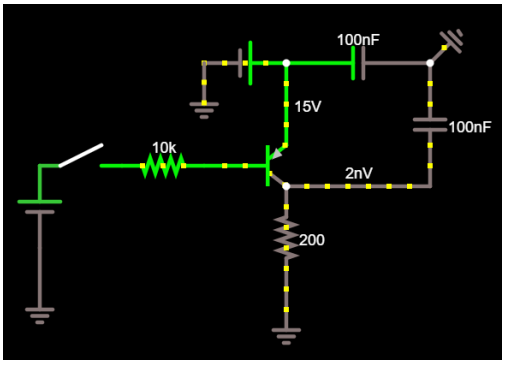


Figure 4. Simulated PTT BJT Circuit

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 addition specifications and format it into the AX.25 protocol format. In this process, we will generate the FCS bits for error checking. Lastly, the packet will be translated into an FM audio tone and transmitted to the radio to be sent off. The components and strategies to perform these tasks are subject to change past this preliminary design point throughout the testing process.

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 form 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 programed 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 are calculating the instantaneous phase of our signal. We then compare the current phase with the previously measured phase to create a delta phase. Then to produce the frequency we use the equation below:

$$A_{t0} \triangleq Amplitude\ at\ time\ t0$$

$$\varphi_{t0} \triangleq Phase\ at\ time\ t0$$

$$\varphi_{t0} = \sin^{-1}(A_{t0})$$

$$\omega = \frac{\varphi_1 - \varphi_2}{t_1 - t_2}$$

It is very important for us to increase the sampling rate as high as possible because it will increase our conversion ratio and lower our error rate. We do this by eliminating most of the math inside the interrupt callback and creating a look up table for our arcsine conversion based off values specific to our ADC. Since we could not lower the error rate enough to simply use

the raw values from this calculation, we implemented an interpreted frequency by counting the number of valid frequencies until we reach a desired minimum. At that point we can say that we successfully determined the accurate frequency value.

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.

IV. Experiments, Testing, and Data Results

A. *Experiment & Testing Style*

The testing process for our project was designed so that testing is done as we work. This is a form of AGILE workmanship. The AGILE strategy of project management is used to characterize the division of tasks into short phases and do frequent reassessment and adapt plans as needed. The short phases we used were divided by our project's functionality. These phases were defined by a planned design and sub functions. It is more clearly defined in our testing tree. The three main large phases of the project were Receiving and Transmitting code. Each of these phases/subsystems contained subassemblies and further down components.

We started by gathering and testing our components for our planned design which covered our basic leaves of the tree. From there, we combined these components into the subassemblies listed in the testing tree. After the subassemblies were constructed from tested components, we thoroughly tested them by feeding test signals and analyzing their outputs. As we moved on to major subsystems in Receiving and Transmitting, we would test our code with debug messages and basic print statements. If the outputs came at not as anticipated, we documented it and made adjustments and adapted our code and hardware. All tests and nodes of the tree are labeled with a numbering system related to the system above. This helped us keep track of our tests completed and currently being worked on. The completed tests are shown in the table below as well as in the appendix with the testing forms.

B. *Design Modifications*

Upon exploration of our project through testing and experiments, some modifications were necessary to the design. Through testing the external hardware with our new software changes it was found that some external components and connectors were unnecessary. For instance, the external amplifier made to increase the voltage of our input signals so that our microcontroller could process them was taken out. This was due to the fact that our code could handle signals of higher amplitudes do to some of the new algorithms used to handle incoming analog signals. In addition, the RS-232 port was taken out due to the feeling of it being not necessary as a communication line because UART handled all the PC to TNC communication very accurately and smoothly. Some of the major modifications are shown in the software. These modifications were discovered after looking at many different conflicting sources of information. Our initial perspective was that incoming signals into the TNC were encoded in NRZ which was stated in many different published TNC protocol documents. However, after analyzing an output from a already functioning off the shelf TNC we discovered that the incoming signals were actually encoded using NRZI. This meant that we would need to change around how we viewed all of our incoming digital signals because the bit values were now shown as mid transition changes. We ended up having to write an entirely different algorithm to handle this type of encoded data. The lack of a solid protocol and regulated information on TNC's data handling protocols made our job very confusing and made it difficult to trust one source when multiple others said conflicting points. This also why we strive to help regulate this protocol so that is easier to understand how these TNCs actually function on a data communication level.

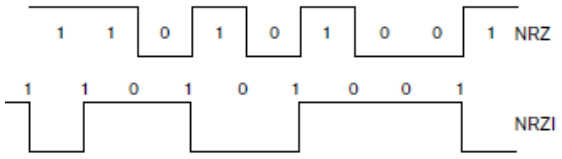


Figure 5. NRZ vs NRZI

Lastly, the lack of a defined method for analog to digital conversion was also a major issue. The information on TNC's FSK keying method and demodulation is so vague or non-existent that we only found a couple of papers that actually talk about it and let alone have an example signal to study. So, after studying the output of an off the shelf TNC we found that our original assumption of the input analog signal being NRZ FSK in the sense that a 1200 Hz tone was defined as a 1 and a 2200 Hz tone was defined as a 0 was incorrect. This was told to us by many actual hamming people and protocol documents that this was the case however we found different. As it turns out, when the audio tone changes frequency denotes a change in bits. So, a transition from one frequency to another would represent 0 and a hold of a frequency for a bit period would represent a 1.

C. *Results*

The following table shows the results from testing all our hardware subsystems and components. The left column contains the test number, and the right column contains whether the test was a pass or fail based on each tests requirement to pass or fail.

Table 1. Hardware Testing Results

Experiment Number	Results
1.2.5	Pass
1.2.4	Pass
1.2.3	Pass
1.2.2	Pass
1.2.1	Pass
1.2	Pass
1.1.2.4	Pass

1.1.2.3	Pass
1.1.2.1	Pass
1.1.2	Pass
1.1.1.1	Pass
1.1.1.2	N/A
1.1.1	Pass
1.1	Pass
1	Pass
1.3.1.1 *(Removed system)*	Fail

The following table shows the results from testing all our software subsystems and components. The left column contains the test number, and the right column contains whether the test was a pass or fail based on each tests requirement to pass or fail.

Table 1. Software Testing Results

Experiment Number	Results
2.3.1.2.2.1	Pass
2.3.1.2.3.1	Pass
2.2.1	Pass
2.1.1.1	Pass
2.1.1.2	Pass
2.1.1.3	Pass
2.1.1	Pass
2.1.2	Pass
2.1.3.1	Pass
2.2	Pass
2.2.2.1.1	Pass
2.2.2.1	Pass
2.3.2.1.1	Pass
2.3.2.1.2.1	Pass
2.3.2.1.3.1	Pass
2.3.2.1.4	Fail
2.3.2.1.4 *Rev 2*	Pass
2.3.2.1	Pass
2.3.2	Pass

In total through testing all of our components and subsystems multiple revisions were made to the code and many new sections added. In addition, some hardware deemed obsolete were removed. The result after our testing was that our major subsystems of hardware and software were fully functioning. Lastly, the combination of both hardware and software proved to be successful to get to achieve our final design.

V. Conclusion

The design, test, and build phases all came with their challenges and their benefits to the total design and implementation of the project. These difficulties were overcome by well thought out problem solving and using our resources to our advantage. Sadly, some of the resources found about the protocols and data communication of a TNC we found were incorrect. This caused the design to adapt to what we could learn by reverse engineering some of the signals from existing TNCs. However, through this the design is almost fully functional. The last steps needed to take in audio tones from a real radio rely on a small algorithm to be implemented in the future that is adaptable to the varying input amplitudes of other radio signals. However, this does not mean our design is not a success. Our design properly handles transmission and receiving using a 3.3 Volt amplitude signal. This means that it applies the basic functionality of a TNC with a modular design all inside a microcontroller. This accomplishes the main goal of our project and provides a platform for others to build atop and add more advanced TNC functionality in the future. By far, this project achieves what it set out to do in its most basic form and provides a platform of well document TNC protocol information and functionality unlike many other sources out there.

References

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

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

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

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

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

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

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

APPENDIX A

A. Scope of Work

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

B. Functional Requirements

Receiving

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

Transmitting

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

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

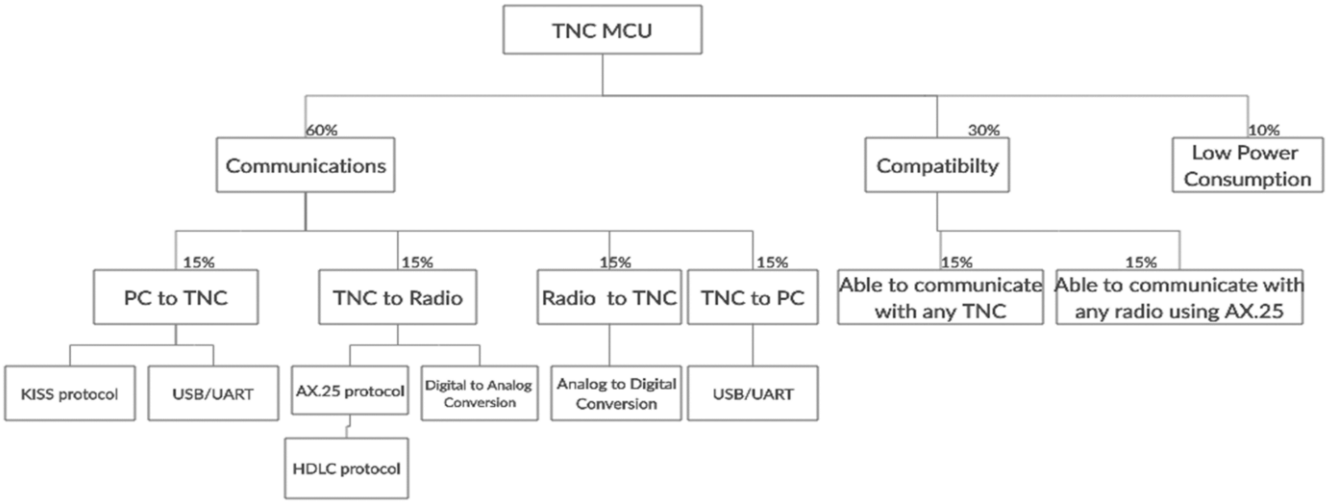


Figure A-1: Objective Tree MCU TNC

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.

A. 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.

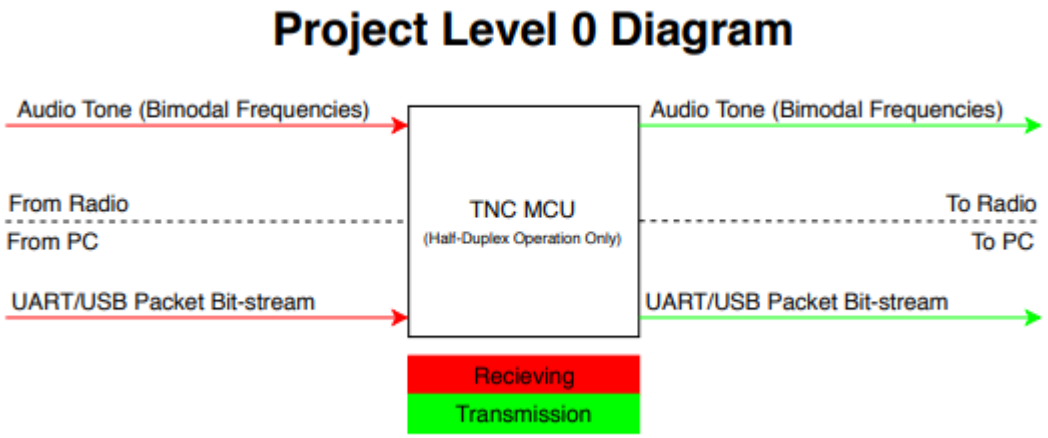


Figure A-2: Level 0 Diagram

B. 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.

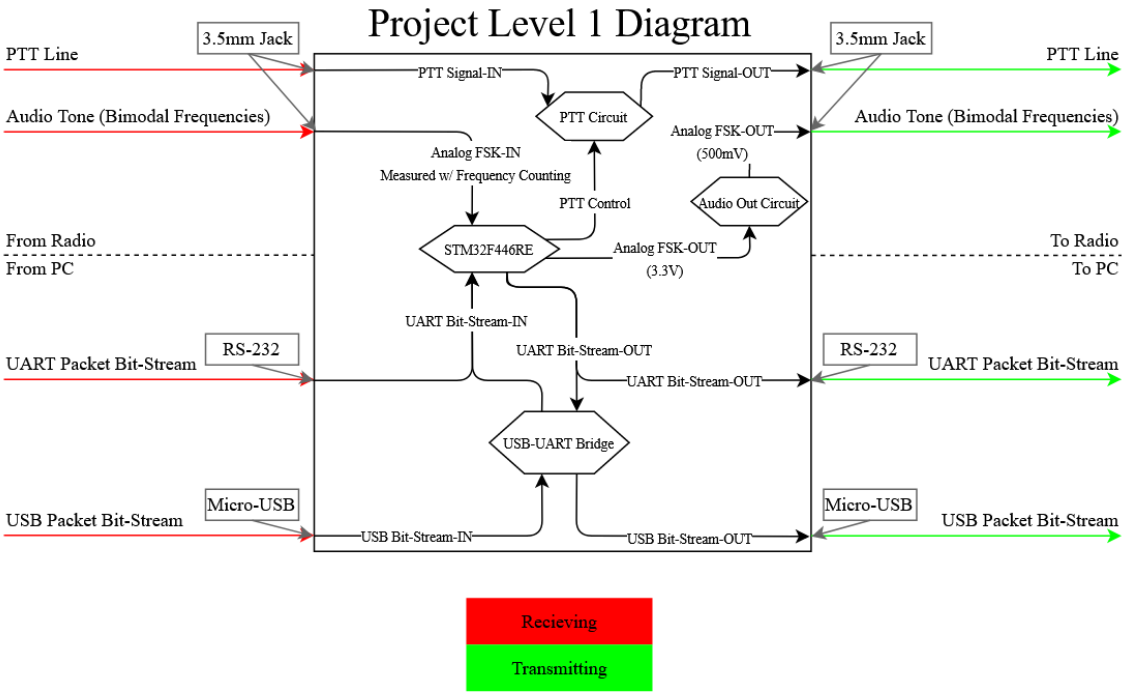


Figure A-3: Level 1 Diagram

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 kΩ, 4.7 kΩ, 470 Ω)	Yes	$0.00
Capacitor (100 nF)	Yes	$0.00
USB logic analyzer	Yes	$0.00
breadboard	Yes	$0.00
3.5 mm/2.5 mm jack	Yes	$2.09/$2.99
jumper wires(M-M, M-F,F-F)	Yes	$0.00
RS232 Cable	No	2 X ($4.29)
Total		$31.51

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

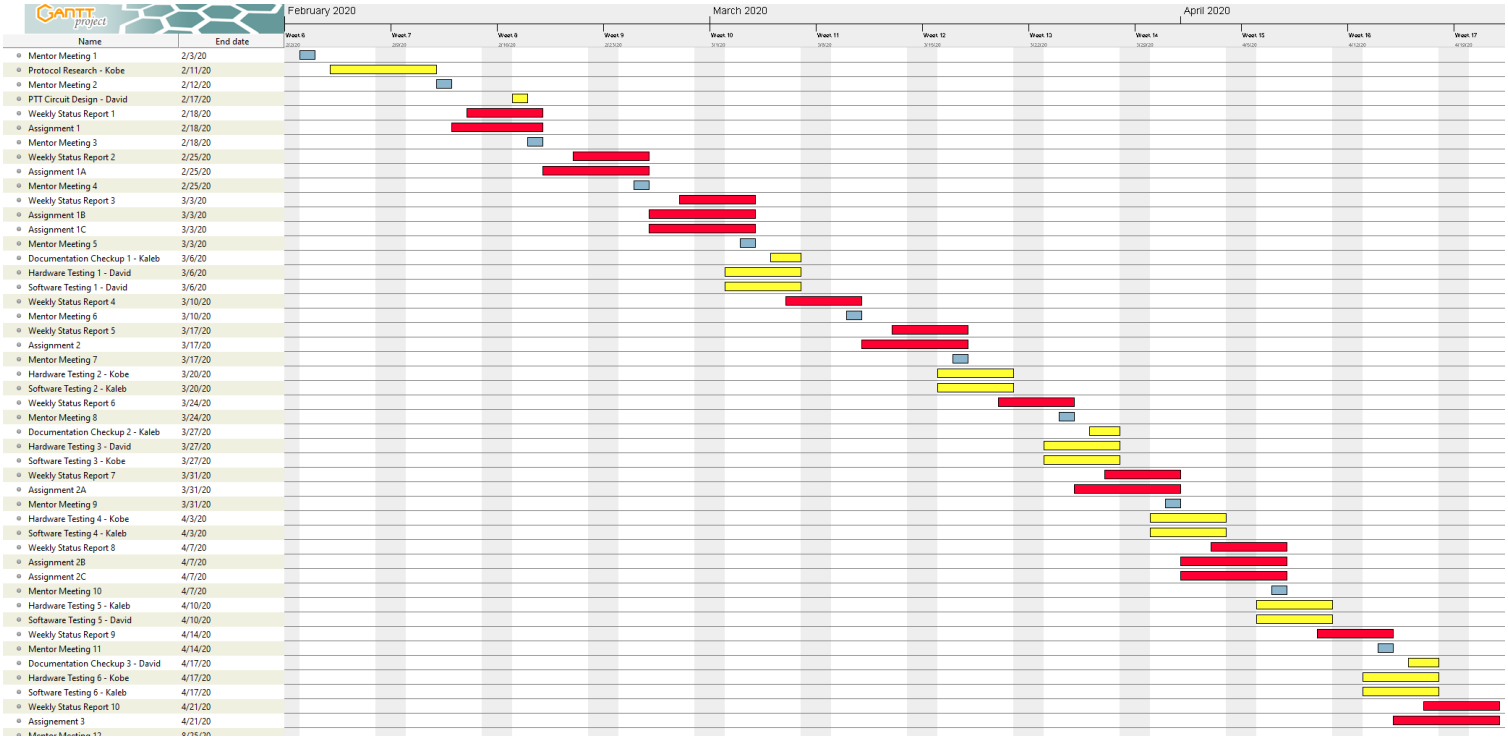


Figure B-1: Gantt Chart Semester One

APPENDIX C

A. *Alternatives and Tradeoffs Analysis*

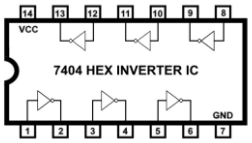
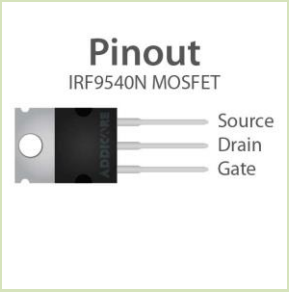
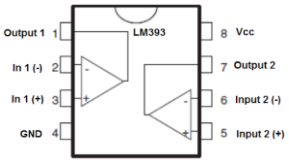
1) Microcontroller Comparison

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 contain the CRC calculation unit. The STM32L4433 is the perfect fit for our design since it has DACs and ADCs for transmitting and receiving data. It also contains a CRC calculation unit which would help detect errors, if any, when transmitting or receiving. The only downside is that we are not familiar with coding in embedded C, but tutorials and guides can aid us in that aspect.

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

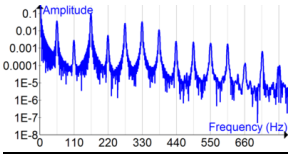
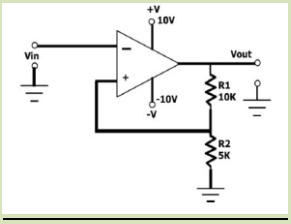
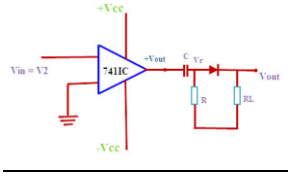
2) PTT

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

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

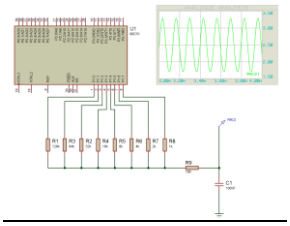
3) ADC

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

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

4) DAC

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

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

5) Algorithms

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

APPENDIX D

Preliminary Design

A. Receiving Flowchart

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

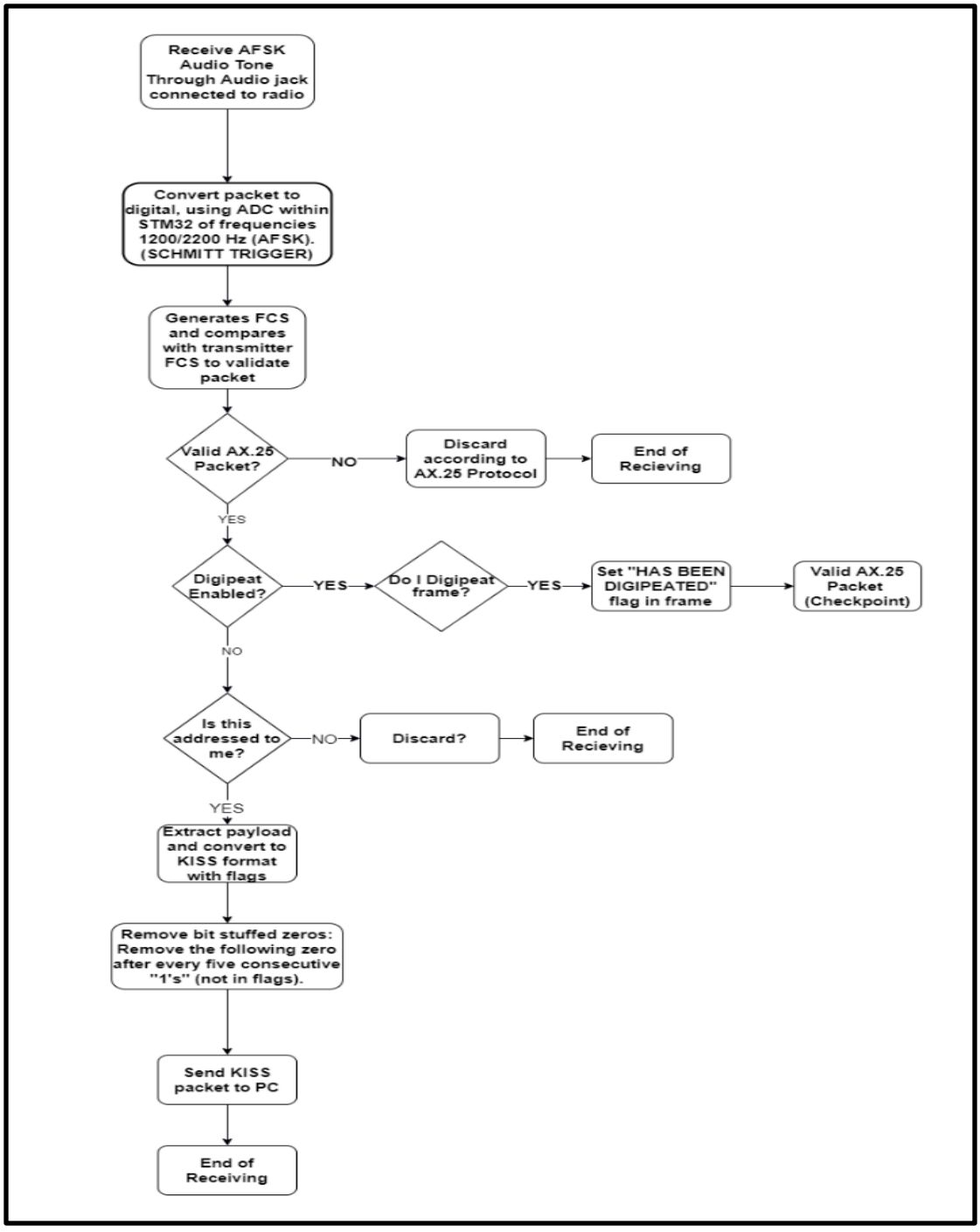


Figure D-1. Flowchart of the receiving process

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.

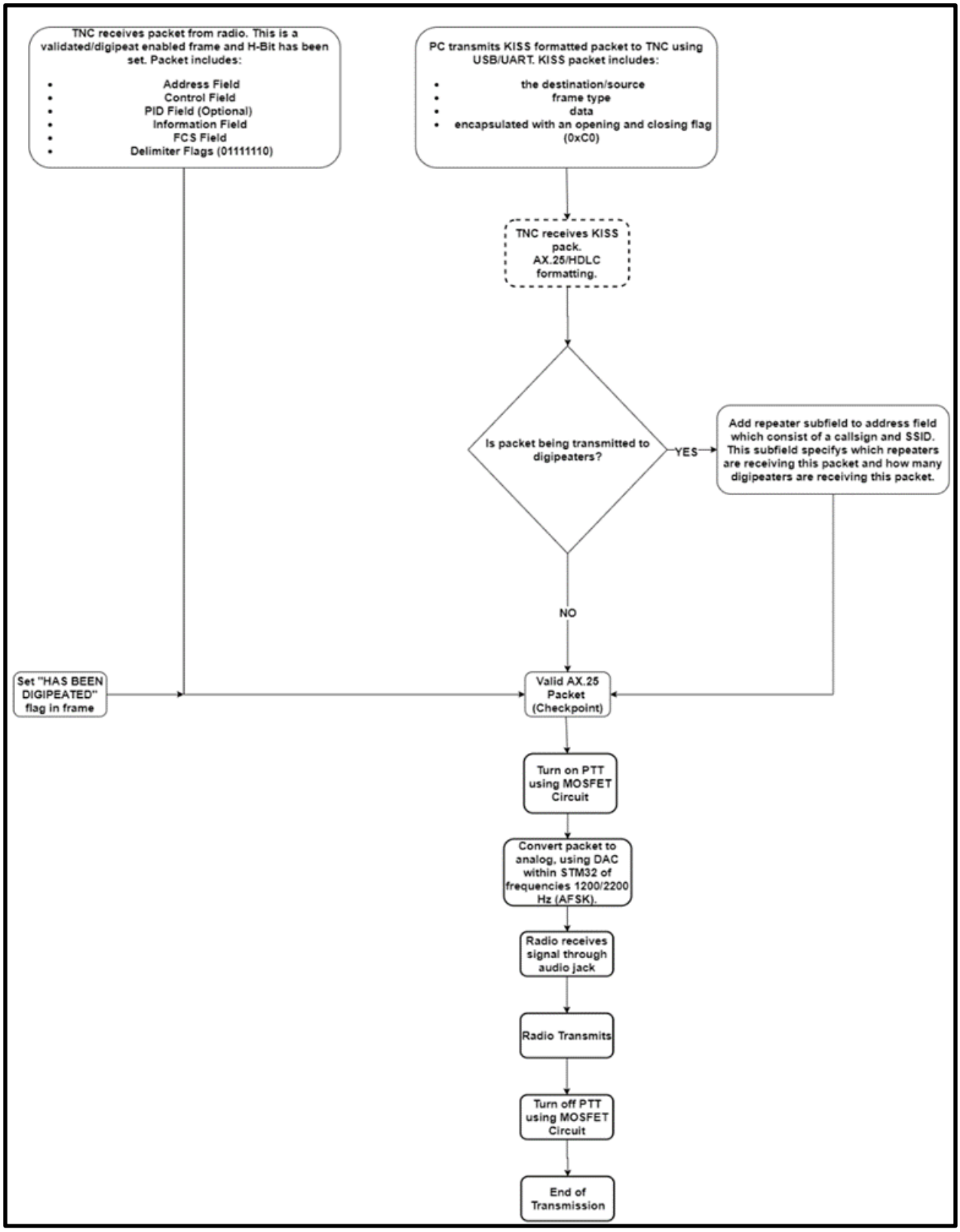


Figure D-2. Flowchart of Transmitting Process

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.

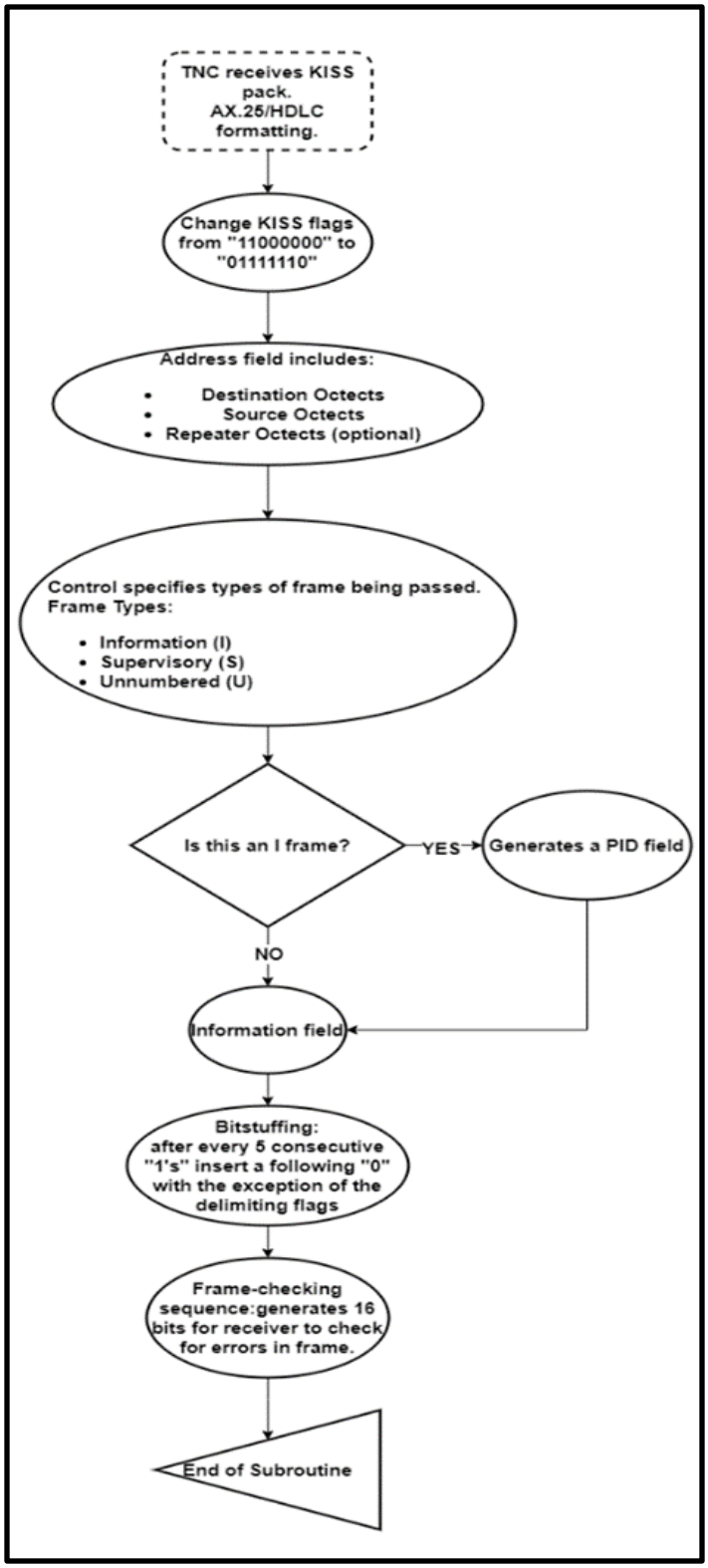


Figure D-3. Flowchart of HDLC Packet Formatting

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.

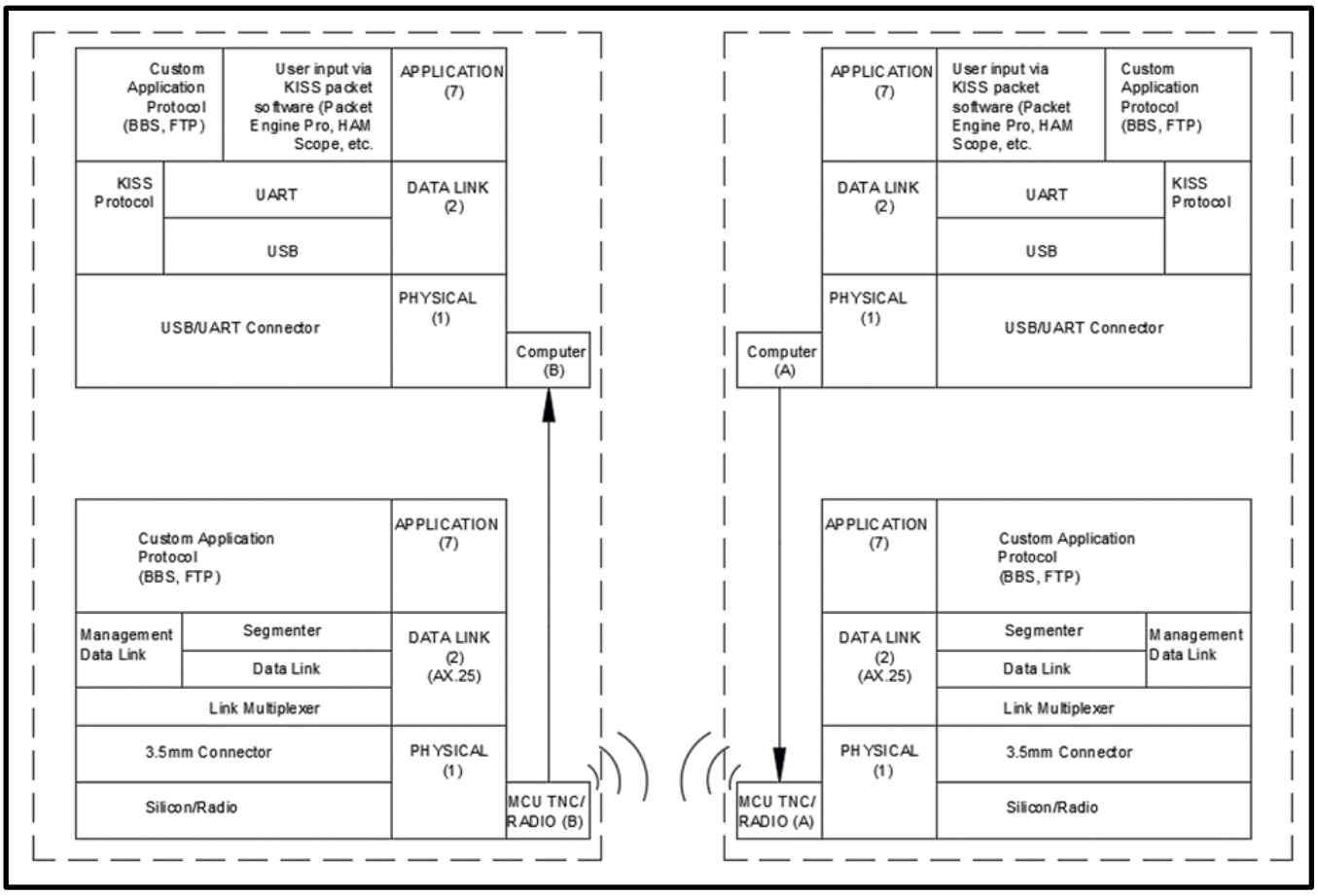


Figure D-4. OSI Layer Communication Model

E. Failure Modes and Effect Analysis (FMEA)

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

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

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.)

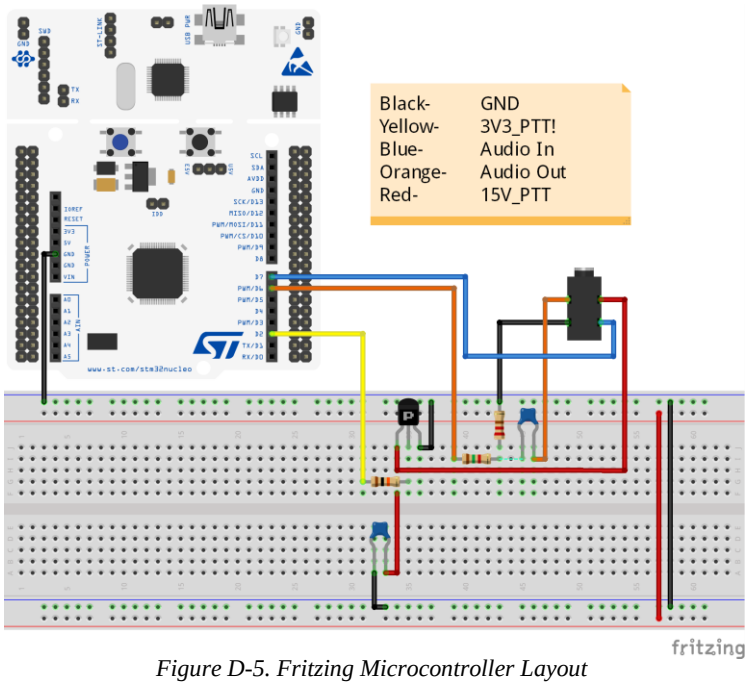


Figure D-5. Fritzing Microcontroller Layout

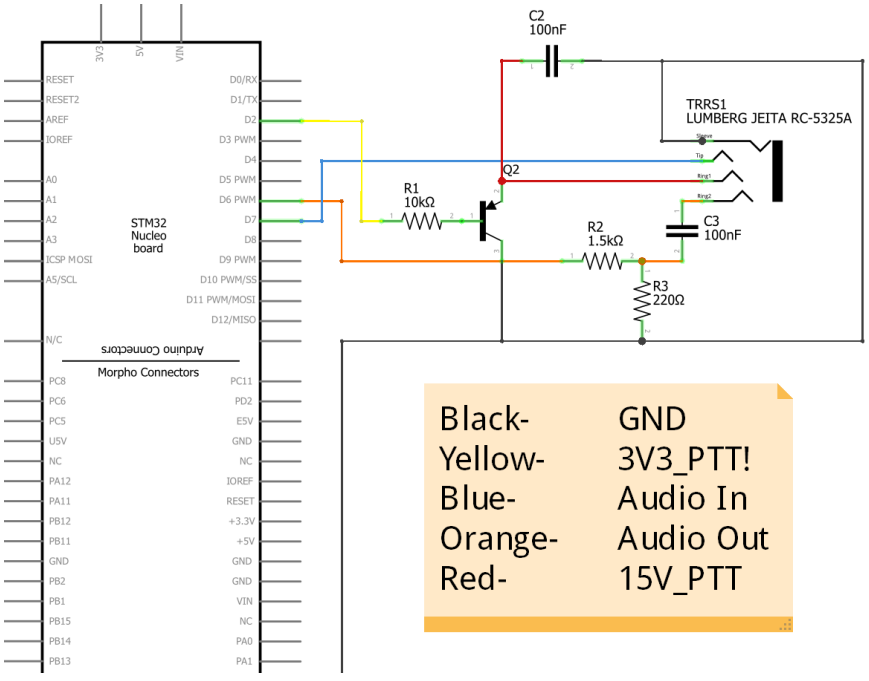


Figure D-6. Fritzing Microcontroller Connections

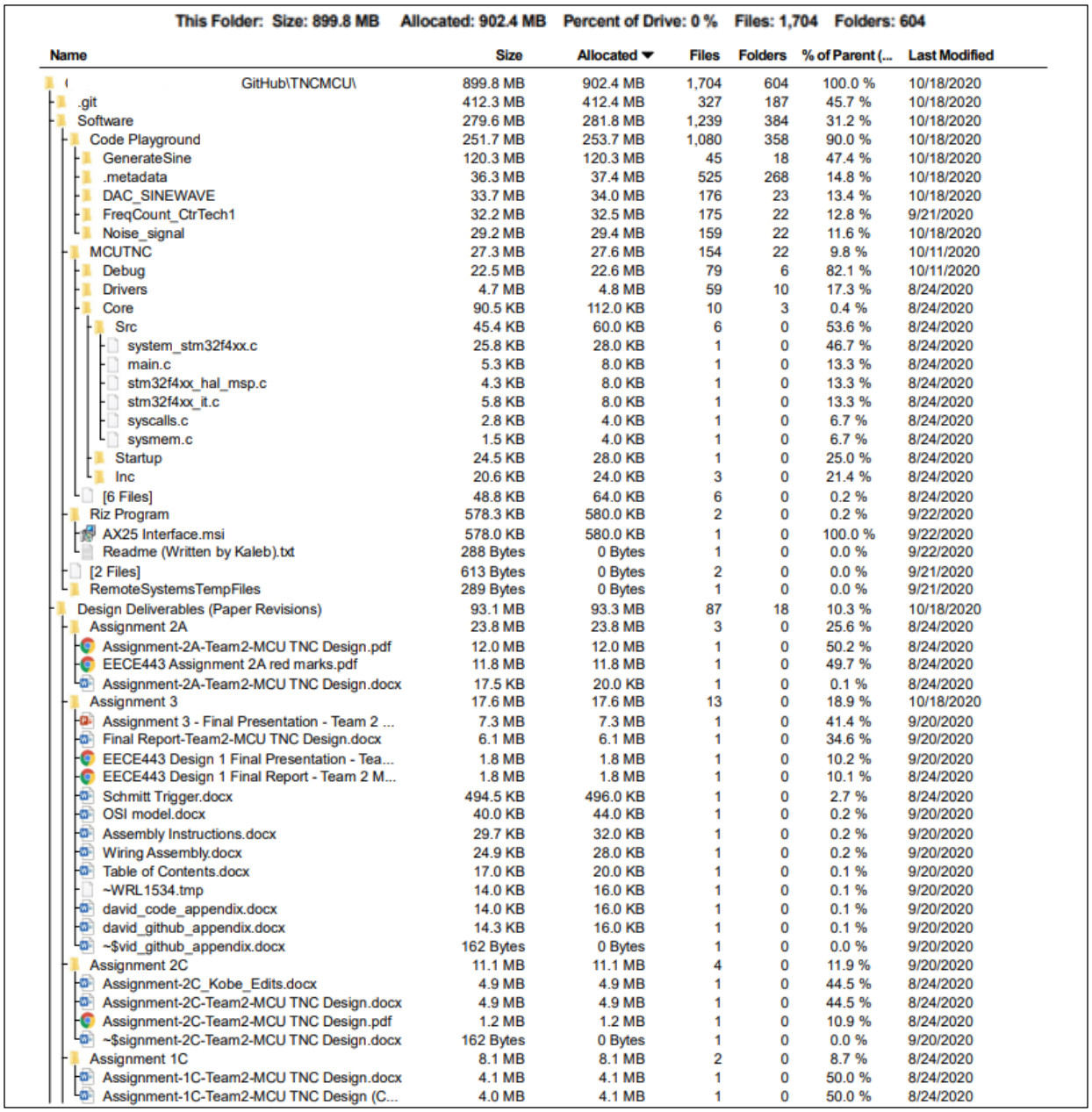
APPENDIX E

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 reversable as every time a change to a file occurs, the user most 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.

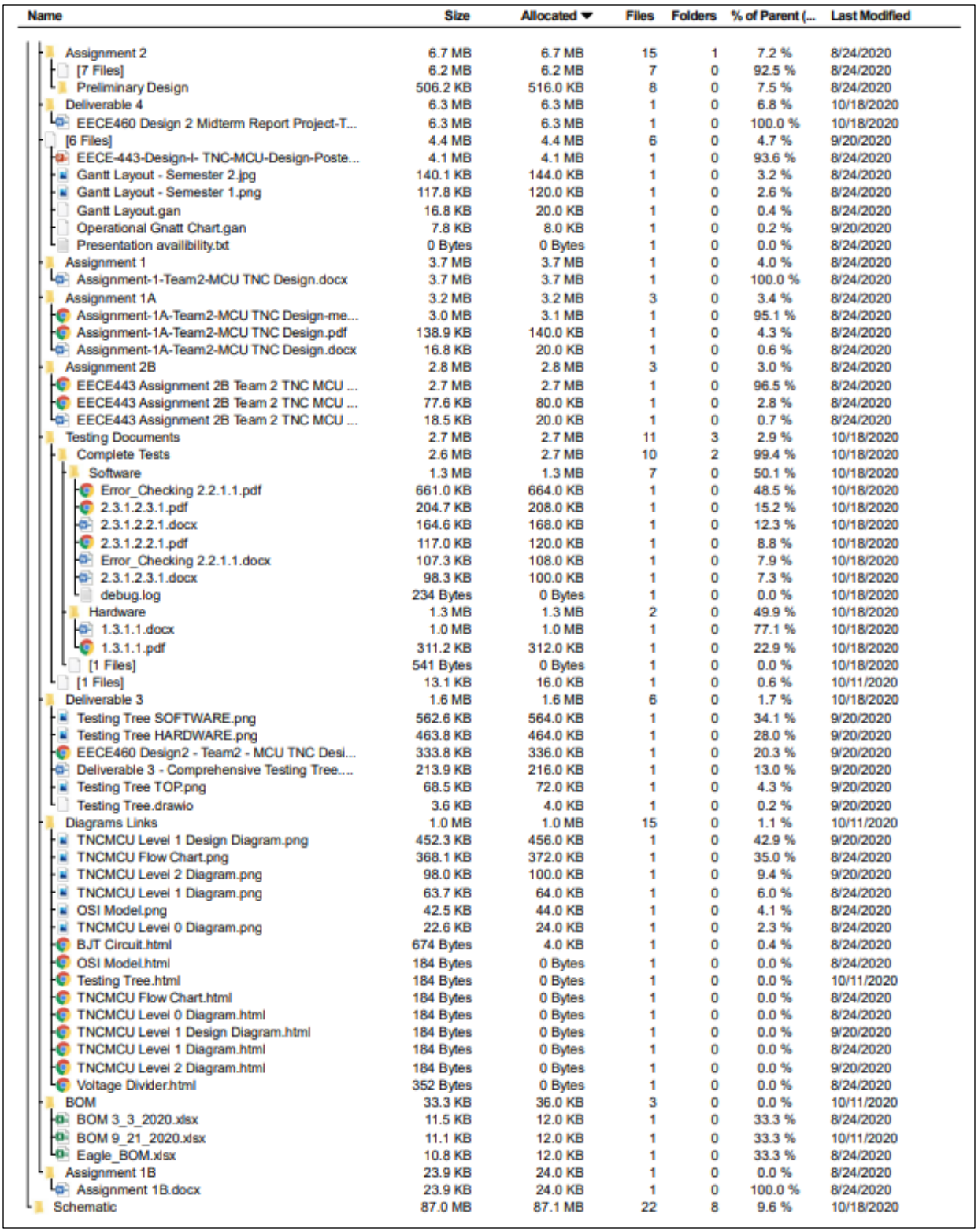
The structure for the file system is designed to be organized and easy to find whatever is needed. The root directory will have folder that will list general areas of the project such as Design Deliverables, Presentations, Software, Schematics, or Protocol Documentation. Within these folders, the files 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

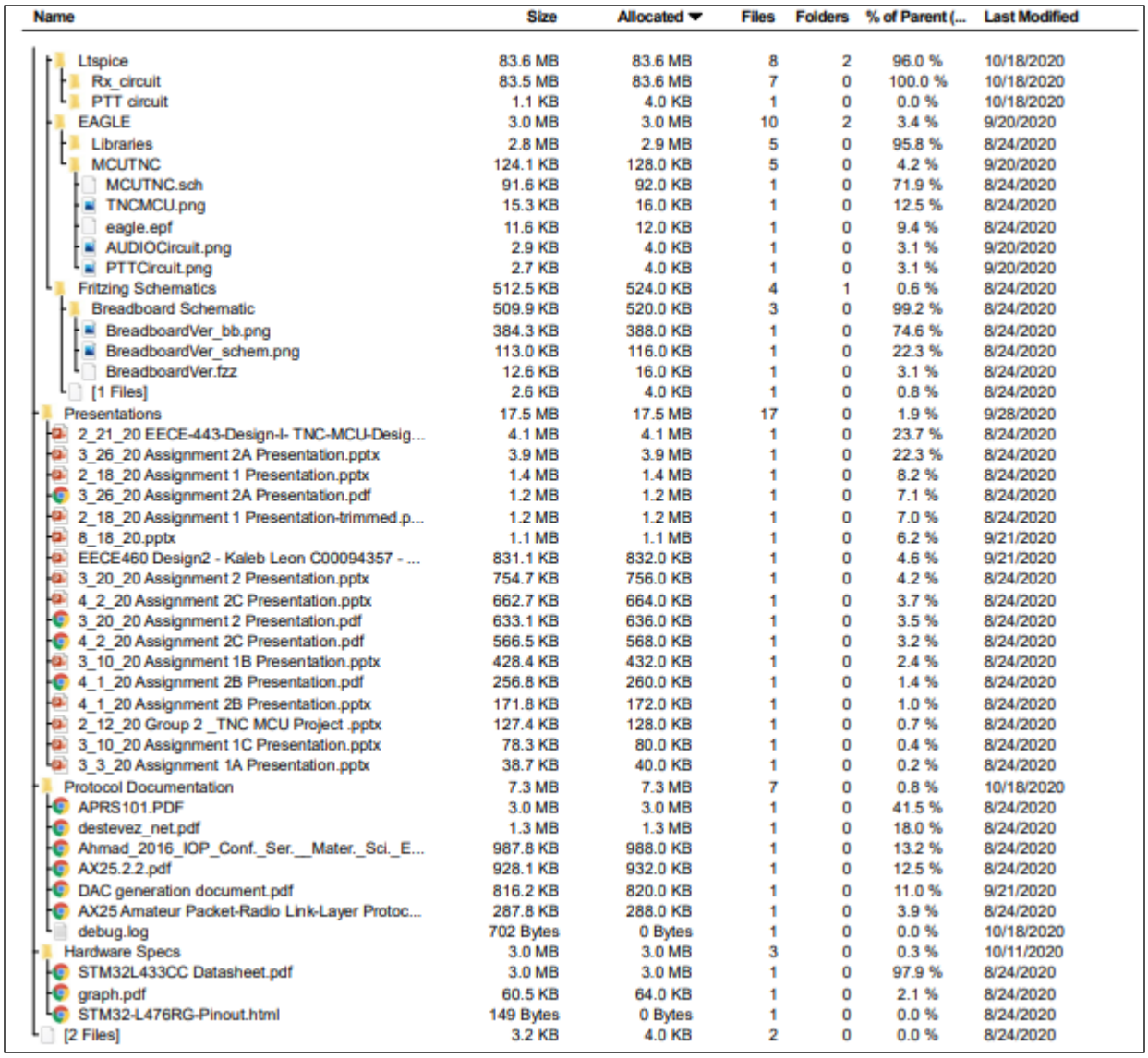
Name	Size	Allocated	Files	Folders	% of Parent (...	Last Modified
GitHub\TNCMCU\	899.8 MB	902.4 MB	1,704	604	100.0 %	10/18/2020
.git	412.3 MB	412.4 MB	327	187	45.7 %	10/18/2020
Software	279.6 MB	281.8 MB	1,239	384	31.2 %	10/18/2020
Code Playground	251.7 MB	253.7 MB	1,080	358	90.0 %	10/18/2020
GenerateSine	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
sysmem.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
~$vid_github_appendix.docx	162 Bytes	0 Bytes	1	0	0.0 %	9/20/2020
Assignment 2C	11.1 MB	11.1 MB	4	0	11.9 %	9/20/2020
Assignment-2C_Kobe_Edits.docx	4.9 MB	4.9 MB	1	0	44.5 %	8/24/2020
Assignment-2C-Team2-MCU TNC Design.docx	4.9 MB	4.9 MB	1	0	44.5 %	8/24/2020
Assignment-2C-Team2-MCU TNC Design.pdf	1.2 MB	1.2 MB	1	0	10.9 %	8/24/2020
~$signment-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 Gnatt Chart.gan	7.8 KB	8.0 KB	1	0	0.2 %	9/20/2020
Presentation availibility.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
Ltspice	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
PTTCircuit.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
destevez_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

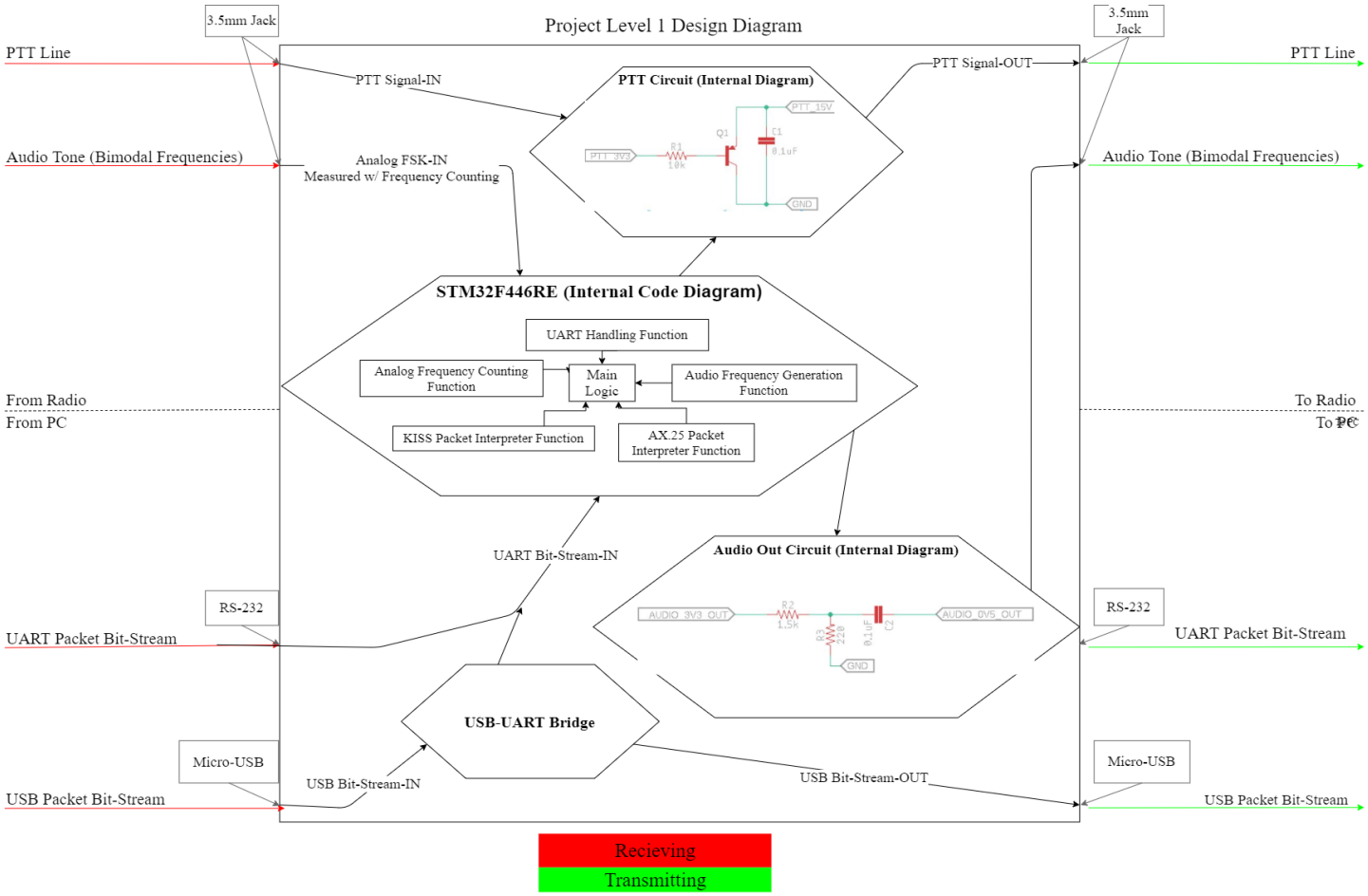


Figure E-4. Level 1 Design Diagram

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/(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:

Frequency = (toggle count * Clock frequency)/(Clock Prescaler * 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:

```
Initalize 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));
```

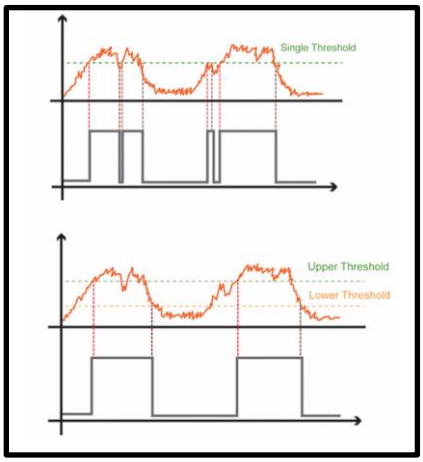


Figure E-5 Threshold Trigger Example

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 = (Input Clock Frequency)/(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:

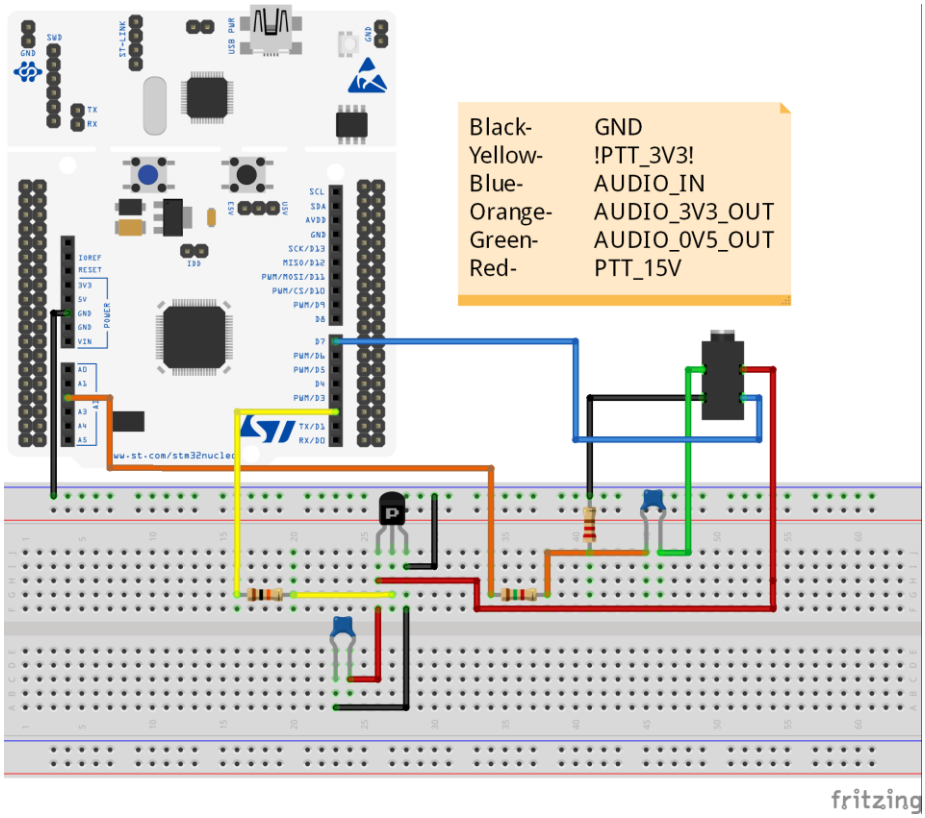


Figure E-5. Breadboard Wiring Schematic

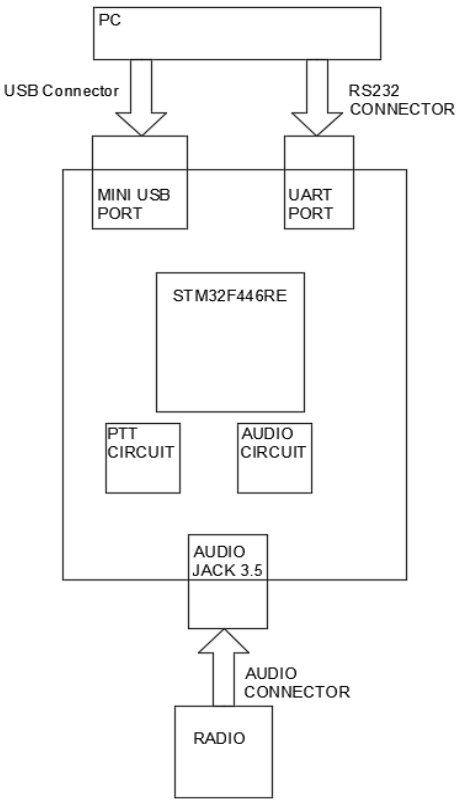


Figure E-6. Simple External Connections

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 connecter 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.

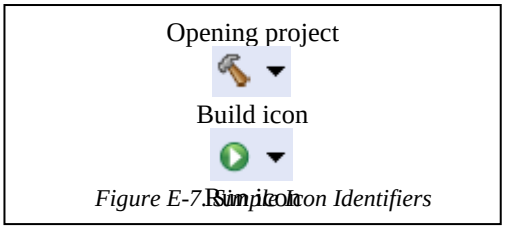


Figure E-7. Simple Icon Identifiers

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.

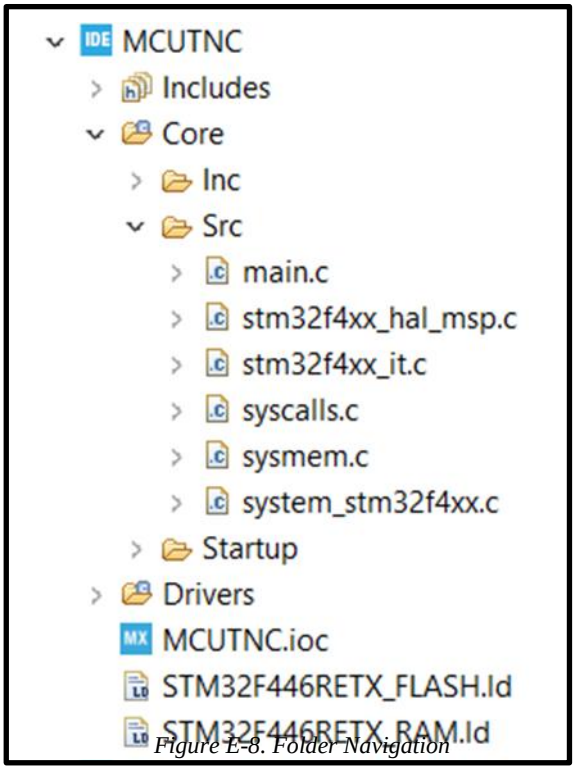
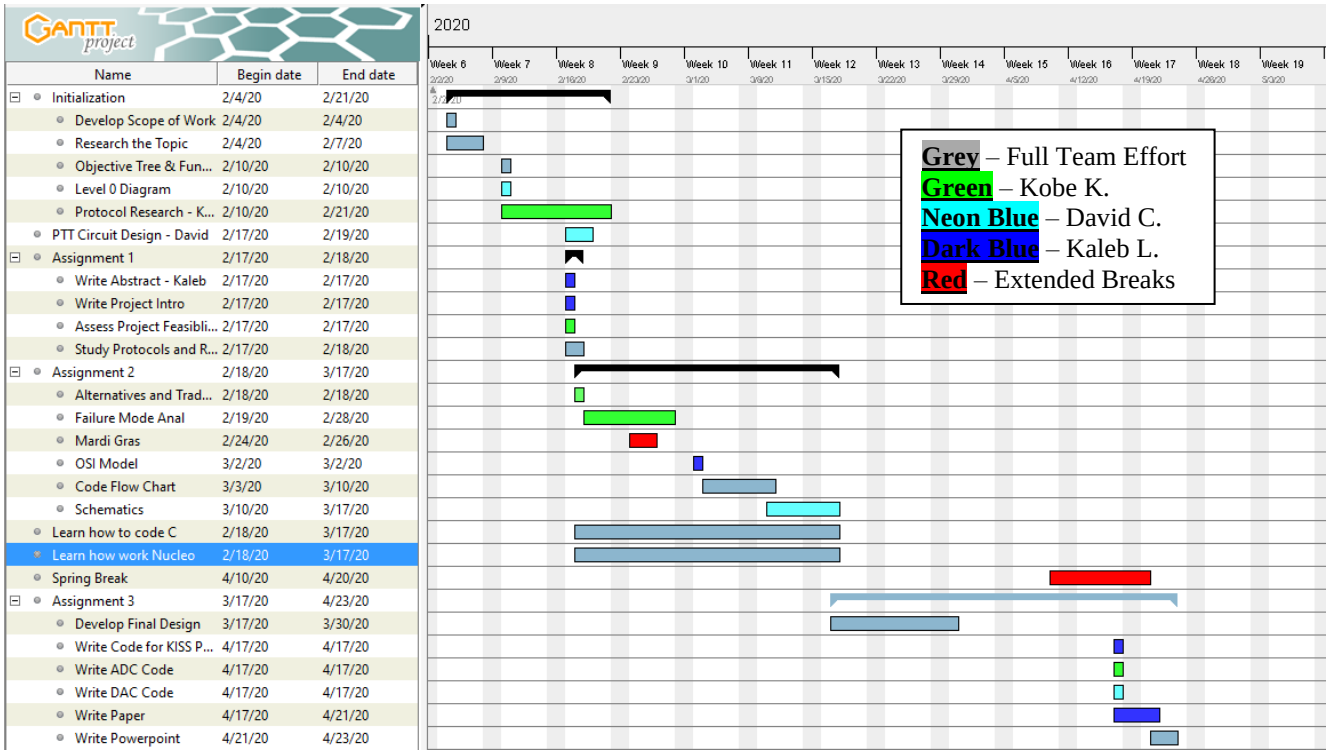


Figure E-8. Folder Navigation

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.



Name	Begin date	End date
Initialization	2/4/20	2/21/20
Develop Scope of Work	2/4/20	2/4/20
Research the Topic	2/4/20	2/7/20
Objective Tree & Fun...	2/10/20	2/10/20
Level 0 Diagram	2/10/20	2/10/20
Protocol Research - K...	2/10/20	2/21/20
PTT Circuit Design - David	2/17/20	2/19/20
Assignment 1	2/17/20	2/18/20
Write Abstract - Kaleb	2/17/20	2/17/20
Write Project Intro	2/17/20	2/17/20
Assess Project Feasibli...	2/17/20	2/17/20
Study Protocols and R...	2/17/20	2/18/20
Assignment 2	2/18/20	3/17/20
Alternatives and Trad...	2/18/20	2/18/20
Failure Mode Anal	2/19/20	2/28/20
Mardi Gras	2/24/20	2/26/20
OSI Model	3/2/20	3/2/20
Code Flow Chart	3/3/20	3/10/20
Schematics	3/10/20	3/17/20
Learn how to code C	2/18/20	3/17/20
Learn how work Nucleo	2/18/20	3/17/20
Spring Break	4/10/20	4/20/20
Assignment 3	3/17/20	4/23/20
Develop Final Design	3/17/20	3/30/20
Write Code for KISS P...	4/17/20	4/17/20
Write ADC Code	4/17/20	4/17/20
Write DAC Code	4/17/20	4/17/20
Write Paper	4/17/20	4/21/20
Write Powerpoint	4/21/20	4/23/20

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 connecter 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.

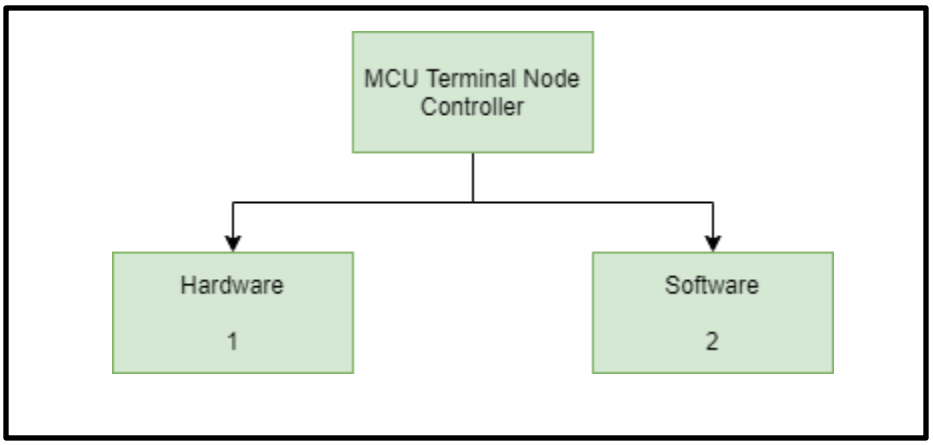


Figure F-1. High Level Comprehensive Testing Tree

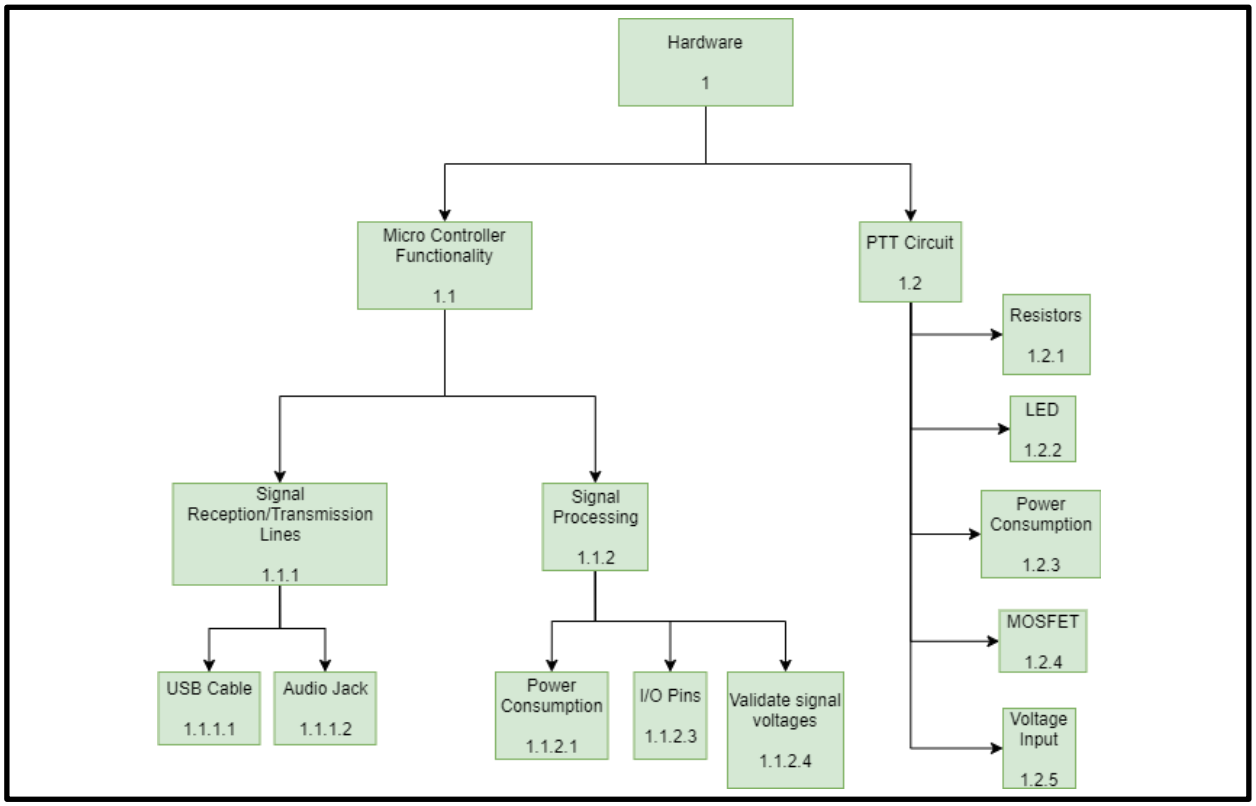


Figure F-2. Hardware Comprehensive Testing Tree

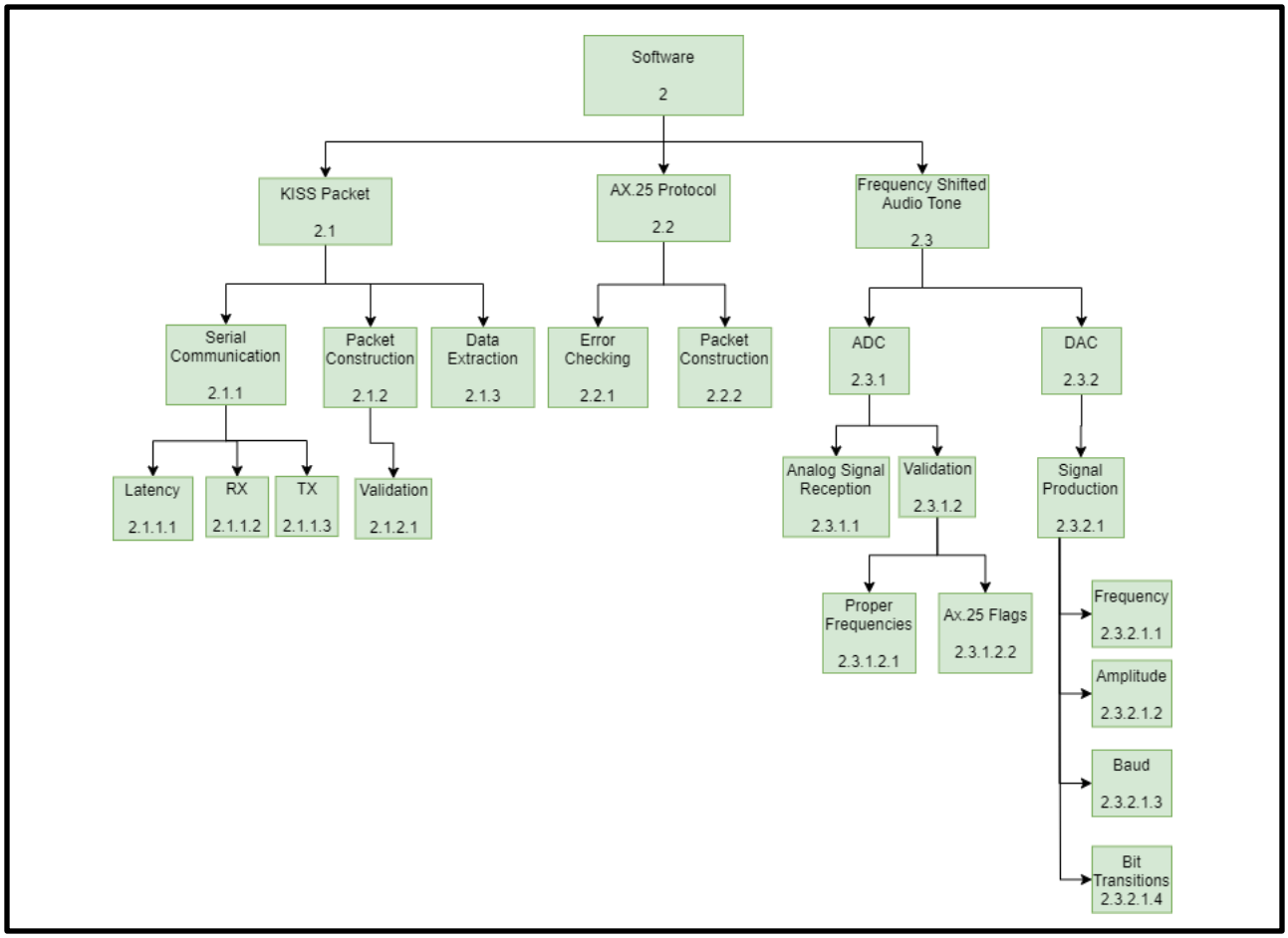


Figure F-3. Software Comprehensive Testing Tree

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:

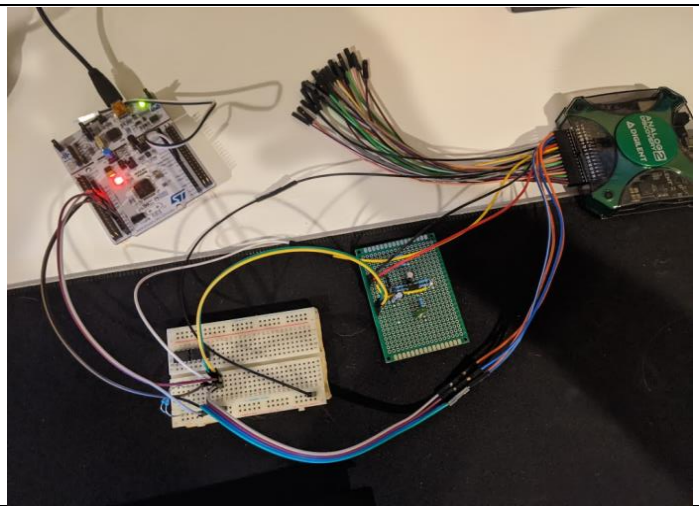
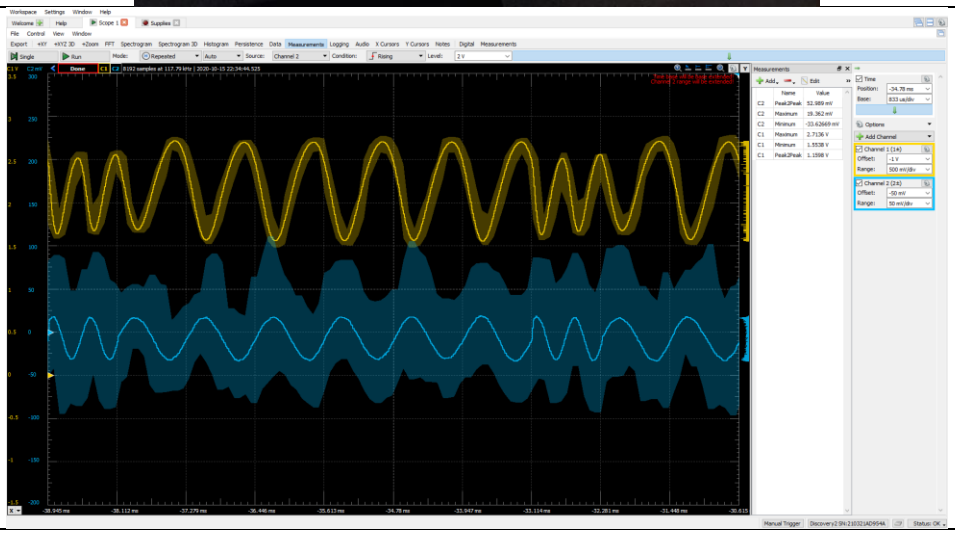
1. What is needed to be done
2. The code or circuit being worked on
3. The code functionality, problem, circuit, and output needed to be tested or created
4. The testing procedure and data collection method
5. Expected results
6. If results not met in one sitting setup a time to revisit

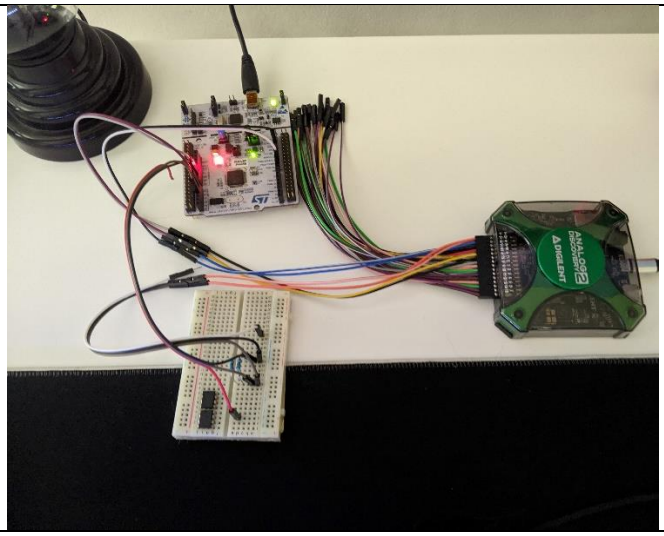
Per experiment we would perform the steps above. Each of us would take action in handling different portions of the design but we follow this same path no matter what task. We code and test things as we go to make sure we are producing accurate results. It is similar to how an AGILE workflow works. We 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.

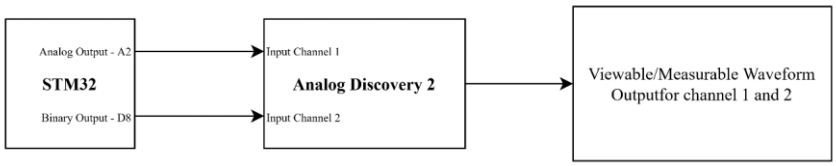
F. Final Presentation Demo

The final presentation demo will consist of two of our microcontroller TNCS with the same software on each. One will be connected to COM6 and the other connected to COM3 of a PC. The first TNC will receive a KISS packet from the PC using our mentor's software and it will format it into AX.25 and form the AFSK signal. It will then send that signal over the wire to the other TNC's analog pins. The TNC that receives that signal will take it and extract out the NRZI encoded digital signal and then extract the Ax.25 packet out of it. Then, it will take the Ax.25 packet and extract the KISS packet out of that. That KISS packet will be sent over the COM port of the second controller to the serial monitor. We will demonstrate this happening both ways with both micro controller TNCs.

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

TNC Testing Form (REV1)	
Leaf on the Tree	Baud
Device Under Test (Testing Tree Number):	2.3.1.2.3.1
Date:	10/4/2020
Person(s) Conducting Experiment:	David Cain
Signature:	
Experiment Purpose:	The purpose of this experiment is to measure and ensure the number of signaling events per second (or baud rate) is correctly established as 1200Hz
Experiment Procedure:	To verify the baud rate, a diagnostic signal will be enabled in software to output the current transmission bit value represented in binary. This binary wave form can easily have baud rate measured.
Equipment Settings / Software Settings (w Revision):	Analog Discovery 2 input channel 1 and 2 will be connected to the STM32 output pins D8(PA9) and A2(PA4)
Testing Diagram / Picture:	
Data Points:	
Pass / Fail:	Pass
Interpreted Notes:	Waveform is sustaining a baud rate of 1200Hz. This was tested with multiple wave forms but easily viewed with alternating bit pattern.
Recommendations for Modifications:	None

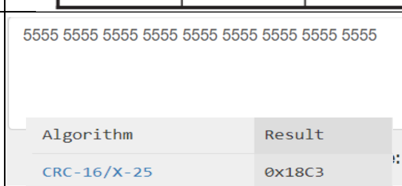
TNC Testing Form (REV1)	
Leaf on the Tree	Error Checking
Device Under Test (Testing Tree Number):	2.2.1
Date:	10/10/2020
Person(s) Conducting Experiment:	Kobe Keopraseuth
Signature:	
Experiment Purpose:	The purpose of this experiment is to verify an inputted packet has the correct FCS field, by performing a crc check on the other given subfields (Address, Control, PID, Info) as shown in the testing diagram.
Experiment Procedure:	To verify that it correctly verifies the FCS field, I made 2 testing array inputs, with a size greater than 120, which is the minimum. One array contains a correct FCS field and the other does not. Both arrays contain an input of 0x555555555555555555555555555555555555 (36 fives), excluding the FCS field. According the crc calculator, this input should have a crc output of 0x18c3. To verify that the generated crc is valid with the given input array, an online crc calculator was used.
Equipment Settings / Software Settings (w Revision):	Code will be implemented in Code Blocks IDE and it print out the input array, array after bit stuffing, the subfields obtained from the input array, FCS field in hexadecimal, crc calculation, and the result of whether the FCS field is valid or not.
Testing Diagram / Picture:	

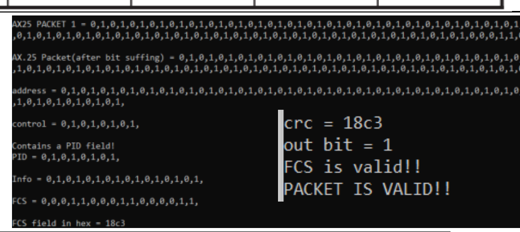
Flag	Address	Control	Info	FCS	Flag
01111110	112/224 Bits	8/16 Bits	N*8 Bits	16 Bits	01111110

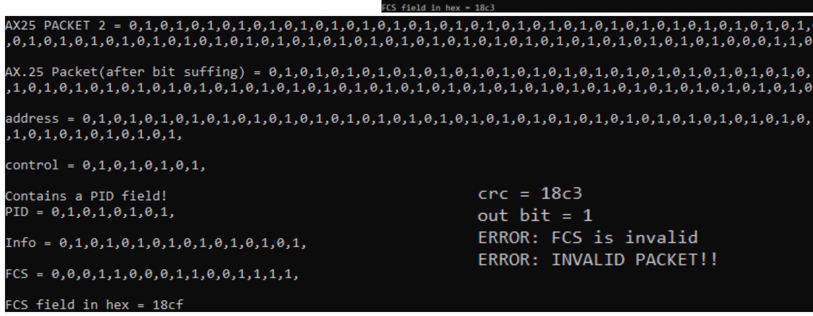
Figure 3.1a. U and S frame construction.

Flag	Address	Control	PID	Info	FCS	Flag
01111110	112/224 Bits	8/16 Bits	8 Bits	N*8 Bits	16 Bits	01111110

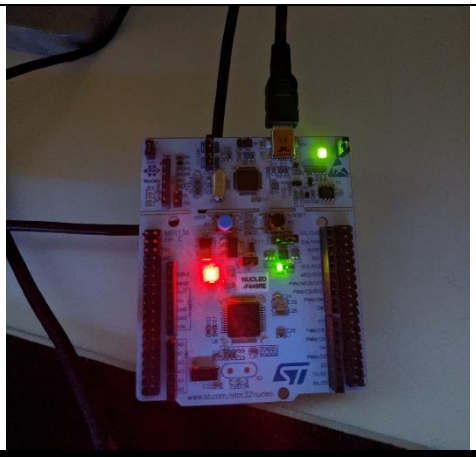
Data Points:

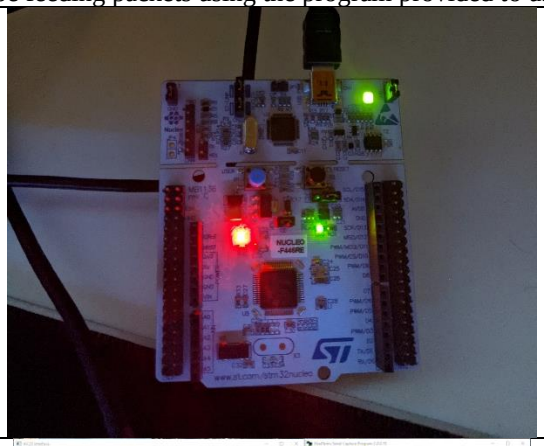
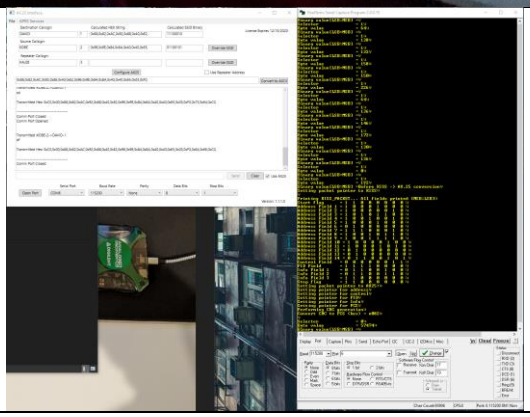


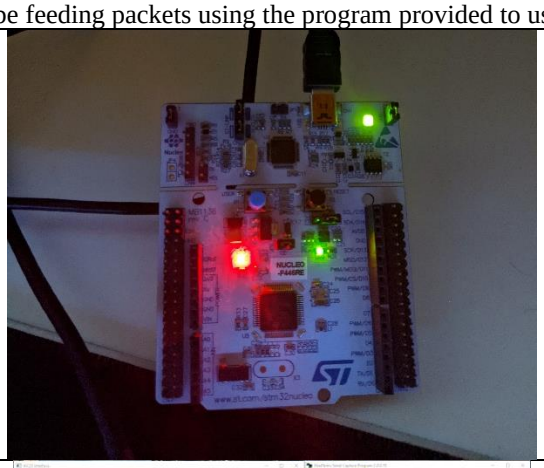
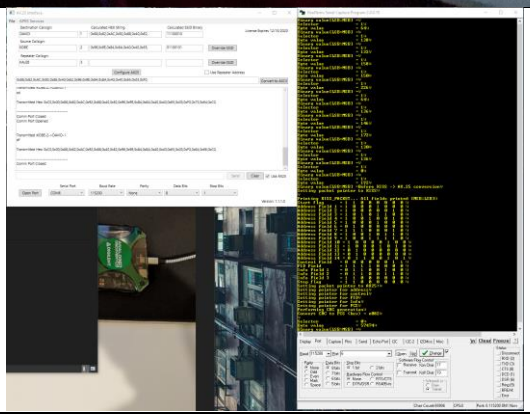


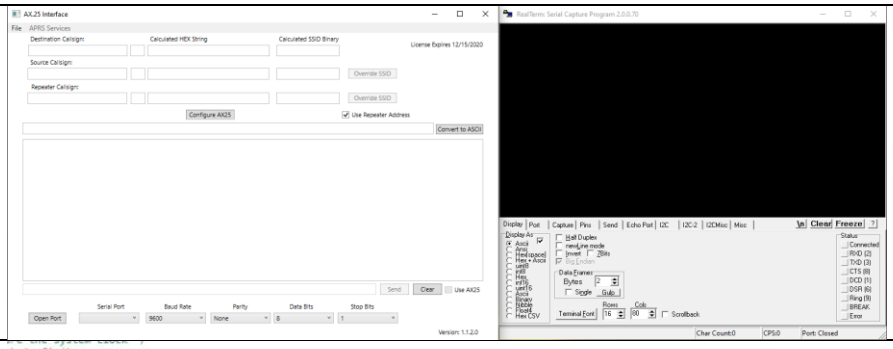
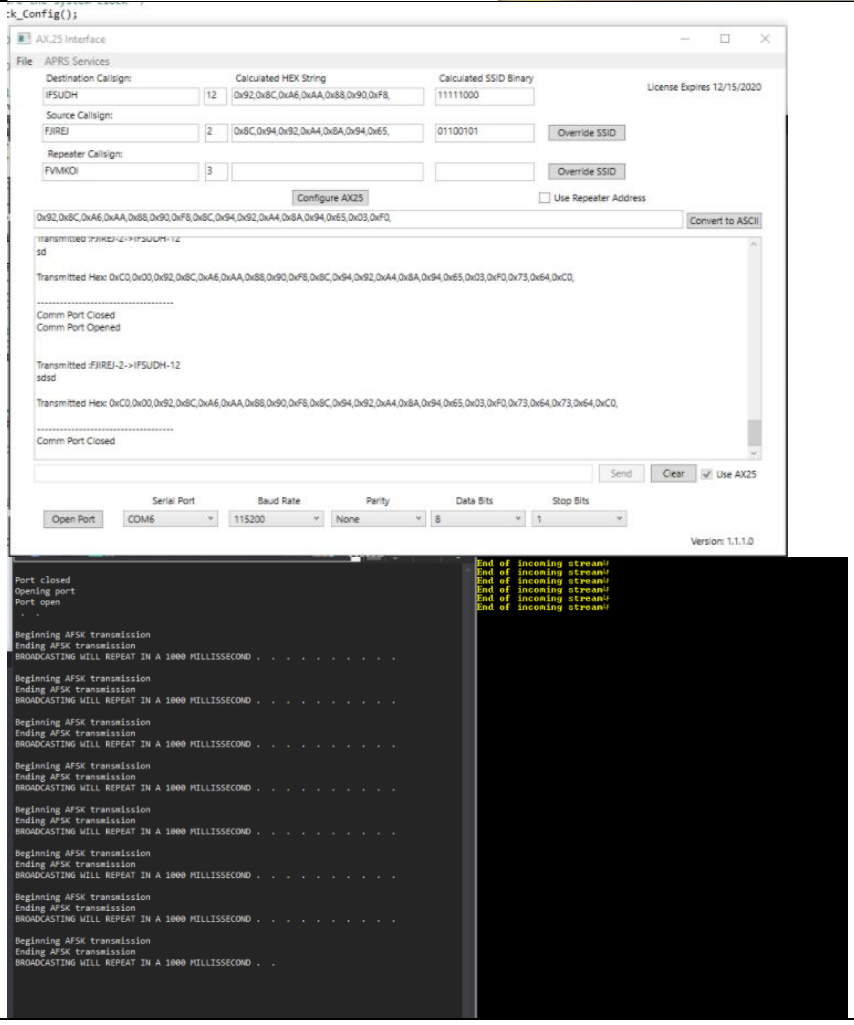


Pass / Fail:	PASS
Interpreted Notes:	Code does verify the FCS field correctly. This was tested with different given FCS fields and different bits for the input array.
Recommendations for Modifications:	None

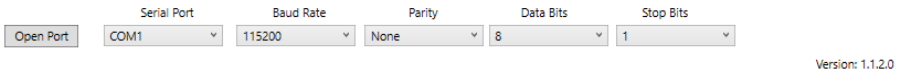
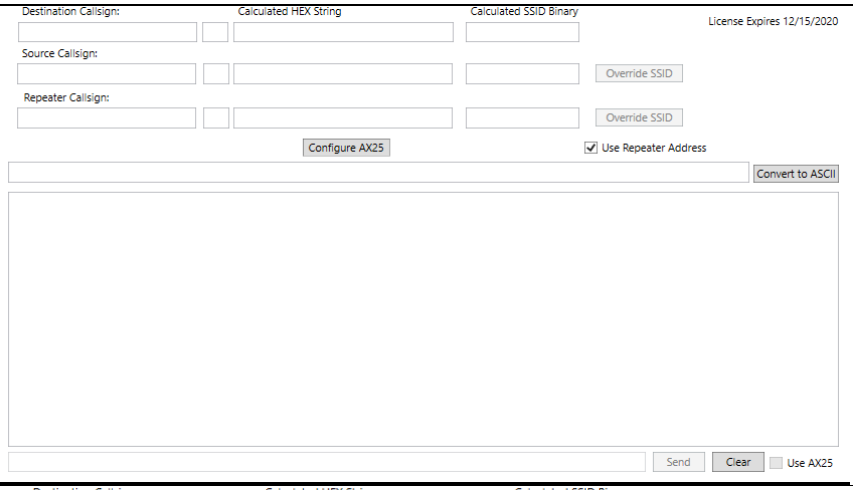
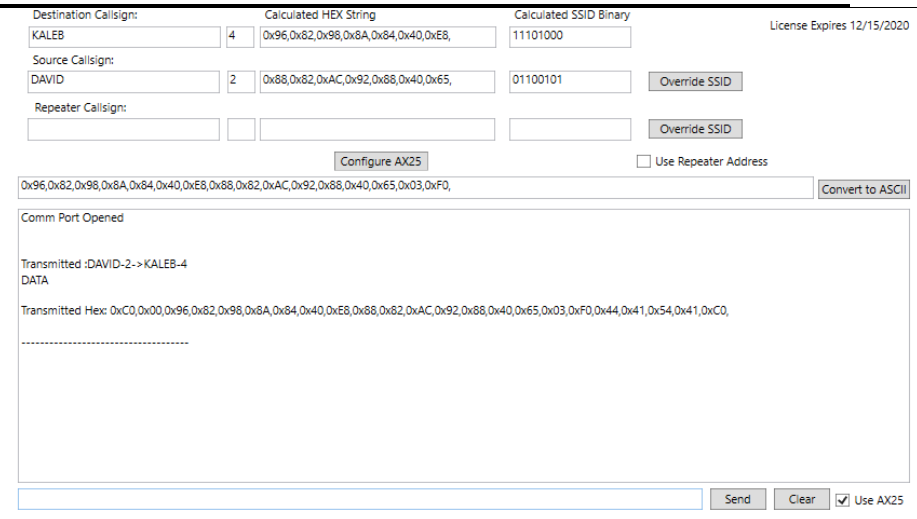
TNC Testing Form (REV1)	
Leaf on the Tree	Latency
Device Under Test (Testing Tree Number):	2.1.1.1
Date:	10/31/2020
Person(s) Conducting Experiment:	David Cain
Signature:	
Experiment Purpose:	The purpose of this experiment is to ensure that the latency of the microcontroller when transmitting data is within an acceptable time. A specified time was not given for the project, but this is an important consideration from a user perspective.
Experiment Procedure:	I will setup a software timer that begins the moment the controller receives a KISS packet over UART. This timer will run until the controller begins to repeatedly output the same message in broadcasting mode.
Equipment Settings / Software Settings (w Revision):	Working entirely within our own software. This is not complete but it is mostly finished.
Testing Diagram / Picture:	
Data Points:	Total transmission time was: 12032LF Beginning AFSK transmissionLF Ending AFSK transmissionLF BROADCASTING WILL REPEAT IN A 2000 MILLISSECOND . . . . . LF LF
Pass / Fail:	Pass
Interpreted Notes:	The device takes ~1.2ms and this is deemed acceptable.
Recommendations for Modifications:	None.

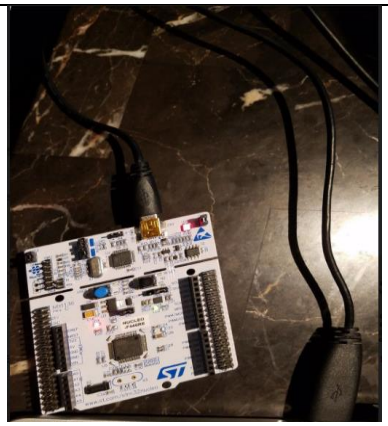
TNC Testing Form (REV1)	
Leaf on the Tree	RX
Device Under Test (Testing Tree Number):	2.1.1.2
Date:	10/31/2020
Person(s) Conducting Experiment:	David Cain
Signature:	
Experiment Purpose:	The purpose of this experiment is to ensure the microcontroller is receiving data over UART.
Experiment Procedure:	To test the data connection over UART, the microcontroller will be given a packet, and this will then printed over serial again.
Equipment Settings / Software Settings (w Revision):	The microcontroller will be running our most current version of software and I will be viewing the output using a serial monitor program called RealTerm 2.0.0.70. I will be feeding packets using the program provided to us by Rizwan.
Testing Diagram / Picture:	
Data Points:	
Pass / Fail:	Pass
Interpreted Notes:	The data being read over the serial monitor seems acceptable for the given packet.
Recommendations for Modifications:	None.

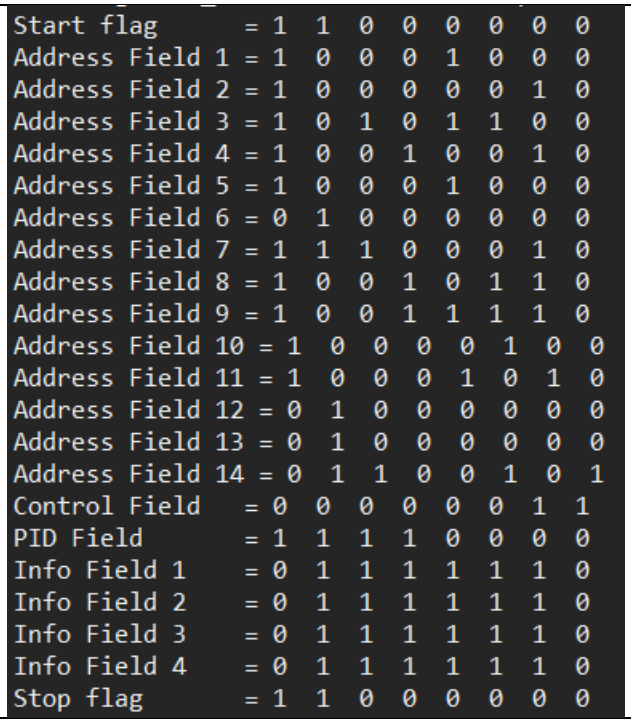
TNC Testing Form (REV1)	
Leaf on the Tree	RX
Device Under Test (Testing Tree Number):	2.1.1.3
Date:	10/31/2020
Person(s) Conducting Experiment:	David Cain
Signature:	
Experiment Purpose:	The purpose of this experiment is to ensure the microcontroller is transmitting data over UART properly.
Experiment Procedure:	To test the data connection over UART, the microcontroller will be given a packet, and this will then printed over serial again. Effectively I am testing bother transmitting and receiving in the same step. If any failure occurs I would simplify this test further.
Equipment Settings / Software Settings (w Revision):	The microcontroller will be running our most current version of software and I will be viewing the output using a serial monitor program called RealTerm 2.0.0.70. I will be feeding packets using the program provided to us by Rizwan.
Testing Diagram / Picture:	
Data Points:	
Pass / Fail:	Pass
Interpreted Notes:	The data being transmitted over the serial monitor seems acceptable for the given packet.
Recommendations for Modifications:	None.

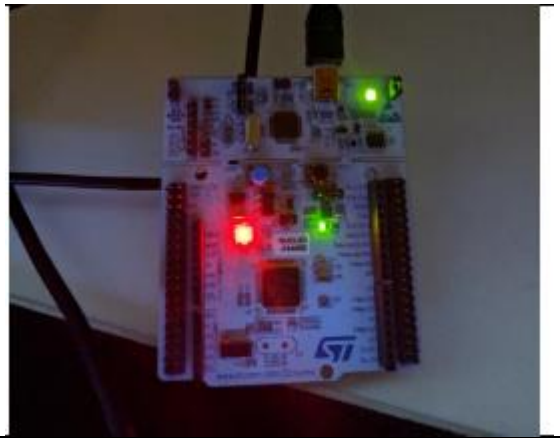
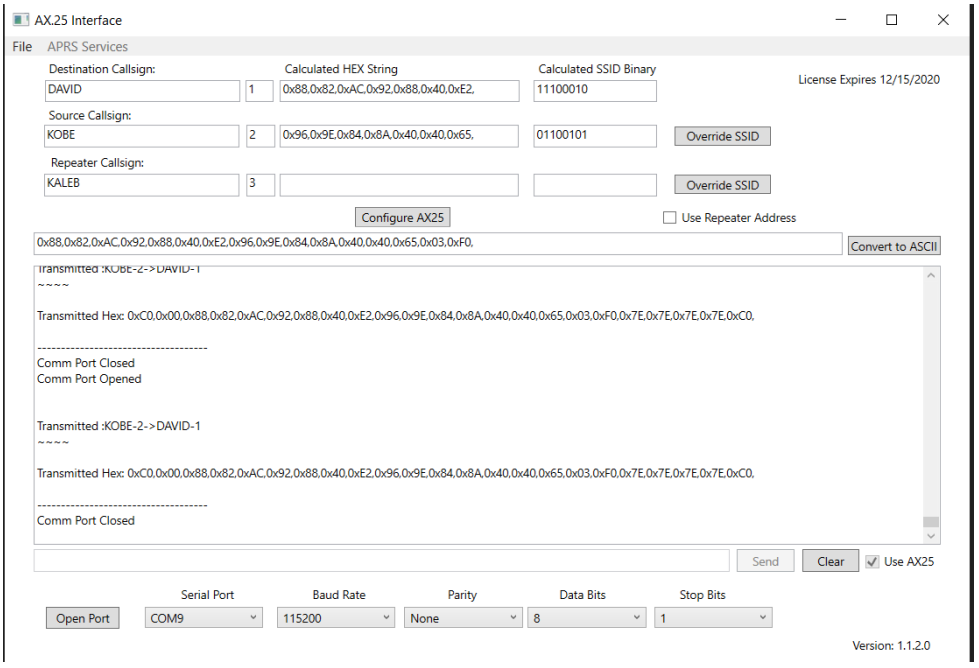
TNC Testing Form (REV1)	
Leaf on the Tree	Serial Communication
Device Under Test (Testing Tree Number):	2.1.1
Date:	11/1/20
Person(s) Conducting Experiment:	Kobe Keopraseuth, Kaleb Leon, David Cain
Signature:	
Experiment Purpose:	To test whether we can send Kiss packets over serial.
Experiment Procedure:	Set up mentor's software to send KISS packet to TNC and then the TNC will send it back over serial and we will see it using realterm
Equipment Settings / Software Settings (w Revision):	Set to communicate over COM1 and store the KISS packet in a buffer and send that buffer over serial using UART
Testing Diagram / Picture:	
Data Points:	

Pass / Fail:	PASS
Interpreted Notes:	Communicates as expected over UART serial.
Recommendations for Modifications:	N/A

TNC Testing Form (REV1)	
Leaf on the Tree	Packet Construction
Device Under Test (Testing Tree Number):	2.1.2
Date:	11/1/20
Person(s) Conducting Experiment:	Kobe Keopraseuth, Kaleb Leon, David Cain
Signature:	
Experiment Purpose:	Testing our mentor's software to make sure it produces the correct KISS packets
Experiment Procedure:	Entering callsigns and data so that it can be made into a packet and then checking that packet to make sure it has the right contents
Equipment Settings / Software Settings (w Revision):	
Testing Diagram / Picture:	
Data Points:	
Pass / Fail:	PASS
Interpreted Notes:	The software produces the desired valid Kiss packet for testing on our software.
Recommendations for Modifications:	N/A

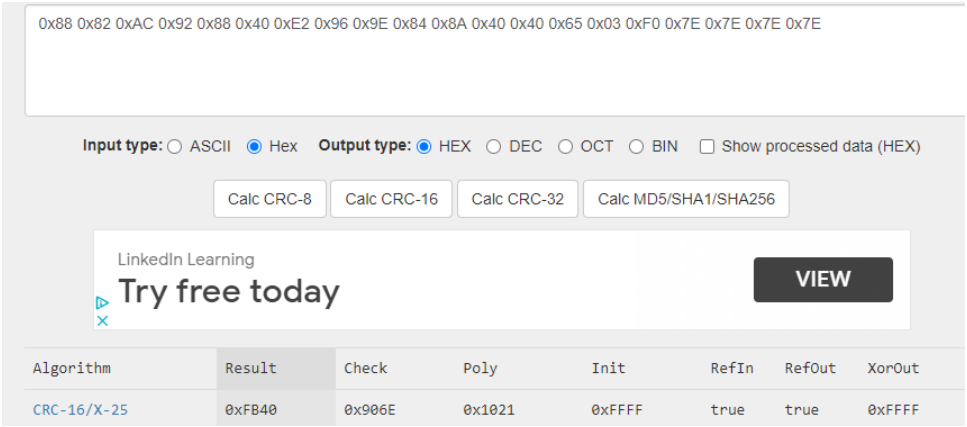
TNC Testing Form (REV1)	
Leaf on the Tree	Data Extraction
Device Under Test (Testing Tree Number):	2.1.3.1
Date:	11/1/20
Person(s) Conducting Experiment:	Kobe Keopraseuth, Kaleb Leon, David Cain
Signature:	
Experiment Purpose:	The purpose of this experiment is to ensure that we were able to extract the received KISS packet over UART and translate that packet into a binary bit stream.
Experiment Procedure:	We will send a packet from our KISS packet generator software and output the binary conversion (done by our microcontroller) through UART which will be displayed on a serial monitor.
Equipment Settings / Software Settings (w Revision):	We will be using Rizwan's given software to generate and send the KISS packet and visual studio's serial monitor to output the data extraction done by our microcontroller.
Testing Diagram / Picture:	 **Setup** Transmitted Hex: 0xC0,0x00,0x88,0x82,0xAC,0x92,0x88,0x40,0xE2,0x96,0x9E,0x84,0x8A,0x40,0x40,0x65,0x03,0xF0,0x7E,0x7E,0x7E,0x7E,0xC0, **input packet**

Data Points:	
Pass / Fail:	Pass
Interpreted Notes:	The binary data being displayed on the serial monitor correctly corresponds to the hexadecimal values sent from Rizwan's software. The second hex value was not displayed, because it is not needed.
Recommendations for Modifications:	None

TNC Testing Form (REV1)	
Leaf on the Tree	AX.25 Protocol
Device Under Test (Testing Tree Number):	2.2
Date:	11/1/20
Person(s) Conducting Experiment:	Kobe Keopraseuth
Signature:	
Experiment Purpose:	The purpose of this experiment of this experiment is to verify that our microcontroller can take in a KISS packet and format the AX.25 Packet Correctly.
Experiment Procedure:	Take in a KISS packet from computer and display the fields of the AX.25 packet. We will also show the calculated crc to show that the KISS packet for properly extracted.
Equipment Settings / Software Settings (w Revision):	Use Rizwan's software to send a KISS packet and display the AX.25 packet on serial monitor.
Testing Diagram / Picture:	
Data Points:	 **Rizwan's software for transmitting KISS packets**

```
Start flag      = 1  1  0  0  0  0  0  0
Address Field 1 = 1  0  0  0  1  0  0  0
Address Field 2 = 1  0  0  0  0  0  1  0
Address Field 3 = 1  0  1  0  1  1  0  0
Address Field 4 = 1  0  0  1  0  0  1  0
Address Field 5 = 1  0  0  0  1  0  0  0
Address Field 6 = 0  1  0  0  0  0  0  0
Address Field 7 = 1  1  1  0  0  0  1  0
Address Field 8 = 1  0  0  1  0  1  1  0
Address Field 9 = 1  0  0  1  1  1  1  0
Address Field 10 = 1  0  0  0  0  1  0  0
Address Field 11 = 1  0  0  0  1  0  1  0
Address Field 12 = 0  1  0  0  0  0  0  0
Address Field 13 = 0  1  0  0  0  0  0  0
Address Field 14 = 0  1  1  0  0  1  0  1
Control Field   = 0  0  0  0  0  0  1  1
PID Field       = 1  1  1  1  0  0  0  0
Info Field 1    = 0  1  1  1  1  1  1  0
Info Field 2    = 0  1  1  1  1  1  1  0
Info Field 3    = 0  1  1  1  1  1  1  0
Info Field 4    = 0  1  1  1  1  1  1  0
Stop flag       = 1  1  0  0  0  0  0  0
```

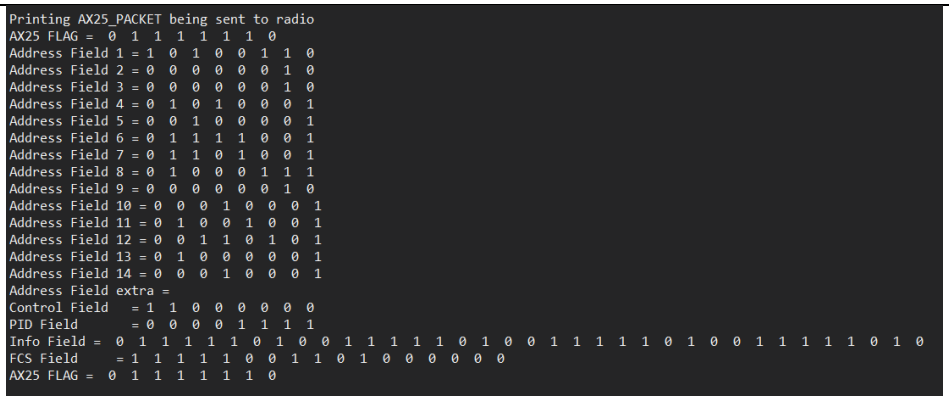
Bitstream output of received KISS packet



Online crc calculation for the KISS packet

```
Convert CRC to FCS (hex) = fb40
```

Microcontroller's crc calculation for KISS packet

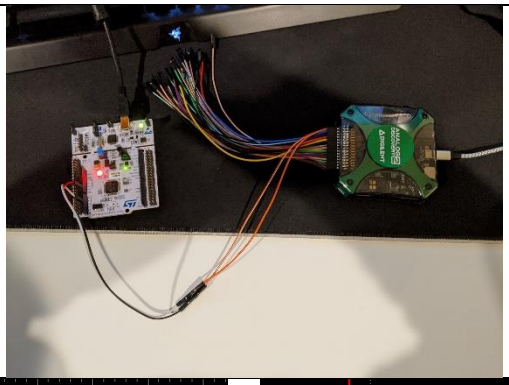
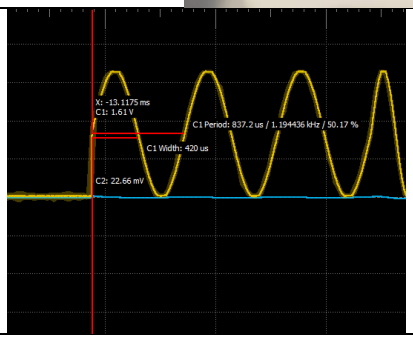
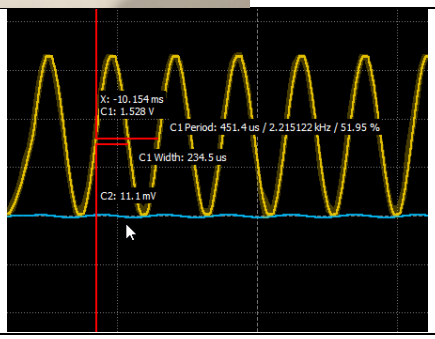
	 **AX.25's bit sequence when sent to radio**
Pass / Fail:	Pass
Interpreted Notes:	Our microcontroller can properly extract a KISS packet and format it into AX.25, along with correctly calculating the crc for the FCS field. It is also able to place the bits to be sent to over radio in the correct order.
Recommendations for Modifications:	None

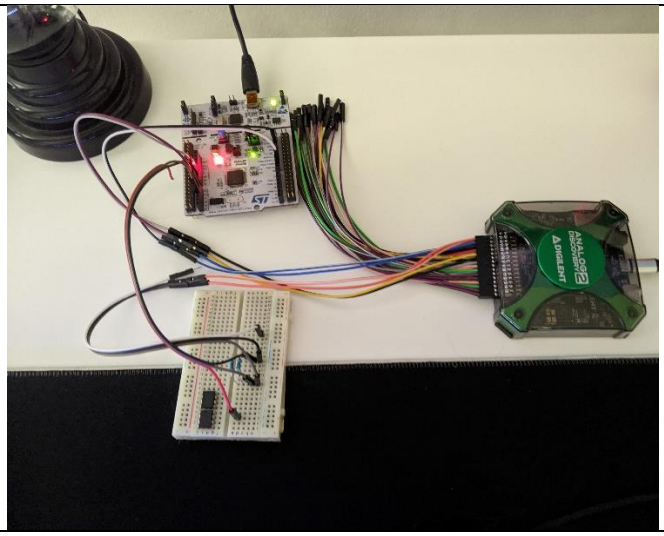
TNC Testing Form (REV1)	
Leaf on the Tree	Validation
Device Under Test (Testing Tree Number):	2.2.2.1.1
Date:	11/1/20
Person(s) Conducting Experiment:	Kobe Keopraseuth
Signature:	
Experiment Purpose:	The purpose of this experiment is to validate that the microcontroller will output the AX.25 (excluding the flags) in correct order.
Experiment Procedure:	I will display how the AX.25 packet will be sent to the radio, using a serial monitor.
Equipment Settings / Software Settings (w Revision):	We will be using Rizwan's software to send a KISS packet over UART, and we will be using visual studio's serial monitor to display the AX.25 packet's bit sequence.
Testing Diagram / Picture:	
Data Points:	Start flag = 1 1 0 0 0 0 0 0 Address Field 1 = 1 0 0 0 1 0 0 0 Address Field 2 = 1 0 0 0 0 0 1 0 Address Field 3 = 1 0 1 0 1 1 0 0 Address Field 4 = 1 0 0 1 0 0 1 0 Address Field 5 = 1 0 0 0 1 0 0 0 Address Field 6 = 0 1 0 0 0 0 0 0 Address Field 7 = 1 1 1 0 0 0 1 0 Address Field 8 = 1 0 0 1 0 1 1 0 Address Field 9 = 1 0 0 1 1 1 1 0 Address Field 10 = 1 0 0 0 0 1 0 0 Address Field 11 = 1 0 0 0 1 0 1 0 Address Field 12 = 0 1 0 0 0 0 0 0 Address Field 13 = 0 1 0 0 0 0 0 0 Address Field 14 = 0 1 1 0 0 1 0 1 Control Field = 0 0 0 0 0 0 1 1 PID Field = 1 1 1 1 0 0 0 0 Info Field 1 = 0 1 1 1 1 1 1 0 Info Field 2 = 0 1 1 1 1 1 1 0 Info Field 3 = 0 1 1 1 1 1 1 0 Info Field 4 = 0 1 1 1 1 1 1 0 Stop flag = 1 1 0 0 0 0 0 0 **Bitnary Bit stream of packet** Printing AX25_PACKET being sent to radio Address Field 1 = 1 0 1 0 0 1 1 0 Address Field 2 = 0 0 0 0 0 0 1 0 Address Field 3 = 0 0 0 0 0 0 1 0 Address Field 4 = 0 1 0 1 0 0 0 1 Address Field 5 = 0 0 1 0 0 0 0 1 Address Field 6 = 0 1 1 1 1 0 0 1 Address Field 7 = 0 1 1 0 1 0 0 1 Address Field 8 = 0 1 0 0 0 1 1 1 Address Field 9 = 0 0 0 0 0 0 1 0 Address Field 10 = 0 0 0 1 0 0 0 1 Address Field 11 = 0 1 0 0 1 0 0 1 Address Field 12 = 0 0 1 1 0 1 0 1 Address Field 13 = 0 1 0 0 0 0 0 1 Address Field 14 = 0 0 0 1 0 0 0 1 Address Field extra = Control Field = 1 1 0 0 0 0 0 0 PID Field = 0 0 0 0 1 1 1 1 Info Field = 0 1 1 1 1 1 0 1 0 0 1 1 1 1 1 0 1 0 0 1 1 1 1 1 0 1 0 FCS Field = 1 1 1 1 1 0 0 1 1 0 1 0 0 0 0 0 0 **Binary Bitstream of how AX.25 packet will be sent to radio**

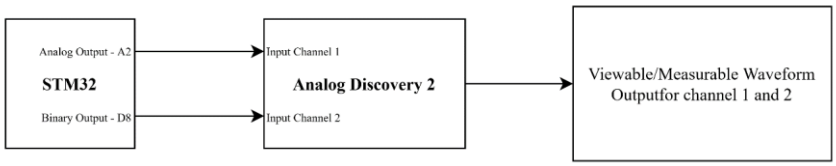
Pass / Fail:	Pass
Interpreted Notes:	As shown on the serial monitor, the AX.25's bits are in the correct order. FCS field is sent MSB first and other fields are sent LSB first. After 5 contiguous ones then a bit stuffed zero is added after.
Recommendations for Modifications:	None

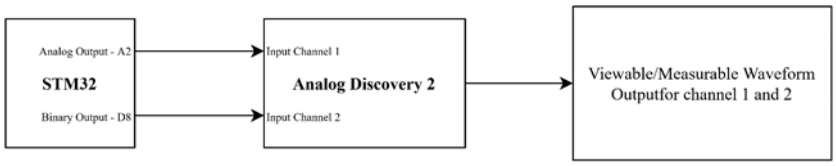
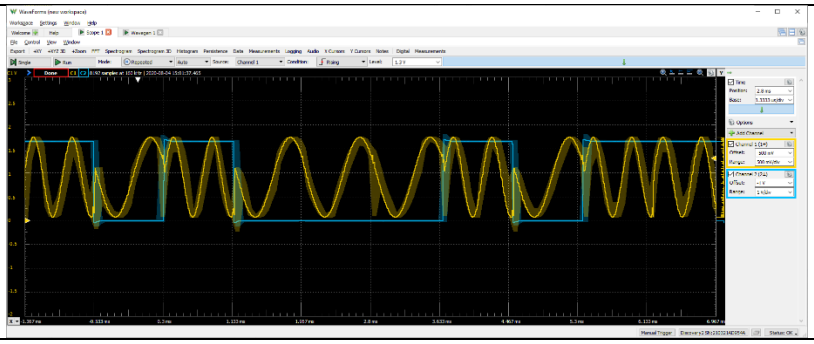
TNC Testing Form (REV1)	
Leaf on the Tree	Packet Construction
Device Under Test (Testing Tree Number):	2.2.2.1
Date:	11/1/20
Person(s) Conducting Experiment:	Kobe Keopraseuth
Signature:	
Experiment Purpose:	The purpose of this experiment is to validate that the microcontroller will output the AX.25 (excluding the flags) in correct order.
Experiment Procedure:	I will display how the AX.25 packet will be sent to the radio, using a serial monitor.
Equipment Settings / Software Settings (w Revision):	We will be using Rizwan's software to send a KISS packet over UART, and we will be using visual studio's serial monitor to display the AX.25 packet's bit sequence.
Testing Diagram / Picture:	
Data Points:	Start flag = 1 1 0 0 0 0 0 0 Address Field 1 = 1 0 0 0 1 0 0 0 Address Field 2 = 1 0 0 0 0 0 1 0 Address Field 3 = 1 0 1 0 1 1 0 0 Address Field 4 = 1 0 0 1 0 0 1 0 Address Field 5 = 1 0 0 0 1 0 0 0 Address Field 6 = 0 1 0 0 0 0 0 0 Address Field 7 = 1 1 1 0 0 0 1 0 Address Field 8 = 1 0 0 1 0 1 1 0 Address Field 9 = 1 0 0 1 1 1 1 0 Address Field 10 = 1 0 0 0 0 1 0 0 Address Field 11 = 1 0 0 0 1 0 1 0 Address Field 12 = 0 1 0 0 0 0 0 0 Address Field 13 = 0 1 0 0 0 0 0 0 Address Field 14 = 0 1 1 0 0 1 0 1 Control Field = 0 0 0 0 0 0 1 1 PID Field = 1 1 1 1 0 0 0 0 Info Field 1 = 0 1 1 1 1 1 1 0 Info Field 2 = 0 1 1 1 1 1 1 0 Info Field 3 = 0 1 1 1 1 1 1 0 Info Field 4 = 0 1 1 1 1 1 1 0 Stop flag = 1 1 0 0 0 0 0 0 **Bitnary Bit stream of packet** Printing AX25_PACKET being sent to radio Address Field 1 = 1 0 1 0 0 1 1 0 Address Field 2 = 0 0 0 0 0 0 1 0 Address Field 3 = 0 0 0 0 0 0 1 0 Address Field 4 = 0 1 0 1 0 0 0 1 Address Field 5 = 0 0 1 0 0 0 0 1 Address Field 6 = 0 1 1 1 1 0 0 1 Address Field 7 = 0 1 1 0 1 0 0 1 Address Field 8 = 0 1 0 0 0 1 1 1 Address Field 9 = 0 0 0 0 0 0 1 0 Address Field 10 = 0 0 0 1 0 0 0 1 Address Field 11 = 0 1 0 0 1 0 0 1 Address Field 12 = 0 0 1 1 0 1 0 1 Address Field 13 = 0 1 0 0 0 0 0 1 Address Field 14 = 0 0 0 1 0 0 0 1 Address Field extra = Control Field = 1 1 0 0 0 0 0 0 PID Field = 0 0 0 0 1 1 1 1 Info Field = 0 1 1 1 1 1 0 1 0 0 1 1 1 1 1 0 1 0 0 1 1 1 1 1 0 1 0 FCS Field = 1 1 1 1 1 0 0 1 1 0 1 0 0 0 0 0 0 **Binary Bitstream of how AX.25 packet will be sent to radio**

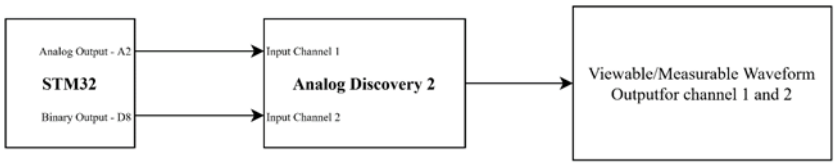
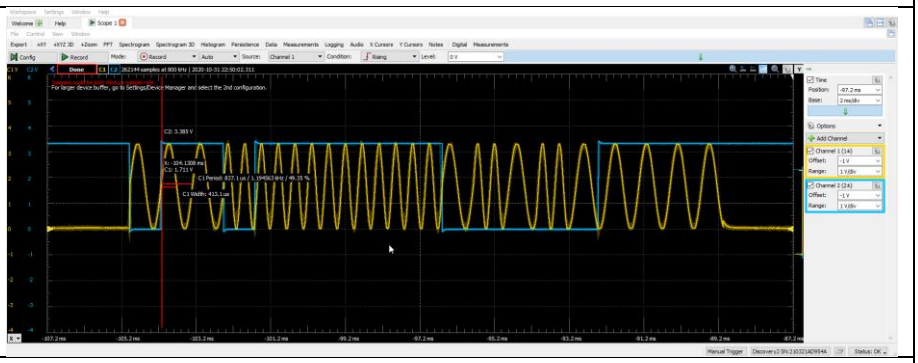
Pass / Fail:	Pass
Interpreted Notes:	As shown on the serial monitor, the AX.25's bits are in the correct order. FCS field is sent MSB first and other fields are sent LSB first. After 5 contiguous ones then a bit stuffed zero is added after.
Recommendations for Modifications:	None

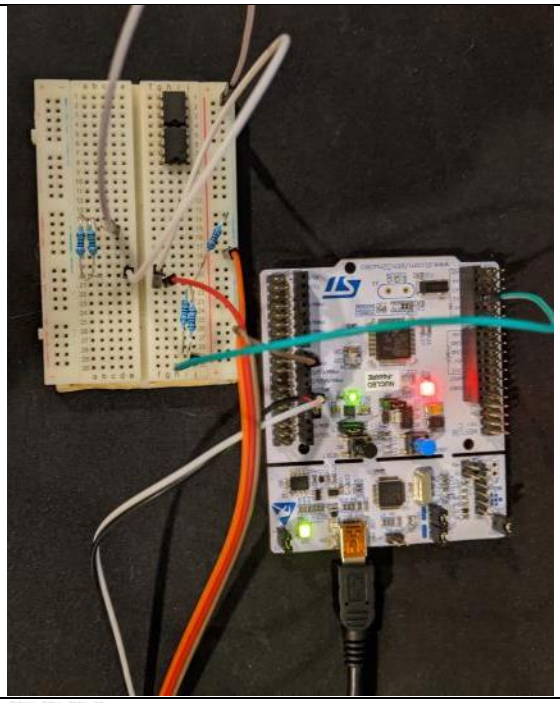
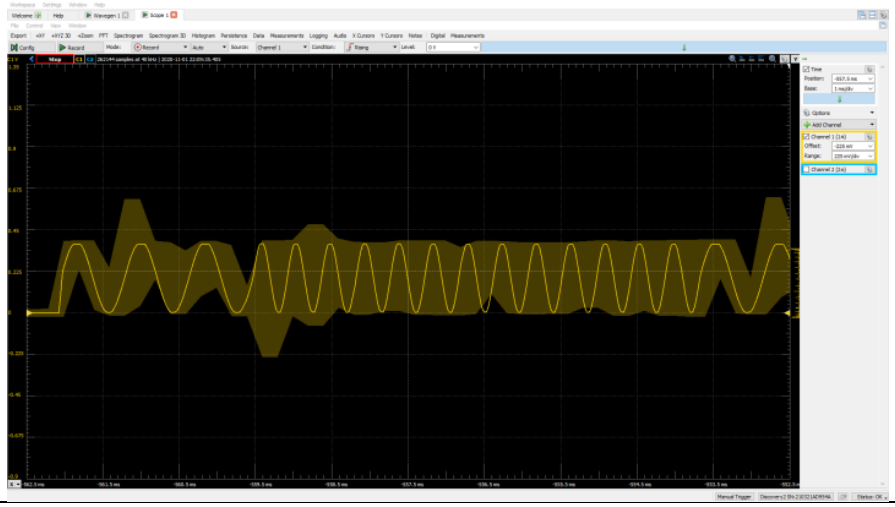
TNC Testing Form (REV1)	
Leaf on the Tree	Frequency
Device Under Test (Testing Tree Number):	2.3.2.1.1
Date:	10/31/2020
Person(s) Conducting Experiment:	David Cain
Signature:	
Experiment Purpose:	Ensure that the output frequencies of the microcontroller are 1200 and 2200 when expected.
Experiment Procedure:	Force the TNC into a debugging broadcast mode, then use the Digilent discovery 2 to measure the waveform frequency at many points.
Equipment Settings / Software Settings (w Revision):	The Digilent will be set to record the frequency of the waveform measured. For general insight, the RMS and minimum were also recorded
Testing Diagram / Picture:	
Data Points:	 
Pass / Fail:	Pass
Interpreted Notes:	As shown in the datapoints, the output waveforms are 1200Hz and 2200Hz as expected.
Recommendations for Modifications:	None.

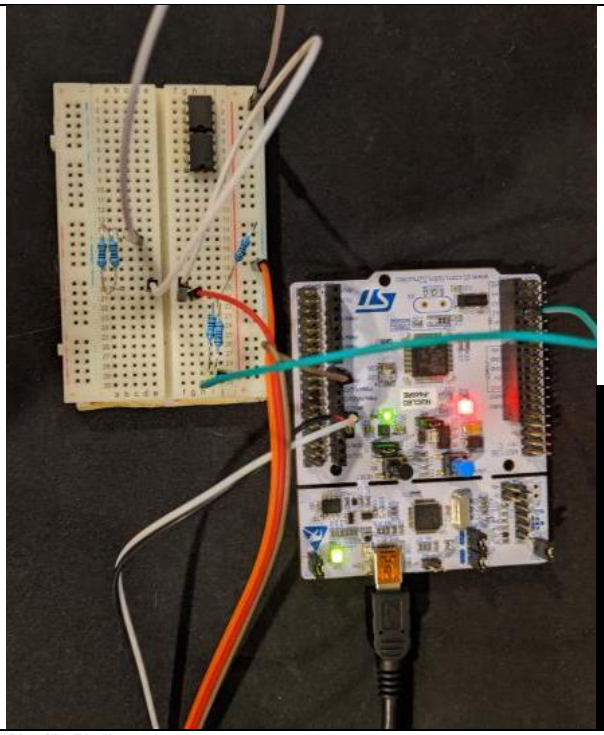
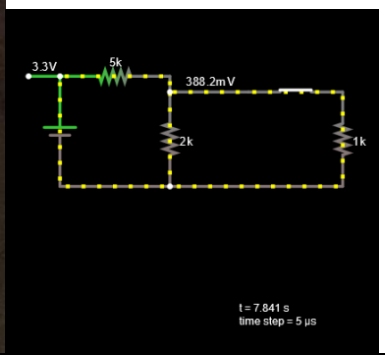
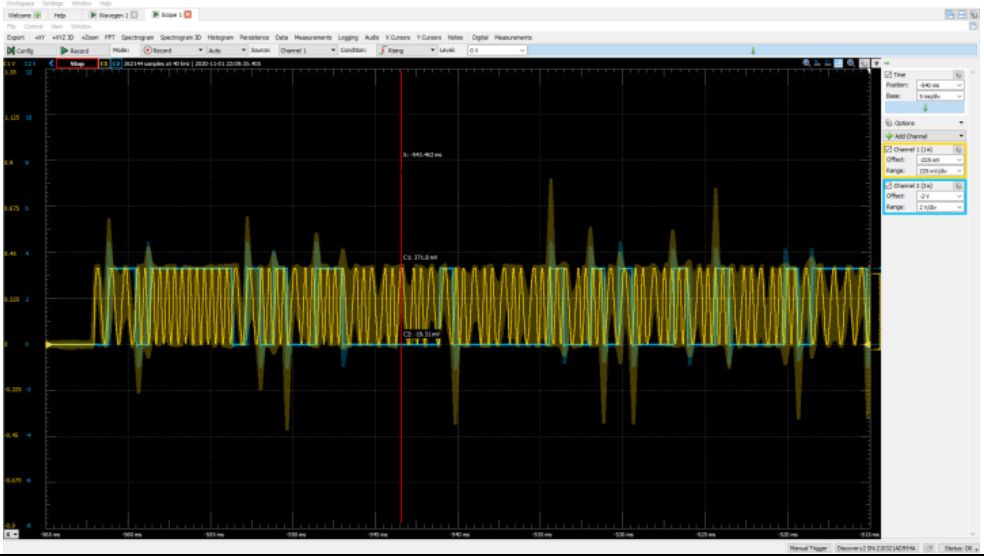
TNC Testing Form (REV1)	
Leaf on the Tree	Amplitude
Device Under Test (Testing Tree Number):	2.3.2.1.2.1
Date:	10/4/2020
Person(s) Conducting Experiment:	David Cain
Signature:	
Experiment Purpose:	The purpose of this experiment is to measure waveform output voltage. Part of our specifications it to be 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

TNC Testing Form (REV1)	
Leaf on the Tree	Baud
Device Under Test (Testing Tree Number):	2.3.2.1.3.1
Date:	10/4/2020
Person(s) Conducting Experiment:	David Cain
Signature:	
Experiment Purpose:	The purpose of this experiment is to measure and ensure the number of signaling events per second (or baud rate) is correctly established as 1200Hz
Experiment Procedure:	To verify the baud rate, a diagnostic signal will be enabled in software to output the current transmission bit value represented in binary. This binary wave form can easily have baud rate measured.
Equipment Settings / Software Settings (w Revision):	Analog Discovery 2 input channel 1 and 2 will be connected to the STM32 output pins D8(PA9) and A2(PA4)
Testing Diagram / Picture:	
Data Points:	
Pass / Fail:	Pass
Interpreted Notes:	Waveform is sustaining a baud rate of 1200Hz. This was tested with multiple wave forms but easily viewed with alternating bit pattern.
Recommendations for Modifications:	None

TNC Testing Form (REV1)	
Leaf on the Tree	Bit Transitions
Device Under Test (Testing Tree Number):	2.3.2.1.4
Date:	10/31/2020
Person(s) Conducting Experiment:	David Cain
Signature:	
Experiment Purpose:	The purpose of this experiment is to ensure that the waveforms of our output do not suffer due to bit transitions.
Experiment Procedure:	Force the TNC into a debugging broadcast mode, then use the Digilent discovery 2 to measure the waveform frequency at many points.
Equipment Settings / Software Settings (w Revision):	The Digilent will be set to record the waveform and an optical inspection will be used.
Testing Diagram / Picture:	
Data Points:	
Pass / Fail:	Fail
Interpreted Notes:	There is an obvious phase shift when the bits are transitioning.
Recommendations for Modifications:	Correct code that generates the output to calculate the next expected starting point of each bit.

TNC Testing Form (REV1)	
Leaf on the Tree	Bit Transitions
Device Under Test (Testing Tree Number):	2.3.2.1.4
Date:	10/31/2020
Person(s) Conducting Experiment:	David Cain
Signature:	
Experiment Purpose:	The purpose of this experiment is to ensure that the waveforms of our output do not suffer due to bit transitions.
Experiment Procedure:	Force the TNC into a debugging broadcast mode, then use the Digilent discovery 2 to measure the waveform frequency at many points.
Equipment Settings / Software Settings (w Revision):	The Digilent will be set to record the waveform and an optical inspection will be used.
Testing Diagram / Picture:	
Data Points:	
Pass / Fail:	Pass
Interpreted Notes:	The waveform is very continuous. You will notice the change in frequency when the digital value is 0. This is due to the NRZI encoding scheme, but the phase is continuous.
Recommendations for Modifications:	None.

TNC Testing Form (REV1)	
Leaf on the Tree	Signal Production
Device Under Test (Testing Tree Number):	2.3.2.1
Date:	11/1/20
Person(s) Conducting Experiment:	Kobe Keopraseuth, Kaleb Leon, David Cain
Signature:	
Experiment Purpose:	To show we can produce a basic signal following the coded frequency, amplitude, and baud rate.
Experiment Procedure:	Hard coded parameters for our signal, then produced by our Nucleo. It will be read and judged using the analog discovery.
Equipment Settings / Software Settings (w Revision):	Frequency shift between 1200Hz and 2200Hz Amplitude of 400mV peak to peak Baud rate of 2200
Testing Diagram / Picture:	
Data Points:	
Pass / Fail:	Pass
Interpreted Notes:	Produces a valid signal
Recommendations for Modifications:	N/A

TNC Testing Form (REV1)	
Leaf on the Tree	DAC
Device Under Test (Testing Tree Number):	2.3.2
Date:	11/1/20
Person(s) Conducting Experiment:	Kobe Keopraseuth, Kaleb Leon, David Cain
Signature:	
Experiment Purpose:	Test the entire DAC subsystem with an actual AX.25 packet actually used in our project.
Experiment Procedure:	Feed a known accurate AX.25 packet then use code to covert that into an analog output signal.
Equipment Settings / Software Settings (w Revision):	Current code base for DAC Sinewave.
Testing Diagram / Picture:	 
Data Points:	
Pass / Fail:	Pass

Interpreted Notes:	Meets the accuracy we required of the output analog signal.
Recommendations for Modifications:	N/A

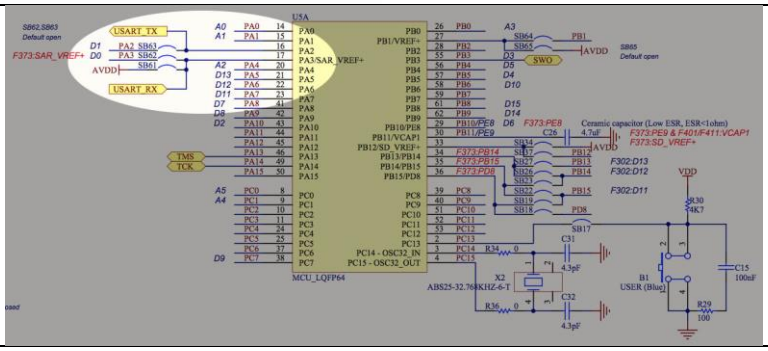
<table>
<tr><td colspan="2">TNC Testing Form (REV1)</td></tr>
<tr><td>Leaf on the Tree</td><td>1.1.1.1</td></tr>
<tr><td>Device Under Test (Testing Tree Number):</td><td>USB Cable</td></tr>
<tr><td>Date:</td><td>10/31/2020</td></tr>
<tr><td>Person(s) Conducting Experiment:</td><td>Kaleb Leon</td></tr>
<tr><td>Signature:</td><td>Kaleb L</td></tr>
<tr><td>Experiment Purpose:</td><td>Functionality of USB-B mini works in communicating with the PC over serial.</td></tr>
<tr><td>Experiment Procedure:</td><td>Plug Nucleo into PC via the USB-B mini cable and run some code to do serial communication and show we can upload code via this cable as well.</td></tr>
<tr><td>Equipment Settings / Software Settings (w Revision):</td><td>Nucleo setup on table connected to USB port on PC using the USB-B mini cable
<u>Software Settings:</u>
HARDWARE INIT FILE
<pre>
void
configure_system_clock(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct;
    RCC_ClkInitTypeDef RCC_ClkInitStruct;

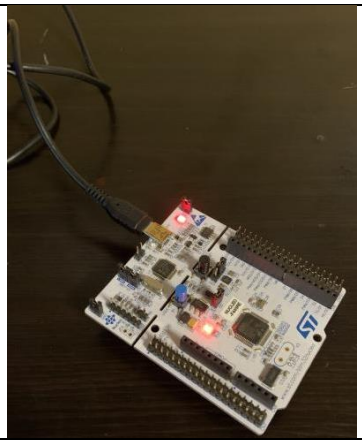
    __PWR_CLK_ENABLE();

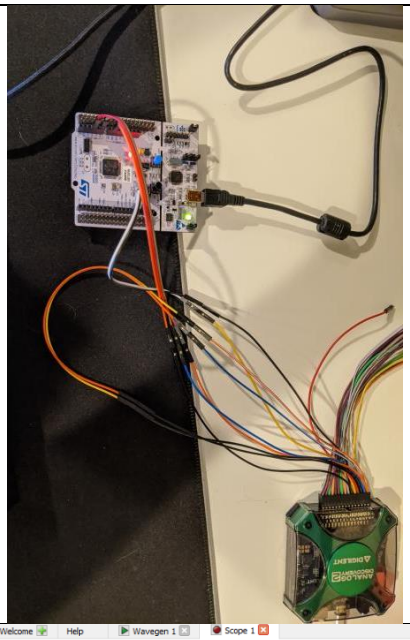
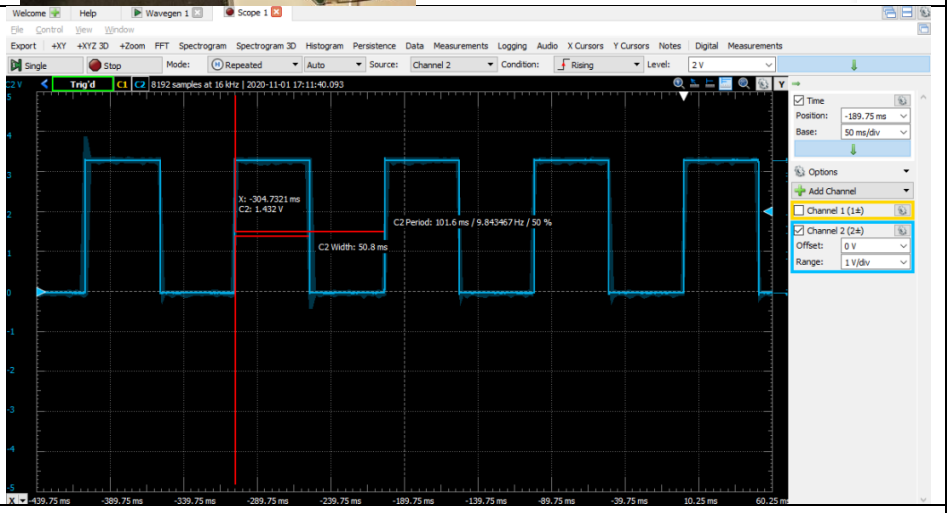
    __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE2);

    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
    RCC_OscInitStruct.HSIState = RCC_HSI_ON;
    RCC_OscInitStruct.HSICalibrationValue = 6;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
    RCC_OscInitStruct.PLL.PLLM = 16;
    RCC_OscInitStruct.PLL.PLLN = 336;
    RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
    RCC_OscInitStruct.PLL.PLLQ = 7;
    HAL_RCC_OscConfig(&RCC_OscInitStruct);

    RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_SYSCLK|RCC_CLOCKTYPE_PCLK1;
    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
    RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
    HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2);
}
</pre>
UART CONFIG
<pre>
UART_HandleTypeDef huart2;
...
void MX_USART2_UART_Init(void)
{
    huart2.Instance = USART2;
    huart2.Init.BaudRate = 115200;
    huart2.Init.WordLength = UART_WORDLENGTH_8B;
    huart2.Init.StopBits = UART_STOPBITS_1;
    huart2.Init.Parity = UART_PARITY_NONE;
    huart2.Init.Mode = UART_MODE_TX_RX;
    huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    HAL_UART_Init(&huart2);
}
</pre>
MAIN CODE
<pre>
int main(int argc, char* argv[])
{
  char *msg = "Hello Nucleo Fun!\n\r";

  HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), 0xFFFF);
  while(1);
}
</pre></td></tr>
<tr><td>Testing Diagram / Picture:</td><td>  </td></tr>
</table>

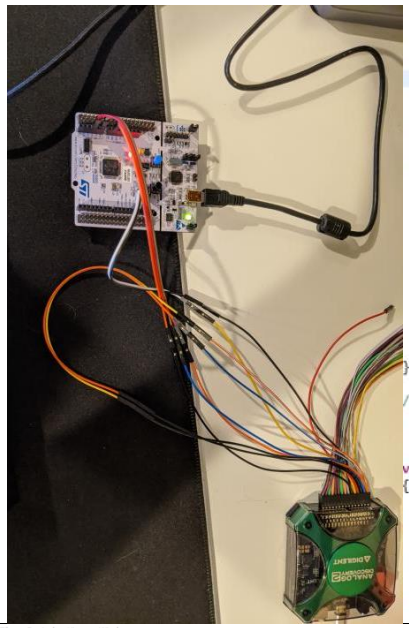
	
Data Points:	Serial Output Type the escape character followed by C to get back, or followed by ? to see other options. --- Hello Nucleo Fun! Uploading Code Memory Programming ... Opening and parsing file: ST-LINK_GDB_server_a08060.srec File : ST-LINK_GDB_server_a08060.srec Size : 20172 Bytes Address : 0x08000000 Erasing memory corresponding to segment 0: Erasing internal memory sectors [0 1] Download in Progress: File download complete Time elapsed during download operation: 00:00:00.708 Verifying ... Download verified successfully Debugger connection lost. Shutting down...
Pass / Fail:	PASS
Interpreted Notes:	The Nucleo seems to be communicated fine with the PC over USB-B mini
Recommendations for Modifications:	N/A

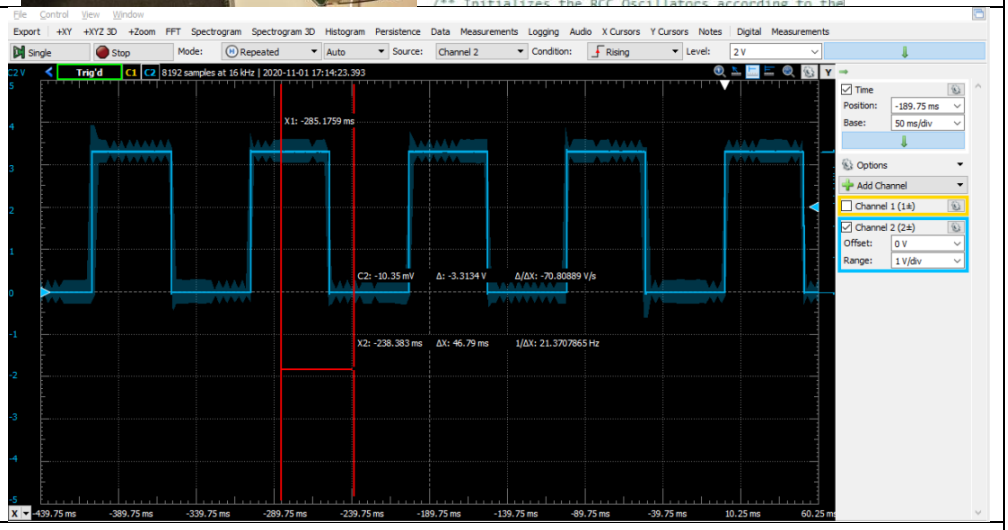
TNC Testing Form (REV1)	
Leaf on the Tree	I/O Pins
Device Under Test (Testing Tree Number):	1.1.2.3
Date:	11/1/20
Person(s) Conducting Experiment:	Kobe Keopraseuth, Kaleb Leon, David Cain
Signature:	
Experiment Purpose:	The purpose of this experiment is to test to make sure the I/O pins on the microcontroller are functioning to our standard. Working meaning able to read input and output accurately.
Experiment Procedure:	We will send a square wave analog signal from our microcontroller by waiting a set number of milliseconds and changing the GPIO pin to output that 1 or 0. This is done on each pin we use in the system. This shows we can output and receive signals correctly on the pins.
Equipment Settings / Software Settings (w Revision):	We use the nucleo board with the code shown below to produce the signal. Then we read the signal using our analog discovery shown on channel 2 in the data points.
Testing Diagram / Picture:	
Data Points:	
Pass / Fail:	Pass
Interpreted Notes:	The signal being displayed is correct because the delay between transitions is the same as defined in the code.

```
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_TIM3_Init();
  /* USER CODE BEGIN 2 */
  HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_1);
  uint32_t captureVal;
  uint32_t time = HAL_GetTick();
  uint32_t millis_wait = 50;
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

      if(HAL_GetTick() - time > millis_wait){
          HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_0);
          time = HAL_GetTick();
      }
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output vo
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTA
  /** Initializes the RCC Oscillators according to th
  * in the RCC_OscInitTypeDef structure.
  */
```

Recommendations for Modifications:	None

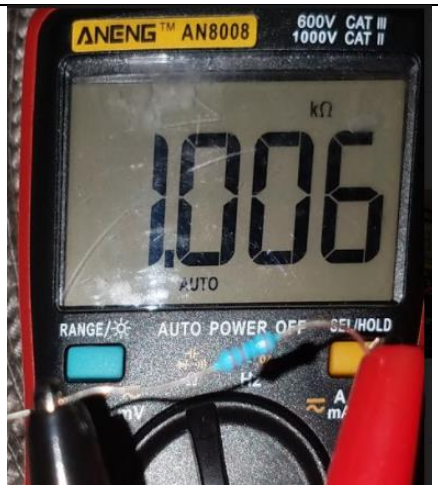
TNC Testing Form (REV1)	
Leaf on the Tree	Validate Signal Voltages
Device Under Test (Testing Tree Number):	1.1.2.4
Date:	11/1/20
Person(s) Conducting Experiment:	Kobe Keopraseuth, Kaleb Leon, David Cain
Signature:	
Experiment Purpose:	The purpose of this experiment is to test to make sure the voltages are accurate for our analog signal on the microcontroller.
Experiment Procedure:	We will send a square wave analog signal from the Nucleo, that is generated by waiting a number of milliseconds and change the GPIO pin to output a 0 or 1. We set the amplitude to a specific value to be measured in our scope analog discovery.
Equipment Settings / Software Settings (w Revision):	We use the nucleo board with the code shown below to produce the signal. Then we read the signal using our analog discovery shown on channel 2 in the data points.

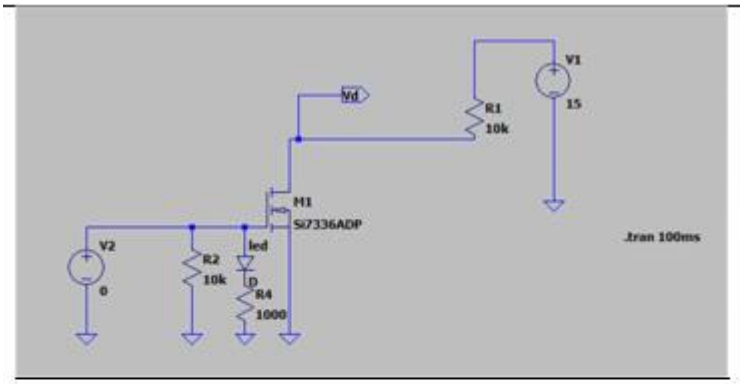
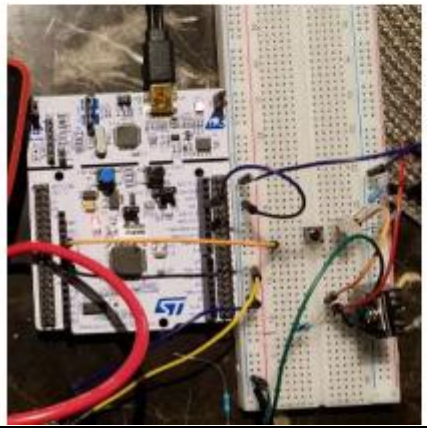
Testing Diagram / Picture:	
	/* Initialize all configured peripherals */
	MX_GPIO_Init();
	MX_USART2_UART_Init();
	MX_TIM3_Init();
	/* USER CODE BEGIN 2 */
	HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_1);
	uint32_t captureVal;
	uint32_t time = HAL_GetTick();
	uint32_t millis_wait = 50;
	/* USER CODE END 2 */
	
	/* Infinite loop */
	/* USER CODE BEGIN WHILE */
	while (1)
	{
	/* USER CODE END WHILE */
	
	/* USER CODE BEGIN 3 */
	
	if(HAL_GetTick() - time > millis_wait){
	HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_0);
	time = HAL_GetTick();
	}
	}
	/* USER CODE END 3 */
	}
	
	/**
	* @brief System Clock Configuration
	* @retval None
	*/
	void SystemClock_Config(void)
	{
	RCC_OscInitTypeDef RCC_OscInitStruct = {0};
	RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
	
	/** Configure the main internal regulator output vol
	*/
	__HAL_RCC_PWR_CLK_ENABLE();
	__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAG
	/** Initializes the RCC Oscillators according to the

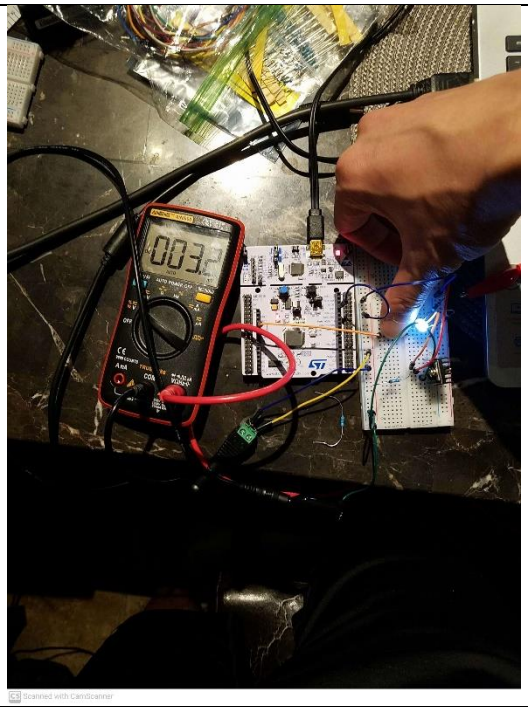
Data Points:	 X1: -285.1759 ms C2: -10.35 mV Δ: -3.3134 V Δ/ΔX: -70.80889 V/s X2: -238.383 ms ΔX: 46.79 ms 1/ΔX: 21.3707865 Hz
Pass / Fail:	Pass

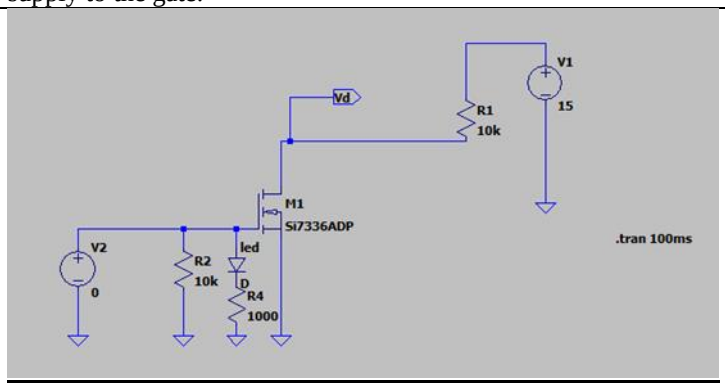
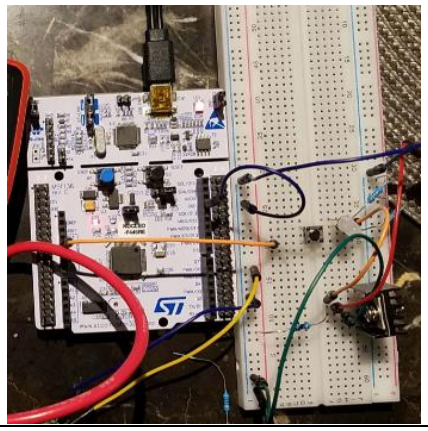
Interpreted Notes:	The signal being displayed is correct because the amplitude of the wave meets the input set in the Nucleo code.
Recommendations for Modifications:	N/A

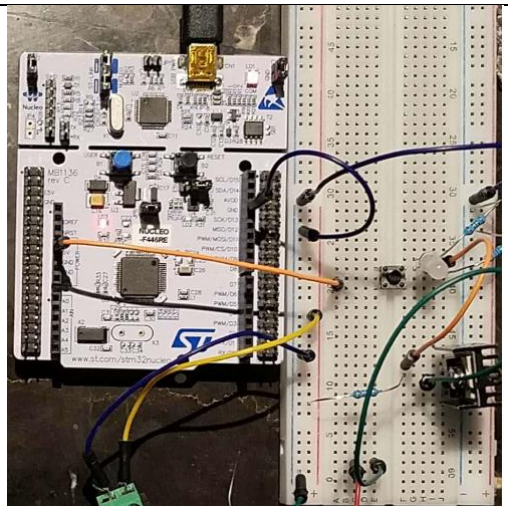
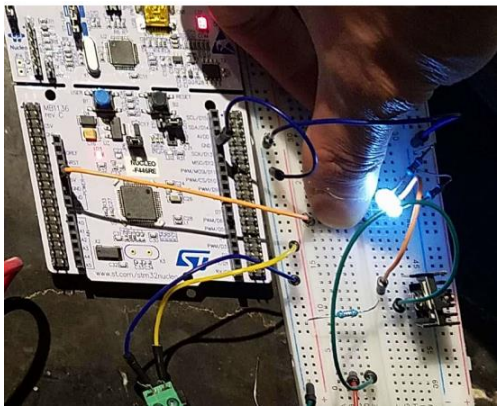
TNC Testing Form (REV1)	
Leaf on the Tree	Resistors
Device Under Test (Testing Tree Number):	1.2.1.1
Date:	11/1/20
Person(s) Conducting Experiment:	Kobe Keopraseuth
Signature:	
Experiment Purpose:	The purpose of this experiment is to ensure that the resistors used for the PTT circuit are within the proper tolerance, which is in between 5 percent over or under their nominal value.
Experiment Procedure:	Use a voltmeter to measure actual resistance of the 10k and 1k ohm resistors connected to the MOSFET's gate. The 10k passes if the measured value is in between 10.5k – 9.5k ohms. The 1k passes if the measure value is in between 1.5k - .95k ohms.
Equipment Settings / Software Settings (w Revision):	Use an Aneng multimeter to measure each resistor's resistance.
Testing Diagram / Picture:	
Data Points:	 **10k ohm**

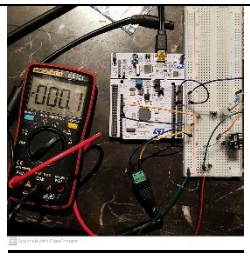
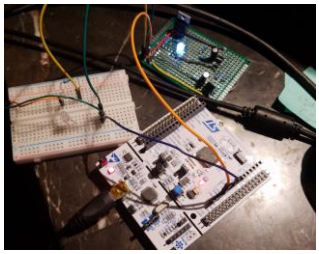
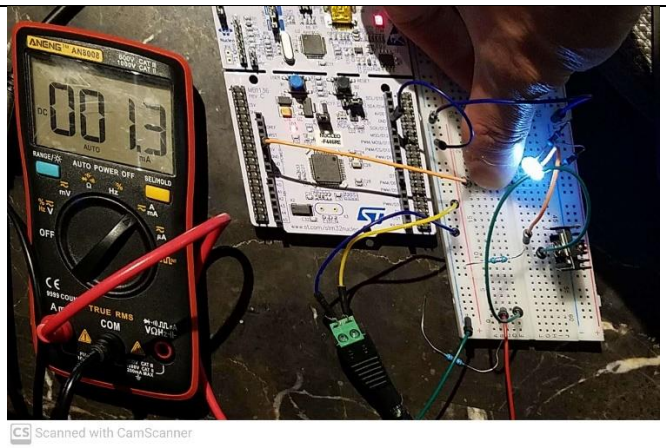
	 **1k ohm**
Pass / Fail:	Pass
Interpreted Notes:	As can be seen both resistors pass since they are with in the desired range.
Recommendations for Modifications:	None

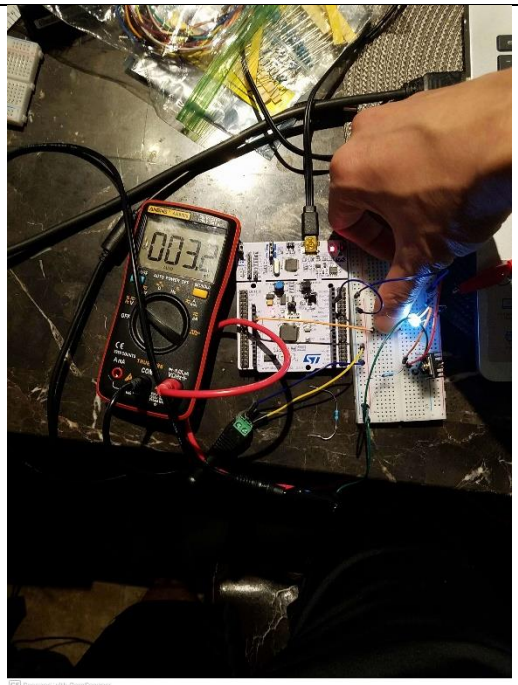
TNC Testing Form (REV1)	
Leaf on the Tree	PTT Circuit
Device Under Test (Testing Tree Number):	1.2.1
Date:	11/1/20
Person(s) Conducting Experiment:	Kobe Keopraseuth
Signature:	
Experiment Purpose:	The purpose of this experiment is to verify that the PTT circuit can pull 15V ,going into the drain, to 0 V when the it is turned on.
Experiment Procedure:	We will implement the circuit shown below and input 15 V with a pull-up resistor, to act as the radio's 15 V. Then we will use a tactile switch to switch the PTT circuit on and measure the voltage across the drain to source to see.
Equipment Settings / Software Settings (w Revision):	We use a breadboard to hook up the circuit shown below and a dc power supply for the 15 V. We used LTspice for designing the circuit. We use 3.3V reference to supply to the gate. Also, we will use a voltmeter to measure the drain to source voltage.
Testing Diagram / Picture:	 **Circuit** 

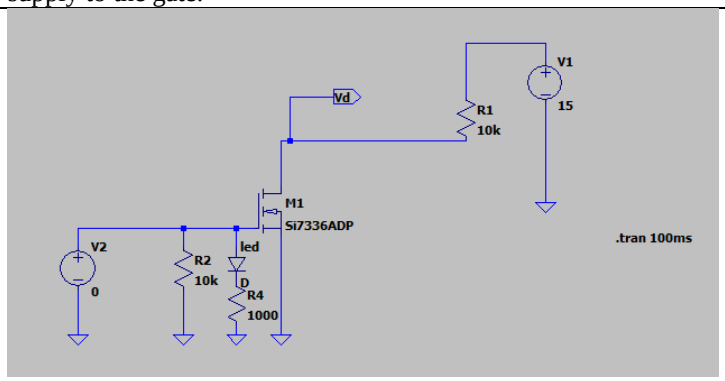
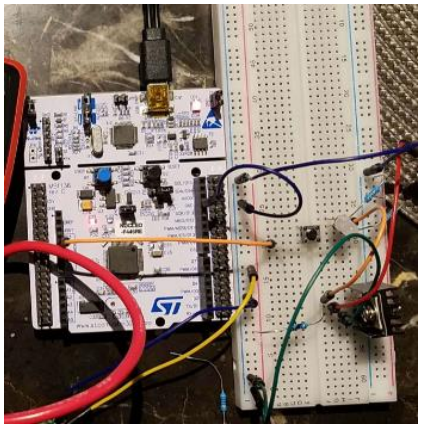
Data Points:	
Pass / Fail:	Pass
Interpreted Notes:	As shown, when a high signal is inputted into the circuit, then it is able to decrease the 15V at the drain down to 3.2 mV which very close to 0 V
Recommendations for Modifications:	None

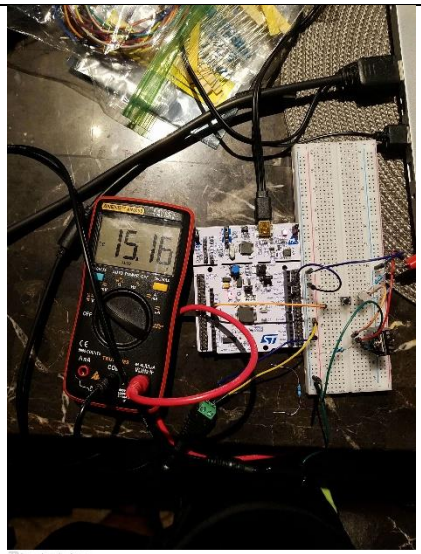
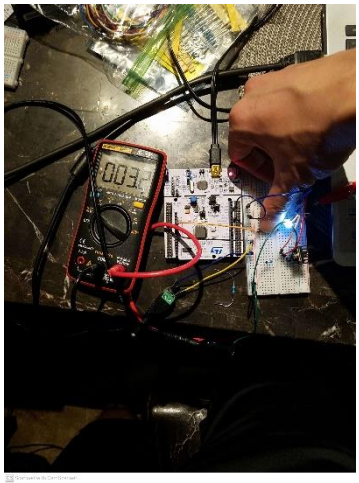
TNC Testing Form (REV1)	
Leaf on the Tree	1.2.2.1
Device Under Test (Testing Tree Number):	LED
Date:	11/1/20
Person(s) Conducting Experiment:	Kobe Keopraseuth
Signature:	
Experiment Purpose:	The purpose of this experiment is to verify the LED, associated with our PTT circuit is turned on, when the MOSFET switches on.
Experiment Procedure:	We will implement the circuit shown below and input 15 V with a pull-up resistor, to act as the radio's 15 V. Then we will use a tactile switch to switch the MOSFET on and off, which should also turn the LED on and off respectively.
Equipment Settings / Software Settings (w Revision):	We use a breadboard to hook up the circuit shown below and a dc power supply for the 15 V. We used LTspice for designing the circuit. We use 3.3V reference to supply to the gate.
Testing Diagram / Picture:	 **Circuit** 

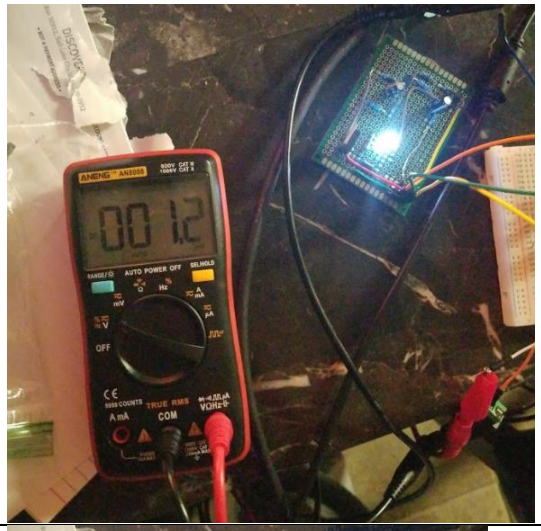
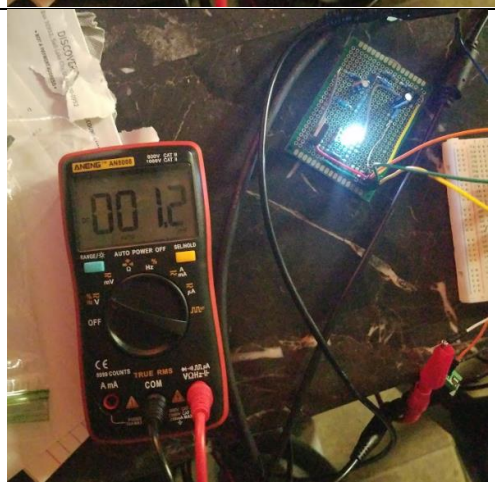
Data Points:	 **LED off**  **LED on**
Pass / Fail:	Pass
Interpreted Notes:	When the MOSFET has 3.3 V inputted into the gate, then the LED turns on. When the MOSFET has 0 V inputted into the gate, then the LED turns off.
Recommendations for Modifications:	None

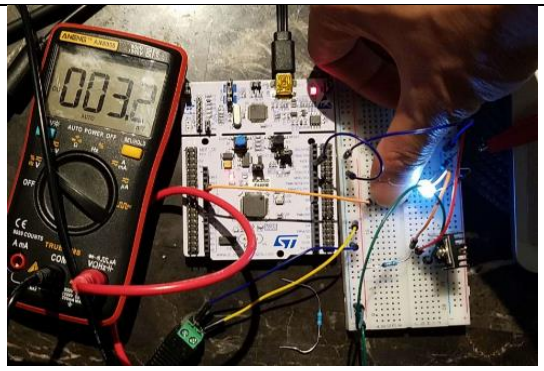
TNC Testing Form (REV1)	
Leaf on the Tree	Power Consumption
Device Under Test (Testing Tree Number):	1.2.3.1
Date:	1/11/20
Person(s) Conducting Experiment:	Kobe Keopraseuth
Signature:	
Experiment Purpose:	The purpose of this experiment was to measure the power consumed by the PTT circuit when turned on. Also, to see how stable the MOSFET's internal temperature is.
Experiment Procedure:	I used the soldered circuit on a perf board with 3.3V supplied to gate from my microcontroller and 15 V DC supply with a 10k ohm pull-up resistor connected to drain. Used a multimeter to measure current and voltage across drain to source.
Equipment Settings / Software Settings (w Revision):	Used a stm32 for the 3.3V DC supply and an external 15 V supply. Used a multimeter to measure current and voltage across drain to source.
Testing Diagram / Picture:	 
Data Points:	

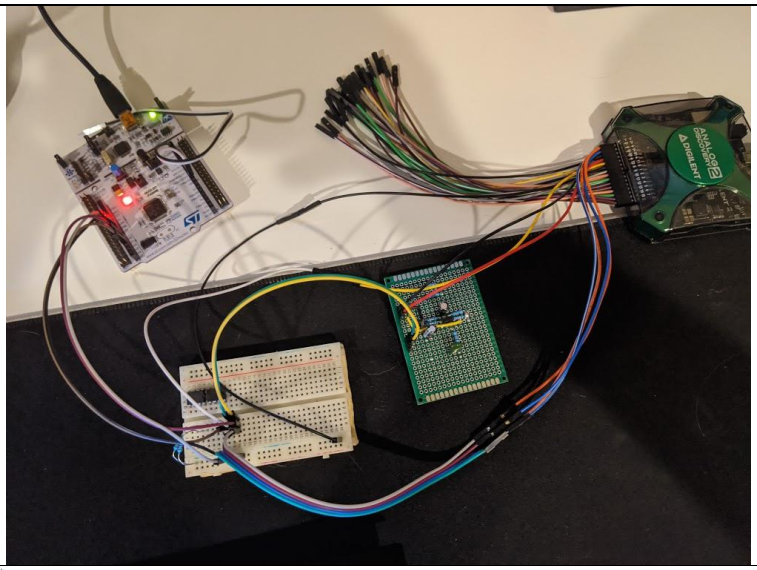
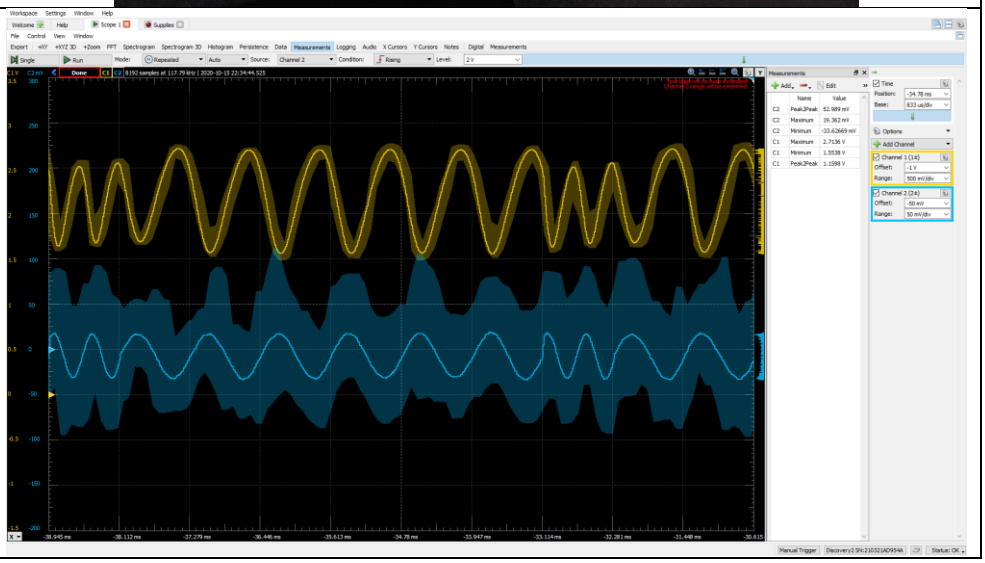
	
Pass / Fail:	Pass
Interpreted Notes:	The calculated Power consumption was 4.16 uW, which barely consumes any power. After leaving the MOSFET on for some time, it not necessary to measure the temperature considering that the MOSFET really did not heat up.
Recommendations for Modifications:	None

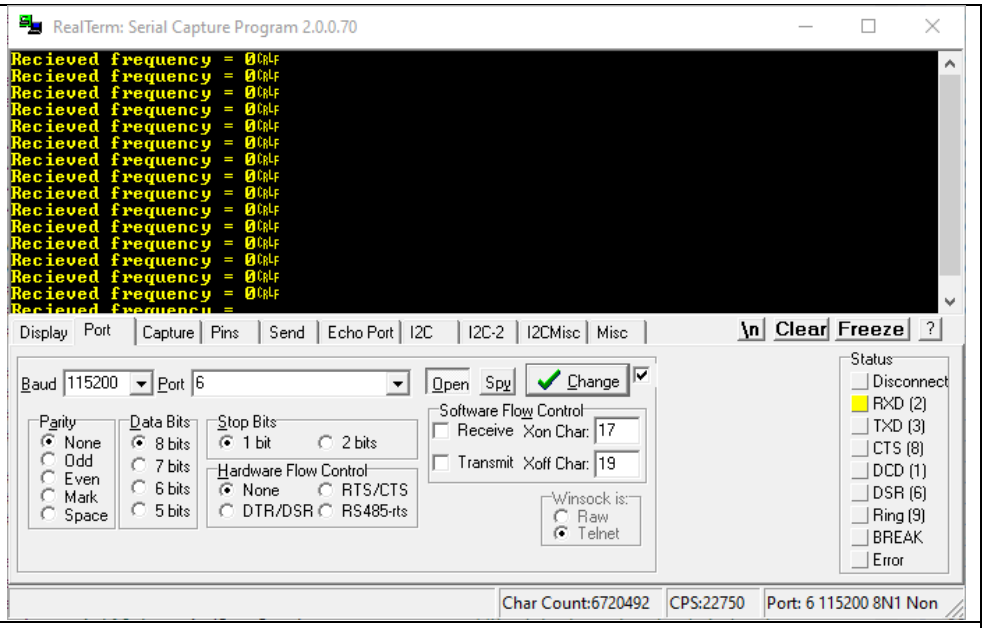
TNC Testing Form (REV1)	
Leaf on the Tree	MOSFET
Device Under Test (Testing Tree Number):	1.2.4
Date:	11/1/20
Person(s) Conducting Experiment:	Kobe Keopraseuth
Signature:	
Experiment Purpose:	The purpose of this experiment is to ensure that the MOSFET is operating correctly. We are using the MOSFET as a switch to pull 15V, coming from the radio, to ground. This will tell the radio that the TNC is transmitting and the radio can stop transmitting.
Experiment Procedure:	We will implement the circuit shown below and input 15 V with a pull-up resistor, to act as the radio's 15 V. Then we will use a tactile switch to switch the MOSFET on and off. A voltmeter will be used to make sure our drain to source voltage becomes very small when the MOSFET has high signal input at the gate.
Equipment Settings / Software Settings (w Revision):	We use a breadboard to hook up the circuit shown below and a dc power supply for the 15 V. We used LTspice for designing the circuit. We use 3.3V reference to supply to the gate.
Testing Diagram / Picture:	 **Circuit** 

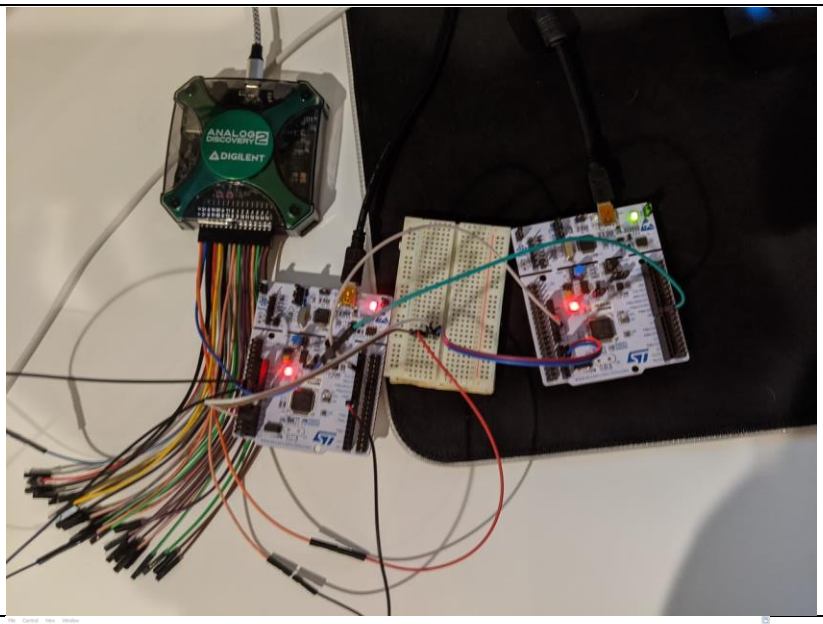
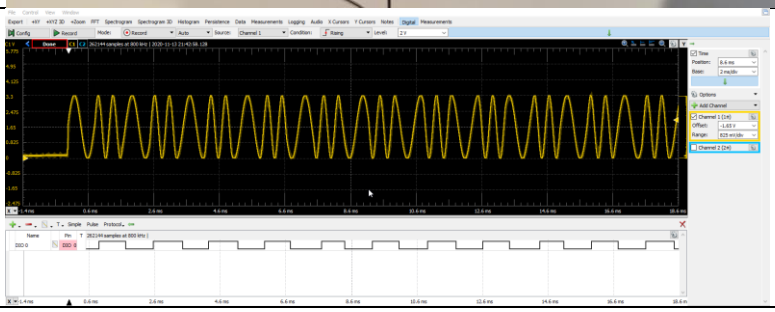
Data Points:	 **MOSFET Off**  **MOSFET on**
Pass / Fail:	Pass
Interpreted Notes:	As can be seen when a low signal is inputted into the gate, then the MOSFET's drain to source is 15 V. As can be seen when a high signal is inputted into the gate, then the MOSFET's drain to source is 3.2 mV, which should signal the radio to stop transmitting.
Recommendations for Modifications:	None

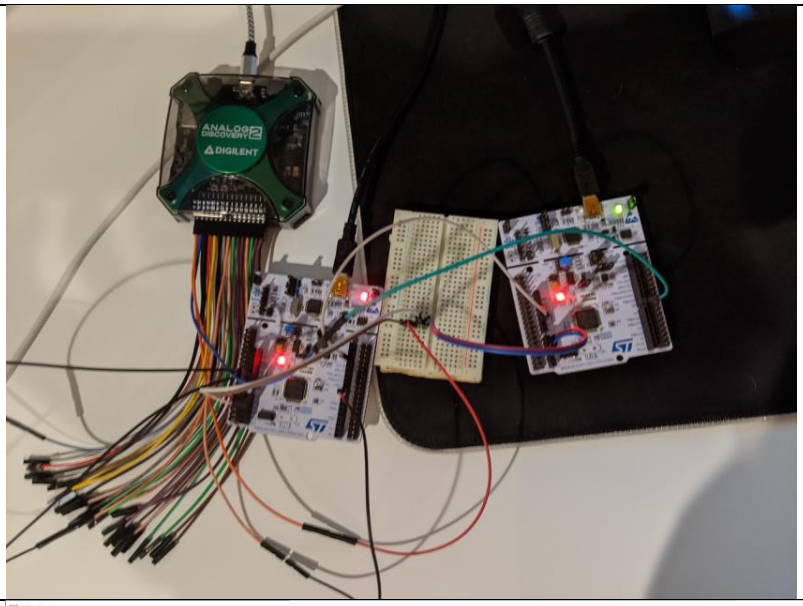
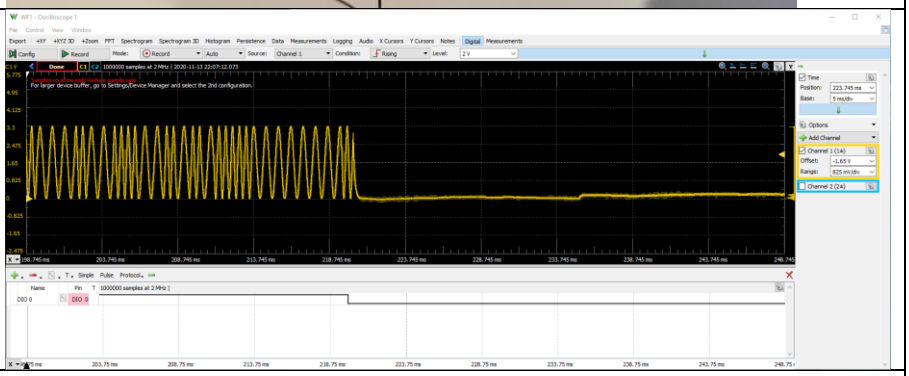
TNC Testing Form (REV1)	
Leaf on the Tree	Voltage Input
Device Under Test (Testing Tree Number):	1.2.5.1
Date:	11/1/20
Person(s) Conducting Experiment:	Kobe Keopraseuth
Signature:	
Experiment Purpose:	The purpose of this experiment is to validate that the MOSFET can operate correctly when inputted 15 V and as low as 5 V.
Experiment Procedure:	Input 5 V and 15 V in series, with a pull-up resistor, into the drain of the MOSFET and see if the voltage from drain to source drops to a very small voltage.
Equipment Settings / Software Settings (w Revision):	Use our microcontroller to input a high signal into the gate and 5V into the drain. Also input 15 V into the drain using an external voltage supply.
Testing Diagram / Picture:	
Data Points:	 **5 V input**

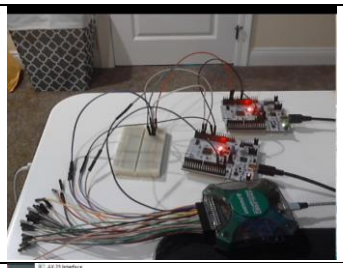
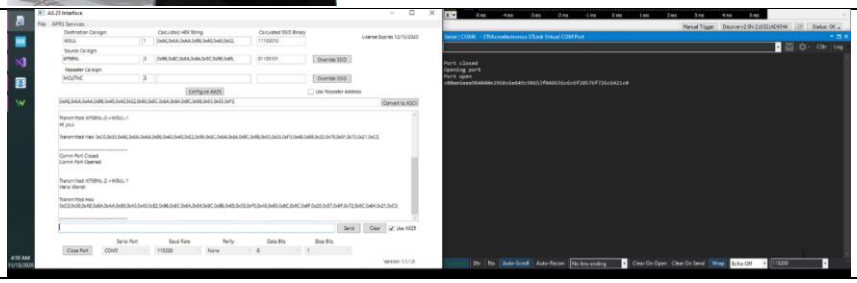
	 **15 V input**
Pass / Fail:	Pass
Interpreted Notes:	The MOSFET was successfully able to drop the input voltages very close to 0 V across the drain and souce.
Recommendations for Modifications:	None

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:	

	
Pass / Fail:	Fail
Interpreted Notes:	Frequency output remains zero due to gain not being high enough to trigger interrupt.
Recommendations for Modifications:	Modify amplifier circuit to increase gain.

TNC Testing Form (REV1)	
Leaf on the Tree	Proper Frequencies
Device Under Test (Testing Tree Number):	2.3.1.2.1.1
Date:	11/13/20
Person(s) Conducting Experiment:	David Cain
Signature:	
Experiment Purpose:	Verify the frequency returned from ADC reading matches from input waveform.
Experiment Procedure:	Connect one TNC, to the next, output a packet. The receiving TNC will output a digital signal representing the interpreted frequency in binary.
Equipment Settings / Software Settings (w Revision):	The Digilent will be set to record the frequency of the waveform Measured and view the digital output of the receiving TNC.
Testing Diagram / Picture:	
Data Points:	
Pass / Fail:	Pass
Interpreted Notes:	Output seems to correlate with the specific frequency very well.
Recommendations for Modifications:	None.

TNC Testing Form (REV1)	
Leaf on the Tree	AX.25 Flags
Device Under Test (Testing Tree Number):	2.3.1.2.2.1
Date:	11/13/20
Person(s) Conducting Experiment:	David Cain
Signature:	
Experiment Purpose:	Verify system detects HDLC frame end flags properly.
Experiment Procedure:	Connect a TNC to output a packet and test if our TNC can correctly detect the ending flag. An inject binary signal will be present on the digital output that toggles when frame end flag is captured.
Equipment Settings / Software Settings (w Revision):	The Digilent will be set to record the frequency of the waveform measured.
Testing Diagram / Picture:	
Data Points:	
Pass / Fail:	Pass
Interpreted Notes:	TNC successfully interprets a frame ending flag.
Recommendations for Modifications:	None.

TNC Testing Form (REV1)	
Leaf on the Tree	MCU Terminal Node Controller
Device Under Test (Testing Tree Number):	0
Date:	11/13/20
Person(s) Conducting Experiment:	Kobe Keopraseuth, Kaleb Leon, David Cain
Signature:	
Experiment Purpose:	Verify full system functionality. This will ensure a TNC can output one signal and be received by another running the same code and vice versa.
Experiment Procedure:	Connect our TNC's together running most current software. Send a packet over UART to one, then receive it on the other and compare the packet.
Equipment Settings / Software Settings (w Revision):	Most current version of our own software running on the TNCs.
Testing Diagram / Picture:	
Data Points:	
Pass / Fail:	Pass.
Interpreted Notes:	The same packet that was sent over uart and transmitted was properly and efficiently received by the other TNC.
Recommendations for Modifications:	None!

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.

Change Order Form (COF) Rev 1	
Date	11/1/2020
Subsystem	Signal Reception/Transmission Lines 1.1.1
Component	RS-232 port
Change	Removal from design
Reasoning	Design will not support this port anymore.
Rationale	Due to the time constraints of the semester, we will not have time to implement a RS-232 jack.
Mod. Details	Removal

Change Order Form (COF) Rev 1	
Date	11/1/2020
Subsystem	1.3 (Audio Input Circuit)
Component	1.3 (Audio Input Circuit)
Change	Will remove audio input circuit all together.
Reasoning	After consulting with our mentors, we realized that the process of recovering the clock and data from an AFSK signal using solely digital logic was quite difficult and they overlooked this when assigning the task.
Rationale	Nolan is providing a modem board that will simply output binary values for us to interpret the data and clock from. We will complete our project utilizing this board but will lay a foundation for the C.A.P.E. research group to build on to achieve a minimal hardware solution for recovering the data in the future.
Mod. Details	Remove amplifier and filtering circuit from the design and will now utilize a modem board provided by Nolan.

Change Order Form (COF) Rev 1	
Date	11/13/2020
Subsystem	2.3.1 (ADC)
Component	2.3.1 (ADC)
Change	Change data encoding method from NRZ to NRZI
Reasoning	NRZI is the correct way data communications are handled.
Rationale	After reverse engineering the signal received from a working off the shelf TNC we discovered that the signal is NRZI encoded rather than NRZ.
Mod. Details	Changed the algorithm that encodes and decodes NRZI rather than NRZ.

Change Order Form (COF) Rev 1	
Date	11/13/2020
Subsystem	2.3.1 (ADC)
Component	2.3.1 (ADC)
Change	Change frequency detection method
Reasoning	Zero crossing and Schmitt trigger was not providing accurate frequencies for the input waves. In addition, it was not fast enough.
Rationale	The two older methods of frequency detection did not provide us with valid frequencies and could not detect the shifts. In addition, it caused a large amount of bit errors. The new algorithm using phase comparison was a lot faster and more accurate.
Mod. Details	Changed frequency detection method to phase comparison.

APPENDIX H

A. Parts List

- STM32F446RE Nucleo board
- 3.5mm Audio Jack
- USB-mini cable

B. Theory of Operation

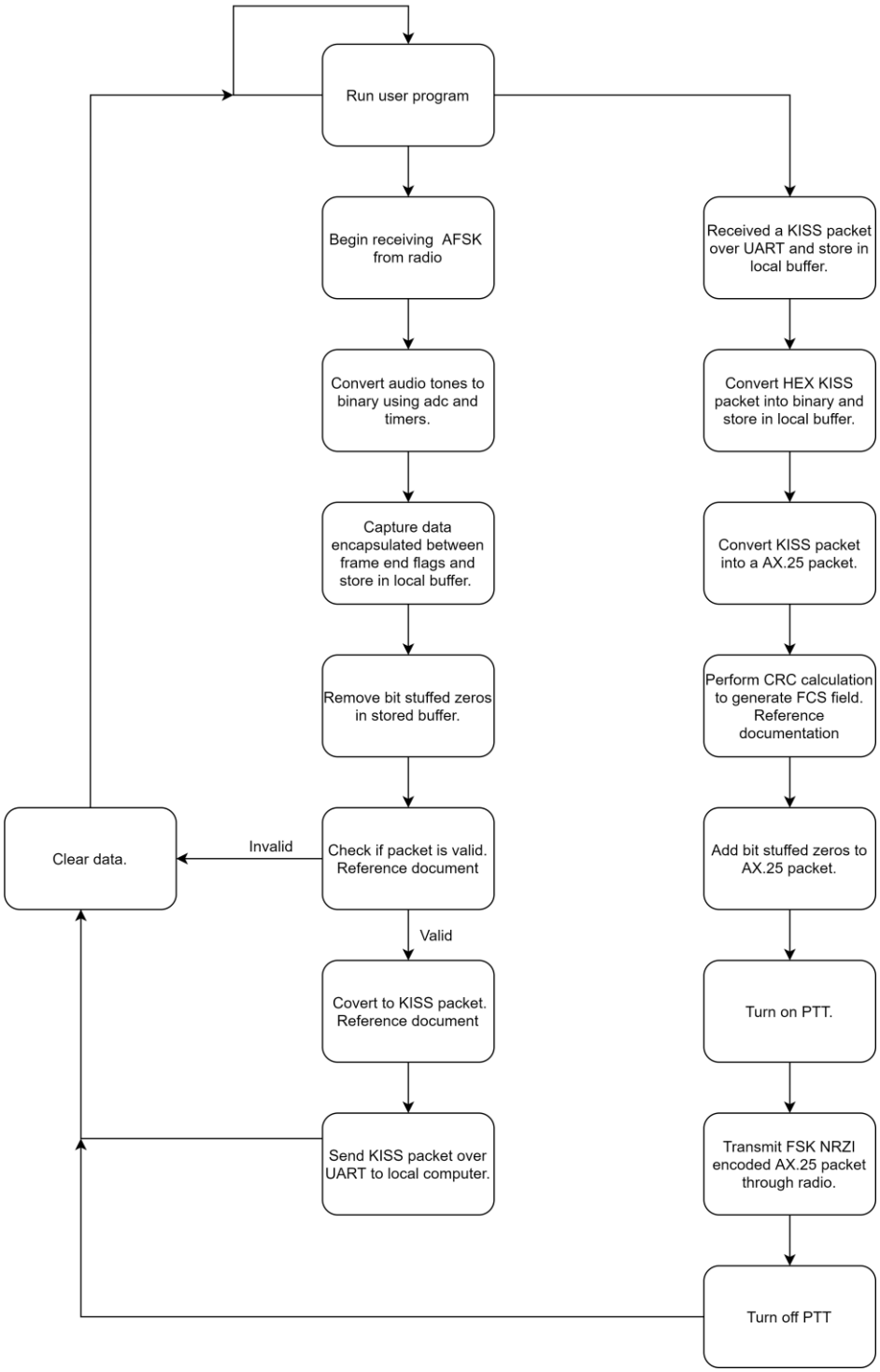


Figure H-1. General Runtime Flowchart

C. Current Hardware Setup

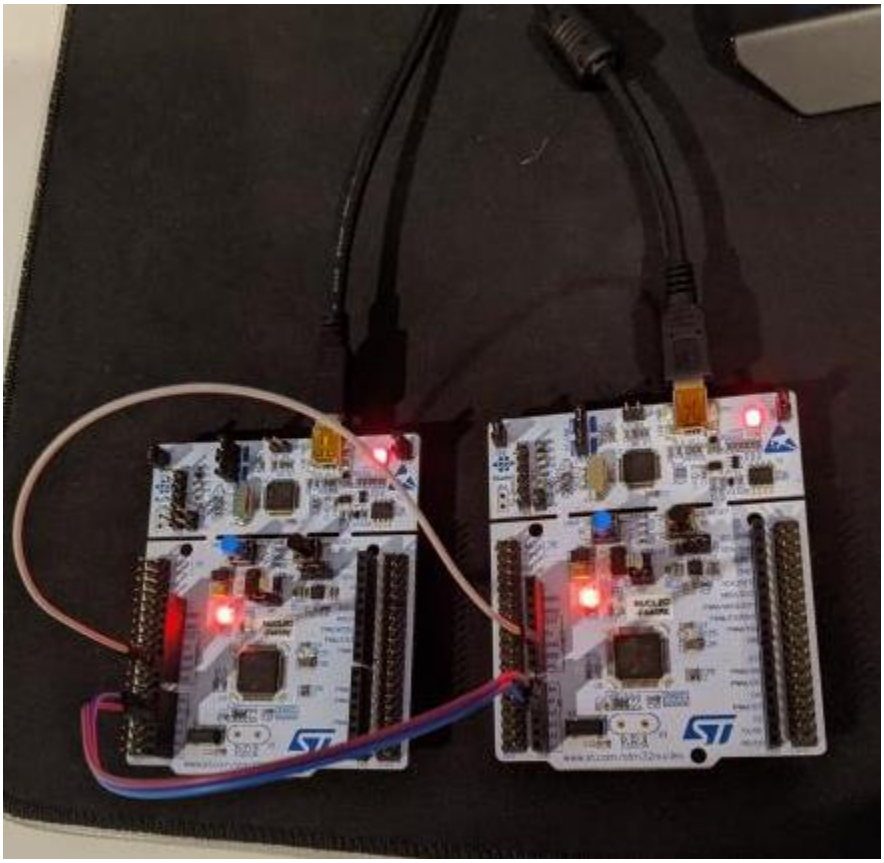


Figure H-2. Current Hardware Setup

D. Hardware

No longer contains any external circuits so the only hardware being used is the micro controller and its connections. The hardware handles all the formatting of packets as well as receiving and transmitting their packets in their proper form. Inputs and outputs into and from the micro controller from the PC side would be a HEX KISS packet sent over UART on the USB-mini cable. Inputs and outputs into and from the micro controller from the radio side is a FSK NRZI encoded analog signal from the radio connected either using GPIO analog pins or the audio jack.

E. Software

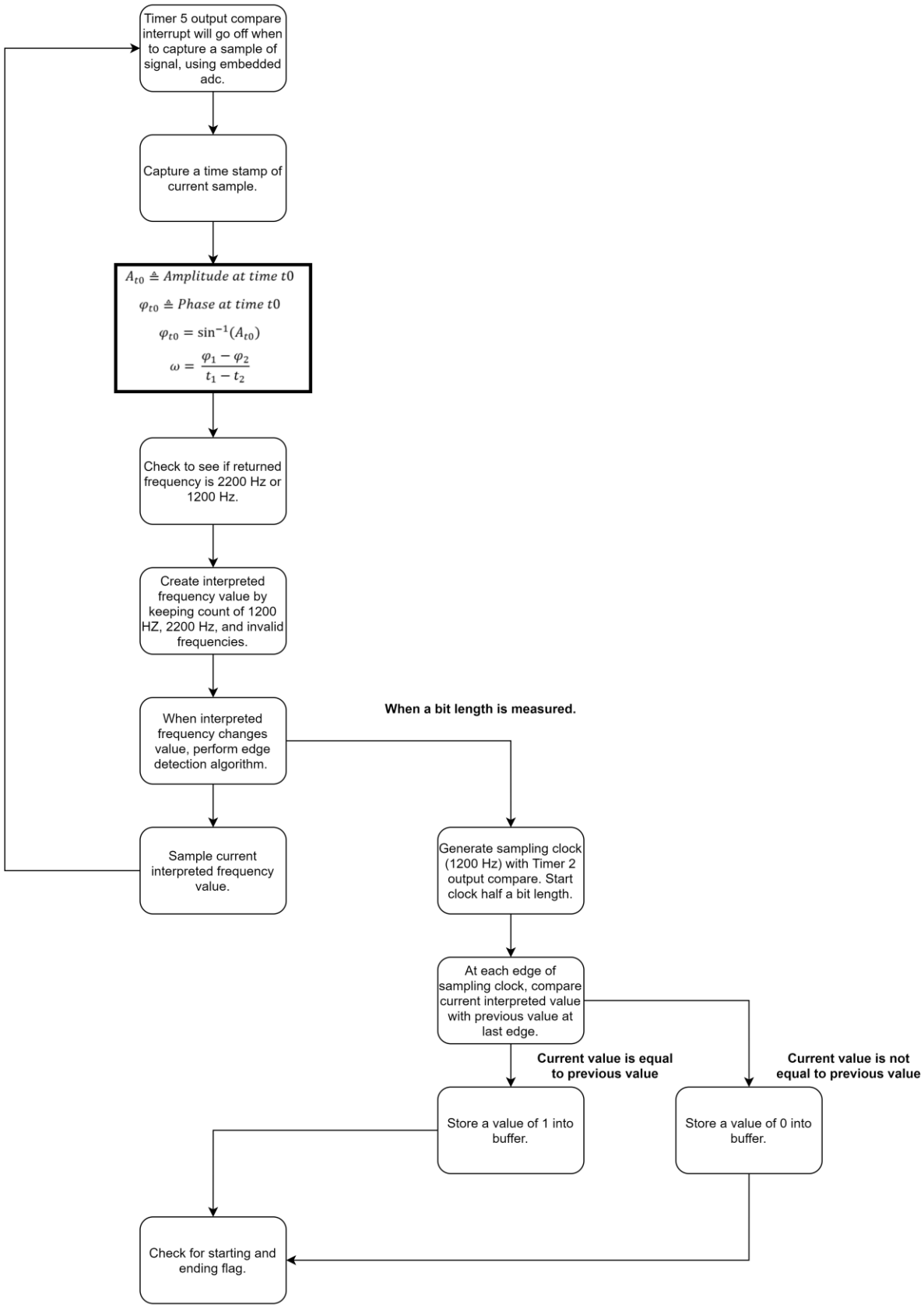


Figure H-3. Decoding AFSK Signals

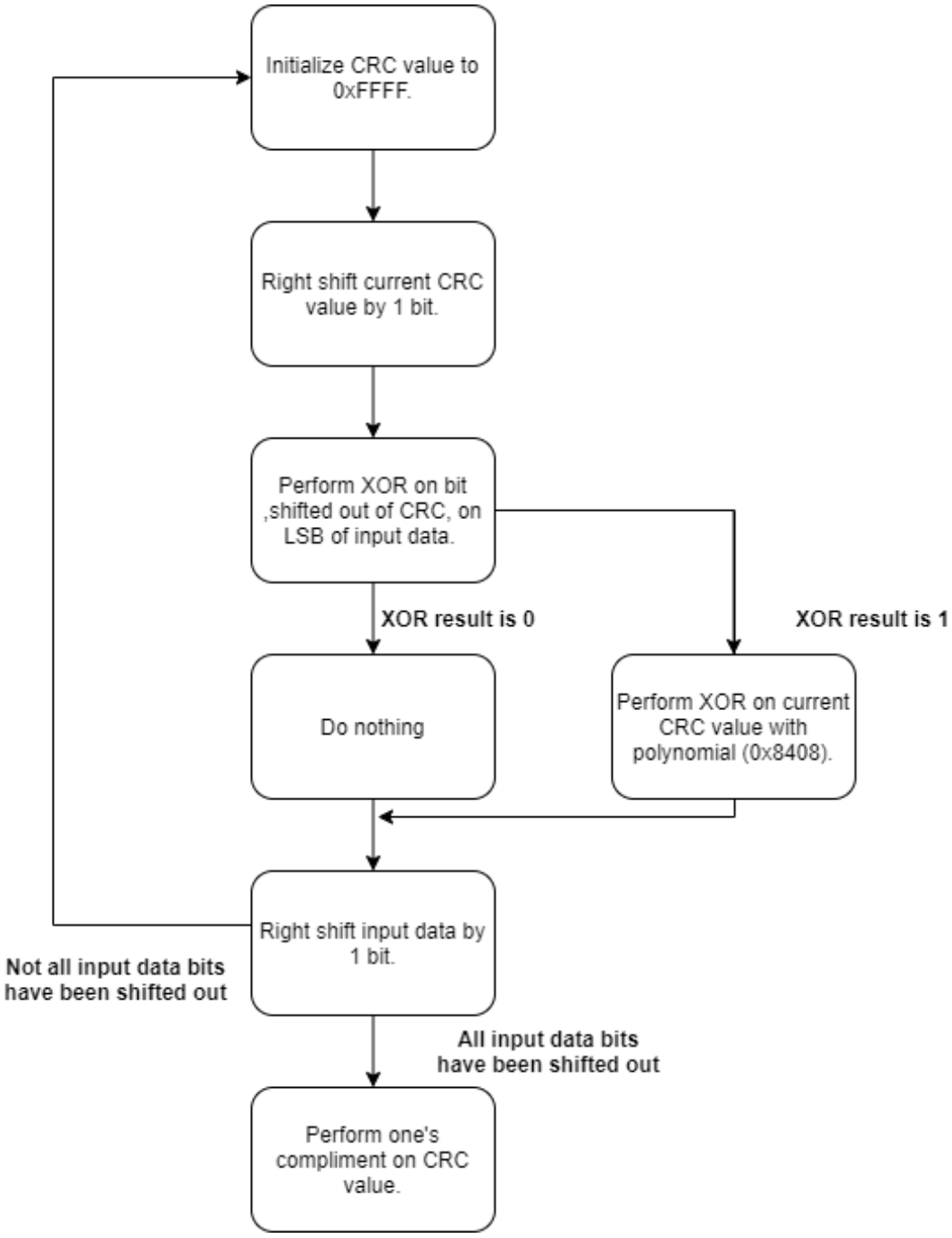


Figure H-4. CRC Calculation

F. Future Work

Our TNC performs the basic functionalities of an actual TNC. Functionalities such as converting AX.25 to KISS, KISS to AX25, receiving digital or analog signals, and transmitting digital or analog signals. Functionalities that were not implemented were APRS which gives location of TNC and others, digipeating which allows for AX.25 packets to go long distances, distinguishing between different types of packets, and sending and receiving signals over radio. Currently our TNC will always assume it is transmitting or receiving an unnumbered information frame, rather than being able to categorize the packet as an information, supervisory, or unnumbered frame.

G. How to find our code?

Link to our GitHub Repository:

https://github.com/MrLordLeon/TNCMCU

ReadMe On How to Navigate the GitHub:

Project 2: TNCMCU

Project Members:
- David C.
- Kobe K.
- Kaleb L.

Project Mentors:
- Nolan E.
- Rizwan M.
- Nick P.
- James P.

Project Description
Current implementations of the Terminal Node Controller use a telephone FSK modem to transmit binary data via the APRS system. These FSK modems are outdated, unavailable, and obsolete. The signals are transmitted at 1200 baud using audio signals of 1200 and 2200 Hz.

Project Challenge
Design, implement, build, and test a TNC with an STM32 or similar microcontroller. The system must accept a string via UART or a similar interface and generate the necessary APRS and HDLC headers/packet data and then produce the correct audio waveform. This audio waveform will then be fed to a speaker, handheld radio, HAM radio set, or dedicated FM modulator (design team not responsible for this portion, only the audio waveform generation).The system must also accept an audio waveform input from a microphone, handheld radio, HAM radio set, or dedicated FM receiver (design team not responsible for this portion, only the audio waveform interpretation) and be able to translate this audio into APRS/HDLC, and transmit a fully translated message to the controlling device via UART or a similar interface. The initial implementation should use COTS development kits as a proof of concept. Once the software and design have been verified and validated, the hardware shall be implemented into a standalone PCB with connections for a PC, Raspberry PI, Arduino, and standard handheld radio.
The team will have to familiarize themselves with the following:
- microcontroller C programming and hardware including:
  - timers
  - interrupts
  - ADC
  - DAC
  - various comms interfaces
- schematic design and PCB layout
- analog circuitry design
- APRS, HDLC, and other packet radio standards
- responsible software development using thorough verification/validation techniques including:
  - flowcharting

- failure analysis
- unit testing
- etc.

Ideally, this team would be composed of 2 students able to put at least 8 hours per week on this project. I have already done some proof of concept work on the frequency detection and sine waveform generation, and both features work to my satisfaction.

Repo notes

Not a permanent structures, very flexible!

Design Deliverables - Generally anything we need for assignments or papers can be put here

Hardware Specs - Documentation for any hardware used in the project. Renaming the file to something interpretable is preferred, I.E. don't download from mouser and have the file named file001.pdf.

Presentations - This is where we'll keep presentations for mentors or Dr. Darby. Please precede file name with date of presentation.

Protocol Documentation - This is where any relevant information about AX.25/KISS/HDLC/etc... can go. Same process with the hardware specs, useful naming means useful people.

Schematics - This is where we put our schematics for our external circuits.

Software - This is where all of our main code and testing software is stored.

Subsystem Documentation - This is where we store our information on CRC calculation.

APPENDIX I

H. *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
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
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
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
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
1	2/24/2020 10am-1:30pm	Beginning to put together a data flow chart for the AX.25/KISS protocol	3.5
2	2/26/2020 8pm-9:30pm	Finish flow chart with Kaleb	1.5
3	3/3/2020 6am-7am	Create BOM to present to mentors to get ahold of components	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
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
3	3/8/2020 2pm-3pm	Review GitHub documentation and ensure it is competent and well documented.	1
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
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
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
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
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
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
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	4/19/2020 10:50 pm-12:32am	Wrote appendix sections for assignment 3 and created the design 1 diagram.	1.70
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
1	9/14/2020 5:30pm-7pm	Begin creating experiment sheet by overlooking sheet provided by Dr. Darby.	1.5
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
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
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
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	9/21/2020 5:30pm-7pm	Create BOM for prototyping components with Kobe and then send to Nolan	1.5
2	9/22/2020 3:30pm-6pm	Worked on AX.25 handling code, implementing struct	2.5
3	9/23/2020 5:30-7pm	Worked on AX.25 handling code, added some packet logic based on frame received types	1.5
5	2/25/2020 12pm-12:30pm	Retrieved perfboard from campus	0.5
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	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	10/1/2020 5:30-7pm	Completed testing for the baud rate output of TNC. Used the Digilent to measure waveform.	1.5
5	10/3/2020 12pm-6:30pm	Worked on packet structure code.	6.5

2) Kobe Keopraseuth

Item	Date/Time	Description	Hours
1	2/5/2020 5:00pm-9pm	Research Protocols	4
2	2/11/2020 5:00pm-7pm	PowerPoint for Mentors	2
3	2/16/2020 11am-1pm	Search Components	2
4	2/17/2020 6:00pm-7pm	Make objective tree	1
5	2/17/2020 7pm-8pm	Cost Analysis	1
6	2/17/2020 8pm-9pm	Regulatory Considerations	1
7	2/17/2020 9pm-11:30pm	Alternatives and Trade Offs	2.5
8	2/20/2020 5:00pm-7pm	Make Poster of project	2
9	2/26/2020 1:00pm-7pm	Flow Chart for Mentors	6
10	2/25/2020 11am-3pm	Research Digipeaters and Bit stuffing for AX.25 protocol.	4
11	3/1/2020 5:00pm-7pm	Peer Review Team 1	
12	3/3/2020 4:00pm-5pm	Meet with Mentors	1
13	3/3/2020 6pm-7pm	Mandatory Cape Meeting	1
14	3/7/2020 11am-6pm	Start using stm32 microcontroller	7
15	3/9/2020 7:pm-10pm	Assignment 1B	2.5
16	3/9/2020 10:pm-12pm	Assignment 1C	2.5
17	3/29/2020 1:00pm-5pm	Programming STM32 to read frequency inputs	4
18	3/31/2020 6pm-7pm	Fixing Flowchart	1
19	3/31/2020 3pm-6pm 4/1/2020 5pm-7pm	Assignment 2B	5
20	3/31/2020 7:pm-10pm 4/1/2020 7pm-10pm	Assignment 2C	6
21	4/6/2020 3pm-8pm	Assignment 3	5
22	4/8/2020 2pm-2:30pm	Timeslots for presentations	0.5
23	9/20/2020 5pm-9:30pm	Deliverable 3	4.5
24	9/19/2020 2pm-7pm	CRC code	5
25	9/17/2020	Rx circuit schematic	4.5
26	9/25/2020 5pm-6pm	BOM for materials purchased	1
27	9/26/2020 9pm – 2am	KISS to AX.25 code	5
28	10/3/2020 5pm-10pm	Code for CRC generatio	5
29	10/4/2020 1pm – 6pm	Solder PTT	5
Total			**91**

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 onLevel 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
1	10/9/2020 5:00 – 6:30 PM 10/18/2020 5:00 – 6:30 PM	Met with Mentors to discuss progress and final design specs	3
2	10/12/2020,10/14/2020,10/16/2020 3:00 – 8:00 PM	Ran Tests on DAC and ADC code	5
3	10/15/2020 4:00 – 5:00 PM	Added Deliverable 3 to paper	1
4	10/16/2020 - 10/18/2020 12:00 – 6:00 PM	Wrote Deliverable 4	6