Northern Arizona University

Team Radio Pi

Capstone Team

Tyler Plihcik

Brandon Click

Jeffrey Williamson

Kaelen Carling

Saurabh Jena

Project Sponsor

General Dynamics

Mission Systems

Randy Derr

Mentor

David Failing

Requirements Specification Document

Version 3.0

November 17h, 2020

Accepted as baseline requirements for the project: “Lightweight Command and Control Solution for Embedded Products”

For the client:________________________ For the team: [illegible]

Date:________________________ Date: 11/18/2020

Table of Contents:

Introduction

The world runs on multitudes of IoT (Internet of Things) devices; everything from smart home devices, to wifi extenders, speakers, even cars and kitchen appliances. These devices communicate via radio waves and run on what are called embedded chipsets, which are inherently small, in both footprint and power consumption. As our technology continues to advance, the need for smaller and smaller embedded systems becomes more urgent.

For nearly three decades software defined radios (SDRs) have been revolutionizing the way these IoT devices communicate. Leading the charge into this new era of is our project sponsor, General Dynamics Mission Systems. The need for newer, smaller, more efficient yet more powerful systems is great. Current solutions are often times built on legacy software, and although they still work, are definitively not the best solution given our technological advances since the late twentieth century. Because of this, Software Defined Radios are prone to:

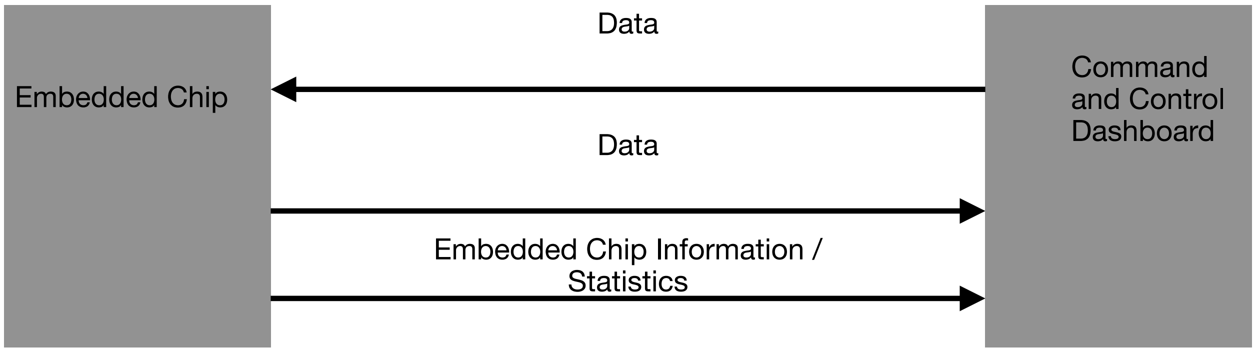
- Failure
- System crashes
- Memory overloads
- General decay

Our envisioned solution is to redesign General Dynamics SDR Command and Control System from the ground up. We will be simulating this solution on a Raspberry Pi, a very small yet powerful, computer. We will establish an interconnected database and web interface in the most efficient language(s) available, including networking protocols and database languages. We will then ensure that the end user has complete and total control, given they have the proper authorization, over the system. From querying the database to collecting, sorting, and storing data to controlling permissions and viewing system diagnostics, our envisioned solution will attempt to right all the wrongs of current command and control interfaces for embedded systems.

The following will serve as the basic requirements that Team Radio Pi will develop and deliver to General Dynamics Mission Systems, as well as provide insights into the problems that we are trying to solve, the solutions we have envisioned, a risk analysis, and our project plan.

Problem Statement

In 2018 General Dynamics reported a revenue of $36.19 billion. Being one of the largest defense contractors in the country, they have many different sectors within the company, including various weapon system and combat vehicles, we focus on their command and control system division. The command and control systems for their software defined radios (SDRs) are used to send and receive data, sort that data, and display statistics and measurements of what is happening onboard the embedded chips.


Embedded Chip
Data
Data
Embedded Chip Information / Statistics
Command and Control Dashboard

When these SDR devices are deployed, they must have a command and control system (hence forth referred to as C2 systems) to monitor and configure / reconfigure their behaviors. The current solution General Dynamics (GD) employs is based on an outdated version of Adobe Flash, which was then updated to a HTML5 based solution. These solutions have done their jobs over the years, but there is a sense of disconnection between them, a feeling that everything has been stitched together to achieve the absolute bare minimum requirements of a C2 system. These solutions were never efficient even when they were using current technology. The solutions in use right now are:

- Inefficient
- Outdated
- Too large of a footprint
- Stitched together by just a few “threads”

Solution Vision

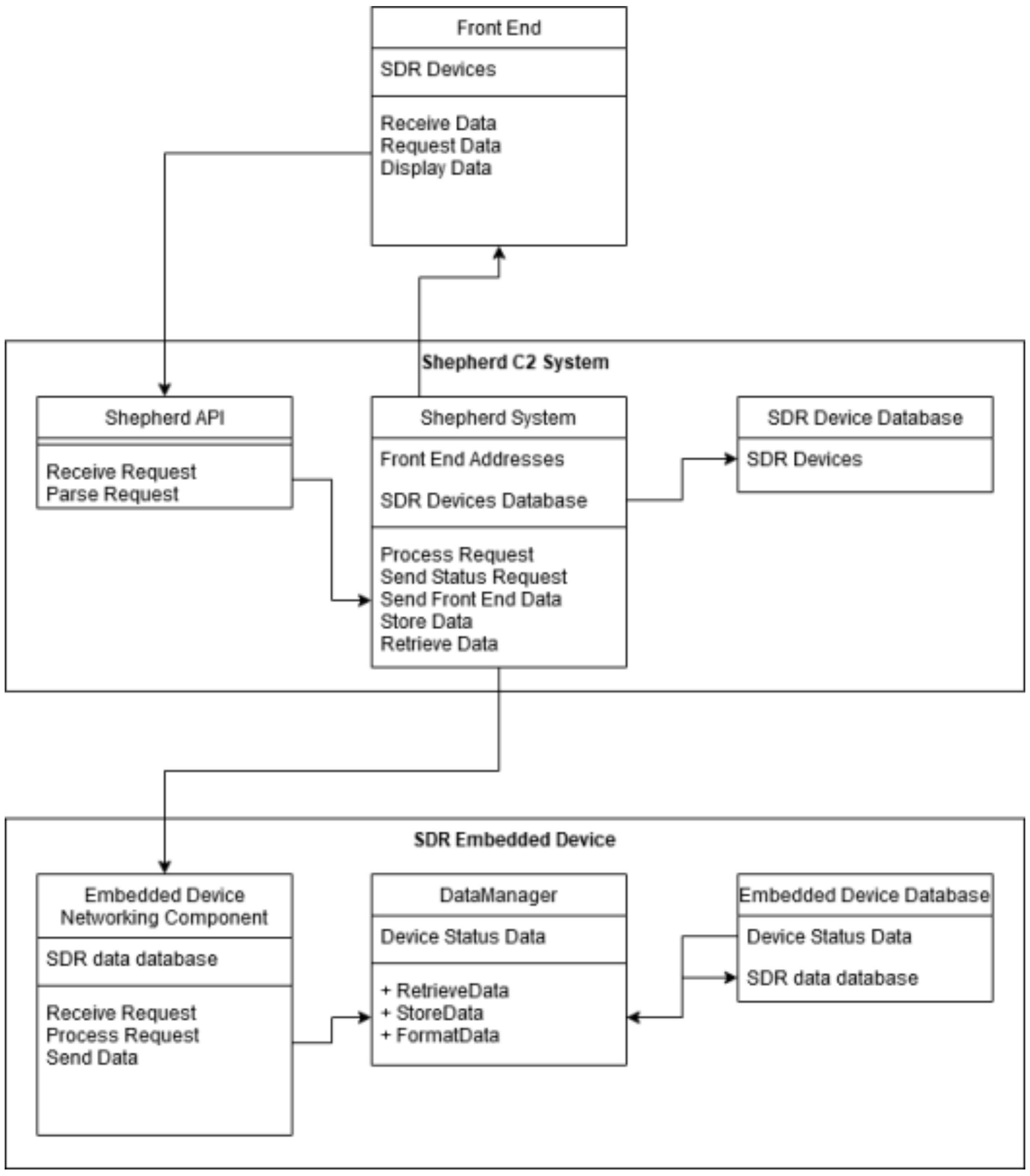
Our envisioned solution, coupled with our directive from the project sponsor, is to build a completely redesigned and reimagined C2 system to interact with GD's software defined radios, simulating real hardware with a Raspberry Pi 3 B+. Our solution will feature:

- Fast and efficient transfer of data
- Modern hardware
- A slick user interface
- Small overall footprint
- High throughput
- Low power consumption
- Elegant continuity of tasks and functions

Our solution will be a simulated prototype, which we will present to the project sponsor as a proof of concept.

Due to the nature of IoT devices and the embedded chipsets that run them, every detail must be accounted for in-order to gain maximum efficiency. Every watt consumed, byte used, and so on. We will be using multiple networking protocols for different tasks, because a one-protocol-fits-all solution simply will not do in this case. Our solution, running on the Raspberry Pi, will support data such as device temperature, throughput, data transfer status messages, memory consumption, processor state, and more. This data will be used to ensure data is sent and received between devices and end users in a correct and timely manner, as well as monitoring the footprint of the system. Tasks involved in monitoring include making sure that the device is not using too much power, and that all processes are being run without taking up too much memory. Our solution will offer General Dynamics a new way of designing C2 systems for their software defined radios. This solution will make an attempt at keeping GD at the forefront of the IoT communication industry, by helping them keep absolute control of everything the SDR does, in one small, efficient package.

We have chosen to complete this using Raspberry Pi's because of their extreme availability, and inherent small footprint. Even so, a Raspberry Pi still has much more computing resources available than the current chips GD is using, so we aim to build our solution using as little resources as possible. This will ensure that as technology continues to evolve, IoT communication will continue to get smarter, and faster, connecting more people and places than ever.



The above diagram details which details our envisioned system architecture and provides some key insights into our project design. Starting from the bottom of the diagram, and moving our way up, the reader can see that the SDR embedded device, or in our case the SDR embedded device being simulated on a Raspberry Pi 3 B+, contains three core components:

- Embedded device database
- Data manager
- Embedded Device Networking Component

These three components interact as follows. The Embedded device networking component will receive and process all requests from the external command and control system, or other functions within the simulated SDR device. The data that it receives will be sent to the device data manager, the device data manager will in turn, send that data to the appropriate tables within the database. The embedded device database will also send data when queried to the data manager, and the data manager will send appropriate queried data to the requesting functions within the simulated SDR device.

The Shepherd C2 system is a component of our overall system that will be essential to its success, especially once we add more networked simulated SDR devices. The Shepherd system contains three core components:

- The SDR Device database
- The Shepherd system
- The Shepherd API

When dealing with multiple networked devices all of which include onboard embedded databases, there must be an intermediary system to monitor and control all of the them, not unlike a shepherd would monitor, corral, and move their flock of sheep, thus: The Shepherd System.

The SDR device database will contain device information for all deployed SDR devices in an effort to ensure complete organization across every level of our solution. This database will interact also with the database that is contained onboard every simulated SDR device.

The Shepherd system itself will interact with the SDR device database and the our front end user interface. It will also process requests, send status requests, as well as storing and receiving data. This will interact with the Shepherd API, whose main function is to receive and parse requests from the front end user interface.

Finally, our graphical user interface will provide the user with an easy to use, slick, accessible way to interact with the command and control system, which is used to monitor, command, and control all of the simulated SDR devices.

Project Requirements

Our directive is to produce a proof of concept for a potential new lightweight command and control system for embedded products.

The following three sections will discuss requirements set forth by General Dynamics. The functional requirements will outline the functionalities our system must and must not have. Non-functional requirements pertain to behaviors of the system, and environmental requirements have to deal with non-negotiable requirements given by the client.

Functional Requirements

The embedded database function shall save radio data provided by multiple other software functions in the radio; We will develop the database in such a way that data such as temperature, file and file transfer metadata, commands to the device, time stamping, statistical data on the receiver, and signal to noise ratio, just to name a few, will all be logged in the database.

The embedded database function shall respond to queries, providing data to multiple other software functions in the Radio; We will develop the database so that when queried by other device functions, the data required by those functions will be retrieved and transmitted efficiently in order to execute all desired functionality of the device.

The embedded database function shall define a query protocol to be used by other software functions in the Radio; We will develop the database such that any and all querying done by all parts of the simulated SDR system will follow the same protocol when querying, and there will be a uniform format used by all functions when performing queries.

The embedded database function shall provide thread-safe access and handle access from multiple processes/applications; We will develop our command and control system such that when necessary, the system will begin using multi-threading in order to run multiple processes at the same time, to maximize efficiency.

The embedded database function shall provide status of pending/in process/completed transactions to other software functions in the Radio; We will develop the database in such a way that it will log the status conditions mentioned for each and every data transaction between various functions within the device.

The embedded database function shall timestamp data as it is entered into the database; We will develop the database in such a way that each and every piece of data entered in has been time stamp and it’s associated metadata logged.

The embedded database function shall accept a time reference from another software function in the radio; In accordance and working in parallel with the requirement above, we will develop the database in such a way that when other functions within the device send data to be logged in the database, that data has a time associated with it — that time not being a timestamp when logged but when that particular measurement or statistic was taken.

The embedded database function shall support storage and retrieval of multiple data types. A list of data types shall include Binary, Hexadecimal, ASCII strings, pointers/links to files stored in the radio file system. We will build the database tables in such a way that they will efficiently and effectively store all of the above data types, with room to accommodate for more if need be. The ambiguity in this requirement pertaining to the specific data types is left on purpose to allow for additions if need be as development goes on. We plan on exceeding the expectations of our client and leave this space open for the end of our development cycle so we can prove the feasibility of our proposed solution to General Dynamics even further.

The embedded database function shall provide status information on its operation when requested. Status information to include database health status, database size, etc…; We will build the database in such a way that upon request, the database will return a variety of information including what is listed and potentially more. The ambiguity in this requirement pertaining to the specific status information to be returned is left on purpose to allow for additions if need be as development goes on. We plan on exceeding the expectations of our client and leave this space open for the end of our development cycle so ewe can prove the feasibility of our proposed solution to General Dynamics even further.

The embedded Database Function shall preserve data while the Radio is powered off; We will build the database in such a way that when the device is powered off, for whatever reason, the data contained in the onboard data base will be preserved with no loss whatsoever.

The embedded Database Function and the data stored within it shall survive the host processor being unexpectedly powered down; We will build the database in such a way that in the even of an unexpected shutdown, whether that be do to system failure, electrical failure, weather conditions, accidental damage, or whatever the reason may be, the data stored within the onboard database will be preserved.

Non-Functional Requirements

The embedded Database Function shall provide configurable mechanisms to manage the storage required for the database by automatically deleting some data. We will develop the database and the tables contained in it such that after a certain amount of time, insignificant data will be deleted in an effort to keep the overall footprint of the system small.

> In this case, insignificant data refers to data which is not system critical, for example processor core temperature data, which according to the client, can be estimated to be collected from five different sensors every thirty seconds. This will collect and start to take up more memory than it should over time, so deletion of it and other data like it is imperative.

The embedded Database Function shall support a configurable set of data. The various tables which will be contained in our database shall be able to be configured and reconfigured when deemed necessary by the database administrator via the command and control system we will develop.

The embedded Database Function shall operate within the processing power and memory constraints of an embedded processor; We must build the database such that its overall footprint is kept extremely small, compared to today's memory and data standards. Since we are simulating this SDR device with a Raspberry Pi 3 B+, we are given much much more memory and processor power than needed. Nevertheless, we will build it to use as little processing power and memory as possible while still accomplishing its required functionality.

Environmental Requirements

The embedded Database Function shall operate under a Linux operating system; We must build the database such that it will operate on Linux or a Linux variant. Our Raspberry Pi 3 B+'s will be runnings Raspberry Pi OS, which itself is a variation of a Linux operating system.

The project shall prototype and evaluate the recommended embedded database product's ability to GDMS requirements. The prototyping will use a mutually agreed upon Commercial Off the Shelf (COTS) computer board; We shall build this system using a Raspberry Pi 3 B+, which was the mutually agreed upon COTS computer board agreed upon unanimously by both Team Radio Pi Members, and General Dynamics Mission Systems.

Potential Risks

The following is an outline of potential risks, which have been analyzed by Team Radio Pi during the planning phase of our development cycle. These risks are rooted in the domain where command and control systems lie, as well as relevance to other solutions currently on the market or in development by competitors.

The first potential risk we outline is perhaps the most important when dealing with data transfer, and that it data loss in transmission. We have chosen to use both UDP and TCP as our networking protocols, when appropriate. TCP offers a confirmation that all data has been delivered to the desired destination, but at the cost of speed. However UDP drops the receipt confirmation all together in favor of that speed that TCP lacks. Now this does not mean the data is entirely lost, more that the data was not delivered with the other data it was sent with. For this reason we have chosen to only use UDP under circumstances in which the data transfer is not system critical. Data transferred using UDP will be for example, processor temperature, data that in the event of loss during transmission, can be omitted and recorded at the time of next transmission. However any and all data loss must be considered and that is why this potential risk is outlined here.

Associated with the above risk, we must also consider the potential of a complete networking connection failure. This could happen as a result of an internet outage or similar events. In this event, although the device will not be able to communicate with other devices or the system at large, it will still have the ability to log and store all of its own data. The issue with this is that if the system if offline for a long enough time, there is potential, however theoretical, of data overflow, and therefore, loss.

Next we explore the potential risks involved in taking advantage of multi-threading in a system. Thanks to modern computer architecture, most computers on the market have the ability to take advantage of multi-threading, or running more than one process at the same time. However many benefits this has, something as complex as multi-threading can actually hinder some systems. If not implemented correctly, multi-threading can cause errors such as data bottlenecks or race conditions. In this case that race conditions occur as a result of threading errors, data transferred could be entirely inaccurate. Inaccurate data could lead to false reports, bad intelligence, and so on. It is imperative that the data we transfer from embedded database to command and control system be kept accurate.

Another potential risk is hardware failure. In the event that the embedded system experiences an hardware failure such as processor malfunction or memory card corruption, data being collected in the database will be halted, leading to the untimely delivery of information to important individuals. Our software must be able to withstand such a hardware failure in any circumstance, and preserve the state of the database.

We must also make sure that our command and control system is secure. Because we will be storing data such as data transfer status, and other important system information, we must make sure that such information is not tampered with during transfer. If system information or status is tampered with, the inaccurate data could be potentially dangerous to those using and acting upon it. Reports made with compromised data are inaccurate and therefore invalid, we must maintain the integrity of our data at all points in the process.

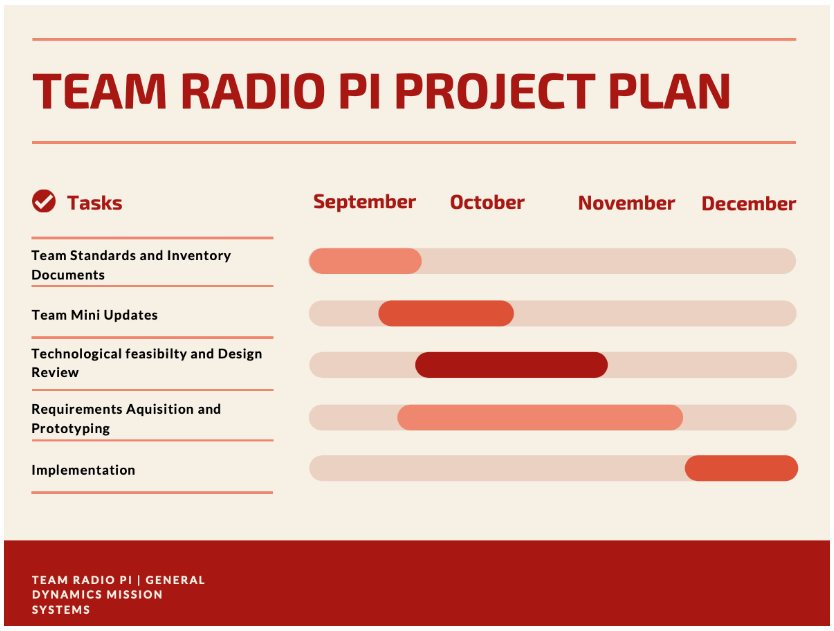
Additionally, we must consider unexpected power supply shutdowns of the device caused by external events. SDR devices can be placed anywhere in the world, sometimes in extremely inhospitable environments. The SDR could in some circumstances be prone to completely unexpected shutdowns of unknown duration. It is imperative that in the event of this happening, the data stored within the embedded device, must be preserved.

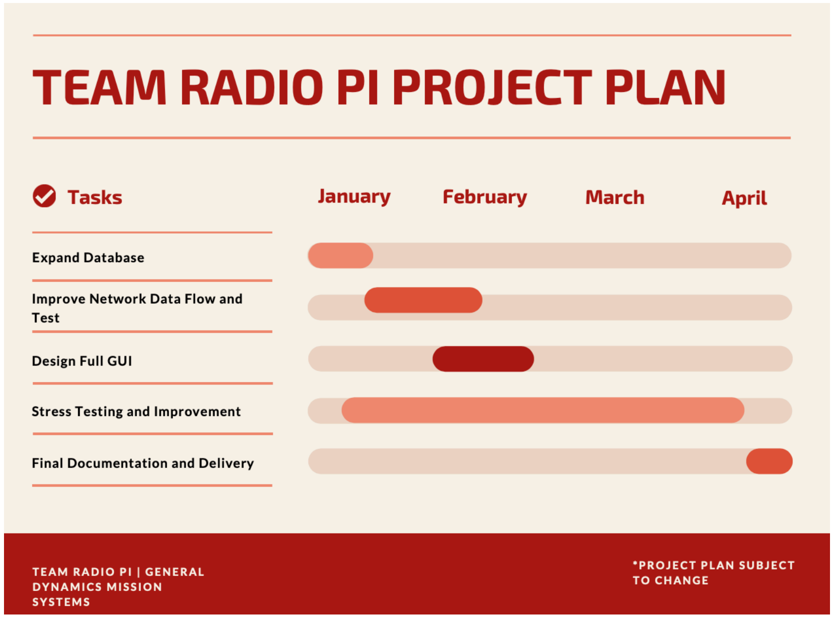
Finally, we must consider certain factors and series of events that could lead to incompletion of the project. These risks include, but are not limited to hardware failure during development, our code repository crashing resulting in complete or partial loss of work, as well as other external events such as physical illness or harm that has happened to one or more of our team members. We plan to mitigate these risks as follows. To plan for possible hardware failure, mainly Pi board failure during development, we have acquired multiple Raspberry Pi 3 B+'s from General Dynamics, to which code can easily be ported over in the event of an individual board failure. To add to our protection we also have multiple multiple secure backups of all of our code to ensure that progress is not loss. To plan for physical illness or harm to team members, we actively work ahead of schedule, thus ensuring that we are able to continue on schedule in the case of unfortunate unforeseen events.

Anytime a team undertakes a complete ground up redesign for a product that has been available for years, and still works, however poorly that may be, there are inherent risks in the undertaking. However, we prefer to look at these risks as hurdles to overcome, rather than walls we should fear. We take the potential for these risks very seriously and will take every precaution necessary during our implementation phase to mitigate such risks.

Project Plan

As of the writing of this document, we have completed our technological feasibility research and have made our decisions as they pertain to the various technologies we will be using in conjunction with the hardware we will be implementing it on. Additionally, we have begun to engineer a prototype demonstration to be presented at the end of this month (November 2020), and plan to work through the winter break on implementing our solution in order to achieve the most complete version of our envisioned solution to be delivered to the project sponsor by the end of the Spring 2021 semester.





In our prototyping, we plan to initialize our database, and create the communications network between that database and what will eventually become our user interface. Thus confirming the statements we have made in our technological feasibility research and the accompanying document, proving that we have chosen the correct database and can in fact communicate with it.

During our implementation phase, we will be expanding on the prototype. We will begin by filling out the database with the proper tables for the various data types it must support, as laid out in our project requirements above. Once those tables have been created, we will expand the robustness of our networking to account for large quantities of data flow. After these essential tasks have been completed we will implement multi-threading into our database solution to ensure that in the case that more than one process must be run at once, our solution can handle it and performance will not be deprecated. After completion and testing of our multi-threading implementation, we will finally move onto creation of a slick, usable, and accessible front end. This front end will contain a dashboard for the embedded chips, including status, various processor statistics, data entry and retrieval from the database, and the ability to retrieve logs from the on-board chipsets as well.

You can see all of these tasks, in a broader sense detailed in the chart pictured above this text, with implementation continuing well into 2021, for final product delivery in late April 2021.

Conclusion

As technology continues to evolve, and the needs of everyday embedded systems continue to become more complex. While SDRs continue to become more prevalent we must strive to make our embedded systems run with less power consumption, require less memory, and efficiently and effectively execute the functions it was designed for.

General Dynamics' current command and control system for their embedded systems, and specifically their SDRs, is outdated, inefficient, and has been stitched together over the years. Our solution will offer a proof of concept for a ground up redesign of this command and control system, simulating GD hardware with a Raspberry Pi 3 B+.

Throughout this document, we outlined the exact requirements set forth to us by the project sponsor, General Dynamics Mission Systems. This document, working in parallel with our first design review, will serve as the starting point for our prototype demonstration, which begins our implementation cycle.

During this requirements acquisition phase, we have learned that, for our project, the database system and our chosen implementation are absolutely crucial to the overall success of the project. Keeping our entire solution footprint as small as possible, in order to have it work extremely efficiently on embedded chipsets is paramount.

From this point forward we move into the implementation of our solution, beginning with prototyping. We are all confident in our ability to complete requirements laid out in this document in full, and look forward to exceeding the expectations of both the NAU capstone staff and the team at General Dynamics, in producing a Lightweight Command and Control System for Embedded Products.

Glossary

UDP - User Datagram Protocol, a core member of the internet protocol suite

TCP - Transmission Control Protocol, a core member of the internet protocol suite

GUI - Graphical User Interface, the interface the user interacts with to control the computer

C2 - Acronym for Command and Control System

Chipset - a set of electronic components in an integrated circuit that control the data flow with in a computing device

Embedded device - a special computing device, which has been specifically programmed for one and only one specialized task. Typically of smaller architecture

SDR - Software defined radio, a radio device in which the traditional hardware components have been converted to software

Raspberry Pi - A small single board computer developed by the Raspberry Pi foundation

IoT - Internet of Things, commonly referring to a network of small embedded devices