Making Microgrids Plug & Play

DESIGN DOCUMENT

Team Number: sdmay23 - 36
Client: Nick David
Advisor: Mathew Wymore
Team Members/Roles
Andrew Frank - API Research/Network Engineer
Christian Pinta - API Research/Developer
Austin Thoreson - Systems Engineer
Ben Eder - Software Developer
Saketh Jonnadula - Software Developer
Team Email: sdmay23-36@iastate.edu
Team Website: https://sdmay23-36.sd.ece.iastate.edu

Revised: 12.02.22 / v1.0

Executive Summary

Development Standards & Practices Used

- Tactical Microgrid Standard (TMS) - Draft
- SunSpec Modbus
- TCP/IP communication

Summary of Requirements

Functionality:

- Facilitate communication between two or more Microgrid pallets through TCP/IP
- Send configuration commands to onboard OutBack Inverter and system
- Arbitrate master/slave designation between pallets automatically
- Provide a display to view system status and major settings

Resources:

- Accept communication information following SunSpec Modbus from a connected grid
- Possibly use this information to advise configuration settings
- Microcontroller to facilitate above communication
- Communication from other grids/controllers

User Experiential

- Users can attach Microgrid pallets together without additional configuration.

Aesthetically:

- Controller should fit on top of the pallet and connect to the existing RJ45 ports and OutBack AXS converter
- The display should replace MATE3 controller/display

Constraints:

- Fixed hardware for the existing system(s)
- Proprietary communication protocols with some existing systems (may be a non-issue)
- No user input, the system should configure itself automatically

Applicable Courses from Iowa State University Curriculum

- E E 201: Electrical Circuits
- E E 230: Circuits & Systems
- COM S 309: Software Development
- COM S 311: Algorithm Analysis

- ENGL 314: Technical Communications
- CPR E 381: Computer Organization & Design
- CPR E 430: Network Protocols & Security
- MATH 424: Intro to High Performance Computing

New Skills/Knowledge acquired that was not taught in courses

- Experience in Python
- SunSpec / Modbus Protocol
- Distributed systems concepts
    - Consensus over a distributed network
- How to put together professional grade design documents for projects

Table of Contents

Definitions

Abbreviation/Symbol	Definition
ISU EPRC	Iowa State University Electric Power Research Center
OMG DDS	Object Management Group Data Distribution Service
TMS	Tactical Microgrid Standard
RPi	Raspberry Pi
SunSpec Modbus	Open communication standard for Distributed Energy Resource systems
OutBack Power	Hardware device supplier
AXS Port	OutBack Power device to translate between TCP/Modbus over ethernet to proprietary OutBack Power communication
Microgrid / Pallet	A self-sufficient local electrical grid. In this document, refers to the PowerPallet Mobile design from the ISU EPRC

List of Figures and Tables

1 Team

1.1 Team Members

1) Andrew Frank 2) Christian Pinta

3) Austin Thoreson 4) Ben Eder

5) Saketh Jonnadula

1.2 Required Skill Sets for Your Project

Knowledge about computer hardware

Knowledge about electrical components

Knowledge about networking and network protocols

Experienced using various programming languages

Ability to Interpret Technical Documentation

1.3 Skill Sets covered by the Team

Knowledge about computer hardware: Christian, Austin, Andrew, Ben

Knowledge about electrical components: Austin, Andrew

Knowledge about networking and network protocols: Christian, Austin

Experienced using various programming languages: All team members

Ability to Interpret Technical Documentation: All team members

1.4 Project Management Style Adopted by the team

Agile

1.5 Initial Project Management Roles

(Enumerate which team member plays what role)

Development Members - Andrew, Austin, Ben, Christian, Saketh

2 Introduction

2.1 Problem Statement

Everyone needs electricity. Microgrid pallets provide a mobile, modular power solution that can be applied to many use cases. Choice use cases are natural disaster relief, tactical deployment for military operations, or replacing gasoline powered generators. Currently, connecting these pallets is time consuming and requires technical expertise specific to these devices.

The goal of our project is to simplify the connection and configuration of pallets so that minimal to no training at all is required to use them. An extension of the project is to allow communication from the pallets to any connected electric grid and provide additional services.

2.2 Intended Users and Uses

The main use case focus for this project is on the military operation deployment scenario. Military personnel will have limited time and training to learn how to use the Microgrid pallets, and some untrained personnel may need to operate the equipment also to set up temporary camps. Because of this, the pallets need to be easy to modify, repair, and set up.

Outside of the military, the microgrids need to be easily used by various disaster relief groups in order to provide power to those in need. The microgrids need to not only be simple to use but also be able to provide power quickly

Another user group is anyone in the general public who may need power for any reason. Whether it's to charge their phone or power a festival, anyone who needs power should be able to receive it simply by using one of these microgrids.

2.3 Requirements & Constraints

Functional Requirements - What a product should do

- Facilitate communication between two or more Microgrid pallets through TCP/IP
- Send configuration commands to onboard OutBack Inverter and system
- Arbitrate leader/follower designation between pallets automatically
- Provide a display to view system status and major settings

Resource Requirements - What external resources a system may use

- Accept communication information following SunSpec Modbus from a connected grid
    - Possibly use this information to advise configuration settings
- Microcontroller to facilitate above communication
- Communication from other grids/controllers
- Communication following the TMS

Physical Requirements - Product size, weight, shape, etc.

- Controller should fit on top of the pallet and connect to the existing RJ45 ports and OutBack AXS converter
- The display should fit alongside or replace MATE3 controller/display

Aesthetic Requirements - How the product should look

- The interface should be simple with little to no interaction needed by the end user

User Experiential Requirements - How the user will interact with the product

- Users can attach Microgrid pallets together without additional configuration.

Constraints - Limiting thresholds

- Fixed hardware for the existing system(s)
- Proprietary communication protocols with some existing systems (may be a non-issue)

2.4 Engineering Standards

Tactical Microgrid Standard (TMS) - Draft

Some of the primary applications of these units are in disaster relief and military applications. TMS is meant to be the standard for Department of Defense (DoD) and industry needs/applications. TMS is meant to provide simple setup, be efficient, with resilient generation and distribution, all while being an open architecture.

SunSpec Modbus

Open communication standard that standardizes parameters and settings for monitoring and controlling distributed energy resources (DER). This protocol enables communication amongst different devices from multiple vendors within the same device/grid. Sunspec is made up of various IEEE protocols standardizing signal-to-noise ratios, and other aspects of network communications.

TCP/IP communication

To communicate with the pallets (which use a proprietary communication protocol) we send TCP/IP packets containing data which conforms to the SunSpec standard. An OutBack hardware device converts from their proprietary communication to TCP/IP, so to implement high level functions like master/slave arbitration we will need to communicate with this standard.

3 Project Plan

3.1 Project Management/Tracking Procedures

We will be adopting the agile project management style. The agile style will allow us to go with the flow and continue to iterate and refine the project as it grows. This methodology works best for our project because, even this late in the planning stage, we still find different ways to solve certain problems. When we do come across such an idea, agile will allow us to face it head on and while we keep moving ahead.

We will use Discord for general communication and collaboration.

We will track issues/milestones by using Gitlab which we will also use to commit and keep our code organized.

Weekly meeting minutes are tracked with documents in a shared Google Drive.

3.2 Task Decomposition

- Prepare RPi with OS and Python libraries
    a. Python 3.8 (may need to change)
    b. Py Sunspec 2
- Raspberry Pi is usable as controller for the Radian Inverters
    a. Develop a master controller arbitration algorithm
    b. Create command issuing and receiver software using SunSpec
    c. Develop several presets that can be issued by the RPi during initialization
- Raspberry Pi can detect or assign a leader/follower role for connected grids
    a. Decide on a leader via Bully Algorithm (to be implemented)
    b. Configure inverter to follow assignment
    c. Detect leader/follower state based on physical configuration
- Display can show important parameters, leader /follower state and heartbeats, and any fault status
    a. Pull model from inverter and parse relevant data
    b. Create display mock-up
    c. Select graphics library
    d. Program actually display
- Microcontroller can accept user input to determine configuration
    a. Determine capabilities of the inverter and network to be exposed to users
    b. Design a system to detect microgrid connections
    c. Use connection status to rollout appropriate preset
- Raspberry Pi can interpret TMS commands and issue SunSpec commands to configure microgrid pallet inverters
    a. Test scripts with SunSpec python library
    b. Create configuration sequences for various setups

3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

- Two Raspberry Pis are able to read and write values sent to each other over TCP
- Two microgrids are able to reliably determine which is a master and which is a slave
- Have a working display that shows important variables about the microgrid
- Fully SunSpec compliant
- Full stable, autonomous communication between two microgrids
- Fully TMS compliant

3.4 Project Timeline/Schedule

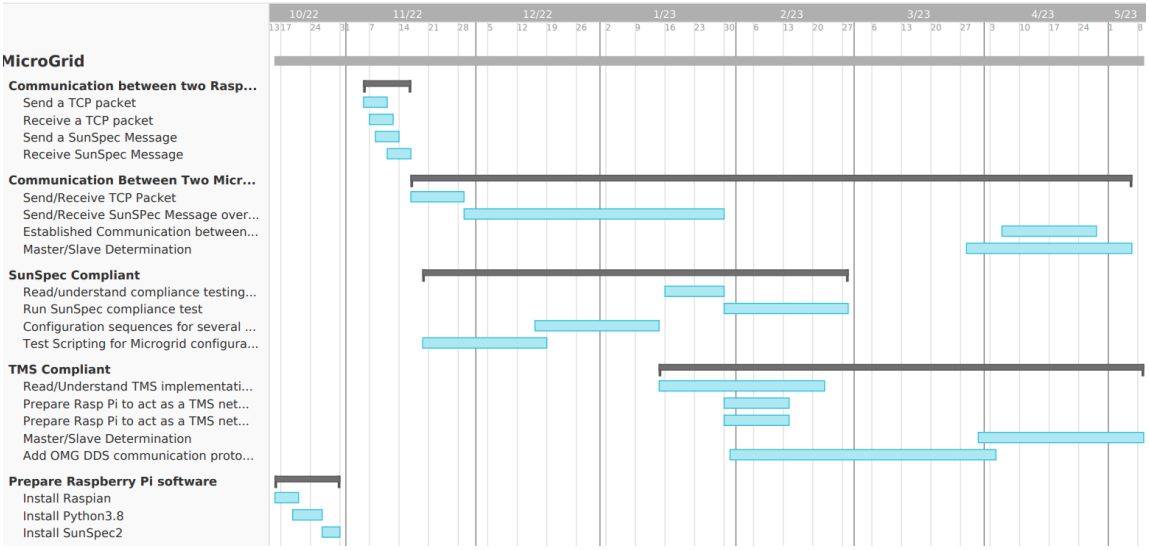

Figure 3.4: Project Gantt Chart

3.5 Risks And Risk Management/Mitigation

Task	Risk Description	Risk Probability	Mitigation Plan
RPi Preparation	Damage to RPi through incorrect installation	0.1	
RPi as Radian Controller	Damaging the Microgrid Units	0.1	
	OutBack hardware does not support setting stacking mode over SunSpec	1.0	Contact OutBack Power for work-arounds, or possible modification to the MATE3. Assign all as their own Leader and coordinate through the TMS network.
Leader / Follower Assignment	Security flaws in network communication	0.3	
Creating the Display	Unable to read inverter status when not Leader	1.0	Send status over network with heartbeats
TMS Interpretation	Unable to fully support TMS functions	0.5	Implement reduced TMS functionality
	OMG DDS may be unavailable	0.2	

Table 3.5: Risks and Mitigation Plans

3.6 Personnel Effort Requirements

	A	B	C	D	E	F	G
1	WBS #	Name / Title	Type	Start Date	End Date	Estimated Hours	Actual Hours
2	1	MicroGrid	project	10/14/2022	5/8/2023	All	0
3	**1.1**	Communication between two Raspberry Pis	group	11/4/2022	11/15/2022	5	0
4	1.1.1	Send a TCP packet	task	11/4/2022	11/9/2022	0.5	0
5	1.1.2	Receive a TCP packet	task	11/7/2022	11/10/2022	0.5	0
6	1.1.3	Send a SunSpec Message	task	11/8/2022	11/11/2022	2	0
7	1.1.4	Receive SunSpec Message	task	11/10/2022	11/15/2022	2	0
8	**1.2**	Communication Between Two Microgrids	group	11/16/2022	5/4/2023	40	0
9	1.2.1	Send/Receive TCP Packet	task	11/16/2022	11/28/2022	X	0
10	1.2.2	Send/Receive SunSpec Message over Network	task	11/29/2022	1/27/2023	13.33333333	0
11	1.2.3	Established Communication between Microgrids	task	4/5/2023	4/26/2023	13.33333333	0
12	1.2.4	Master/Slave Determination	task	3/28/2023	5/4/2023	13.33333333	0
13	**1.3**	SunSpec Compliant	group	11/18/2022	2/27/2023	40	0
14	1.3.1	Read/understand compliance testing documents	task	1/16/2023	1/27/2023	5	0
15	1.3.2	Run SunSpec compliance test	task	1/30/2023	2/27/2023	2	0
16	1.3.3	Configuration sequences for several setups	task	12/15/2022	1/12/2023	23	0
17	1.3.4	Test Scripting for Microgrid configuration	task	11/18/2022	12/16/2022	10	0
18	**1.4**	TMS Compliant	group	1/13/2023	5/8/2023	100	0
19	1.4.1	Read/Understand TMS implementation documents	task	1/13/2023	2/21/2023	20	0
20	1.4.2	Prepare Rasp Pi to act as a TMS network reciever	task	1/30/2023	2/13/2023	20	0
21	1.4.3	Prepare Rasp Pi to act as a TMS network controller	task	1/30/2023	2/13/2023	20	0
22	1.4.4	Master/Slave Determination	task	3/30/2023	5/8/2023	15	0
23	1.4.5	Add OMG DDS communication protocol	task	1/31/2023	4/3/2023	25	0
24	**1.5**	Prepare Raspberry Pi software	group	10/14/2022	10/28/2022	1.5	0
25	1.5.1	Install Raspian	task	10/14/2022	10/19/2022	1	0
26	1.5.2	Install Python3.8	task	10/19/2022	10/25/2022	0.25	0
27	1.5.3	Install pySunSpec2	task	10/26/2022	10/28/2022	0.25	0

Figure 3.6: Personnel Effort Requirements

1.1 Communication between two Raspberry Pis: The goal here is to simulate the end product with less moving parts. Sending and receiving TCP packets should be a simple websocket connection. In order to send a SunSpec message there may be more involved with using the SunSpec api to do so.

1.2 Communication Between Two Microgrids: To accomplish this task, we need to learn more about microgrids what they will be exchanging in their communication. This seems like a task that will be much more involved than a simple TCP message going from one microcontroller to another. We are also factoring in the time it will take to get acquainted with the pySunSpec library as well.

1.3 SunSpec Compliant: Most time spent here will be understanding the tests and how they need to be run to test compliance. The running of the tests themselves should not be overly tedious. Depending on results of the test, there may need to be adjustments made to the configurations. Some time spent on understanding initial configurations may show up here also.

1.4 TMS Compliant: The Tactical Microgrid Standard doesn't seem to have a straightforward or automated way to test for compliance. Compliance testing here will most likely require a high level of understanding of TMS and other related standards, including Sunspec and OMG DDS protocols. In this process the team will also have to work out Master/Slave assignment in an automated manner. This on its own may be a heft goal.

1.5 Prepare Raspberry Pi software: This process should be pretty straightforward, the longest part may be simply downloading the OS image onto the pi.

3.7 Other Resource Requirements

Identify the other resources aside from financial (such as parts and materials) required to complete the project.

- Semi-Regular access to micro-grid pallets
- Raspberry Pi's (2x)
- Documentation on API's and associated tech
    - Microgrids
    - Sunspec
    - Tactical Microgrid Standard
    - OMG DDS

4 Design

4.1 Design Context

4.1.1 Broader Context

Our design would potentially have a broader impact on communities in disaster relief situations, and other portable power applications. If the setup of the microgrids is more accessible, the deployment should be quicker and thus further reaching in terms of scale of the networks. With larger networks/more deployments, more people in need would be able to get the help they need. With our automation of microgrid setup and coordination, they will be able to be used by people with a variety of backgrounds and serve higher-power requirement applications.

Relevant considerations related to our project:

Area	**Description**	**Our Project**
Public health, safety, and welfare	How does your project affect the general well-being of various stakeholder groups? These groups may be direct users or may be indirectly affected (e.g., solution is implemented in their communities)	Increasing availability/ease of use of emergency deployment equipment by providing portable power. Increasing availability/ease of use of renewable energy deployment equipment by providing portable power.
Global, cultural, and social	How well does your project reflect the values, practices, and aims of the cultural groups it affects? Groups may include but are not limited to specific communities, nations, professions, workplaces, and ethnic cultures.	The microgrids are much more environmentally friendly compared to generators as they are rechargeable and do not emit harmful gasses. The batteries used are also recycled. These microgrid pallets can act as distributed storage devices on an energy grid. When they are fully developed it may encourage moving to a distributed grid model rather

		than the centralized utility grid in use today. The Tactical Microgrid Standard (TMS) used as a framework for this project is in the draft stages right now, however our successful implementation may improve its development.
Environmental	What environmental impact might your project have? This can include indirect effects, such as deforestation or unsustainable practices related to materials manufacture or procurement.	The microgrids are designed for use with renewable power sources such as wind and solar, so they may increase the use of renewable energy in military operations. They use reclaimed batteries for energy storage which reduces electronic waste.
Economic	What economic impact might your project have? This can include the financial viability of your product within your team or company, cost to consumers, or broader economic effects on communities, markets, nations, and other groups.	The product itself is quite expensive, our work shouldn't affect cost much compared to the cost of existing hardware.

Table 4.1: Project Impact Context

4.1.2 Prior Work/Solutions

There are various products on the market that provide similar solutions in terms of generating and storing energy from renewable sources. Very few of these products are compliant to the TMS standard, which is a goal for our project.

Pallets like this exist and the specific pallets we are working on, do function. The problem is that those pallets are not TMS compliant and they have complex interfaces that require previous experience with electricity in order to operate them. The goal of our project is to automate a lot of the pallet setup and create a simple, user friendly interface to allow anyone to use the pallets if they wanted to.

What our project aims to achieve is building upon these existing solutions by making them easier to configure. The interconnectivity, portability, and thus scalability of these packs/microgrids is what sets our project apart. Many solutions out there either do not support this connectivity or it is difficult to achieve.

4.1.3 Technical Complexity

The existing system contains an inverter/charger and energy storage unit with a user interface that is networked with a proprietary communications protocol developed by Outback Power.. Our design will require connection to this network and to a typical TCP/IP ethernet network, and coordination between the two.

The software we are creating must handle many different tasks, and we have divided it into several modules. One to handle user input and a graphical display, several to handle communications over the TMS network, and one to translate between user input, TMS communications, and communications over the Outback proprietary network.

Communication with proprietary hardware and software protocols is difficult in and of itself. We don't have access to proprietary information for troubleshooting and/or expanding upon device functionality.

In order to implement anything, we need a strong understanding of the system and standards that are required. It takes a lot of time to sift through and digest the documentation before we can begin making informed design decisions.

4.2 Design Exploration

4.2.1 Design Decisions

1. Network topology
    a. Multiple networks and protocols
        i. Raspberry Pi communication
            1. TCP/IP
            2. Tactical Microgrid Standard
        ii. Raspberry Pi to AXS Port
            1. SunSpec communication
        iii. Existing hardware communication
2. Graphic interface hardware
    a. Display
    b. Physical controls
3. Master/Slave arbitration between microgrids

The Network topology is a very important aspect of our project because it dictates how each of the microgrids will communicate with each other. All of the data that we are going to be looking at and modifying will come from this aspect of the project so it is vital that we get it right. Graphic interface hardware will also be very important because it is how the general user will know what the device is doing and if it is functioning properly

4.2.2 Ideation

Leader Selection Algorithm

The idea of leader selection across a distributed system is not novel, so we assumed there would be existing algorithms that are generally accepted by engineers. To find these algorithms we started with a simple Google search for "consensus in distributed systems". This led to the discovery of a family of distributed computing protocols known as Paxos. An example of how basic Paxos works is shown in figure 4.2.2.1.

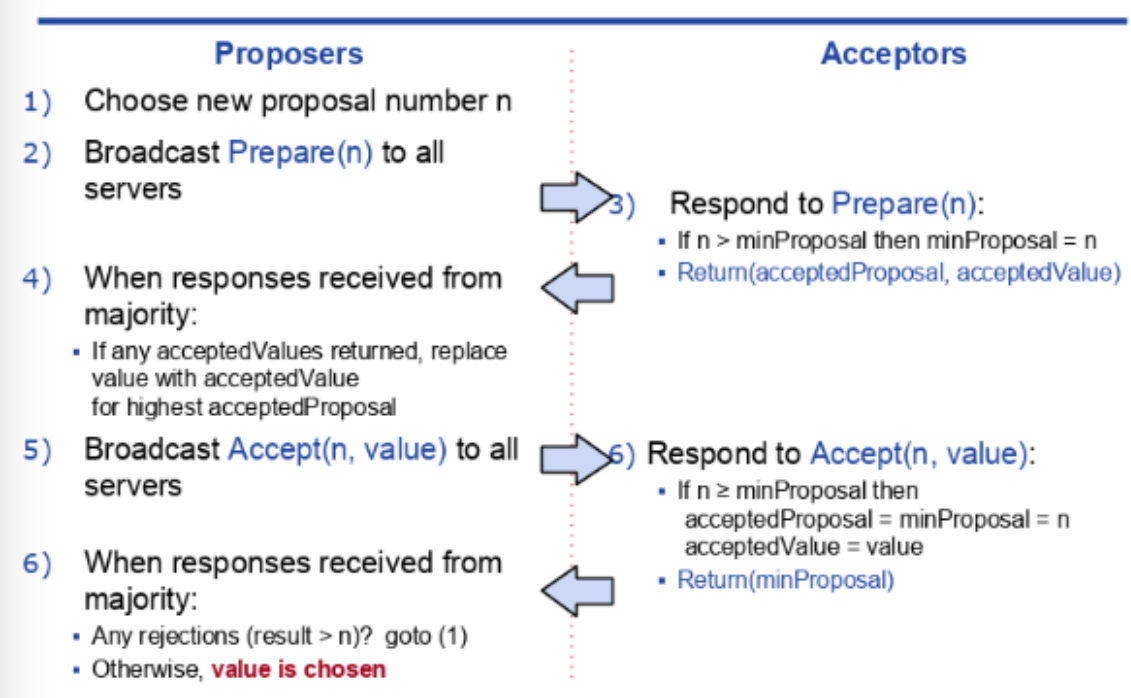

Figure 4.2.2.1: Basic Paxos (L. Pan)

While researching to understand how Paxos is implemented, we discovered that it can lead to livelock situations where controllers are not making progress. In the video "Paxos Simplified" (C. Colohan), they introduced the Bully Algorithm as a simpler alternative without the full implementation of Paxos. During this time we also discovered a list of leader election algorithms in a Wikipedia article. Many of these looked promising such as the Hirschberg–Sinclair algorithm or the Chang and Roberts algorithm.

Display Graphics Library

Display graphic libraries are fairly common and we know from prior experience that there are several different good options out there. We started by first researching GUI libraries that fit our requirements and we discovered four different options that each have their own different pros and cons. In our search we found several different GUIs such as Guizero, PySimpleGUI, Tkinter, Kivy, and Pyside2. in our search we found recommendations for each of these libraries for creating GUIs as each has their own strengths. Guizero, one of the libraries we looked at works by allowing the user to create objects that they can customize to design their UI. It also works to make it easier to

develop by abstracting away the low-level implementation details that can make development difficult or a library difficult to learn.

4.2.3 Decision-Making and Trade-Off

Demonstrate the process you used to identify the pros and cons or trade-offs between each of your ideated options. You may wish to include a weighted decision matrix or other relevant tool. Describe the option you chose and why you chose it.

Leader Selection Algorithm

Upon further investigation, the Hirschberg–Sinclair algorithm and the Chang and Roberts algorithm relied on the network being organized in a ring topology. The IP/TCP organization of our network was going to be set up as a mesh, which eliminated these options early on.

The basic Paxos was also promising, however it was complex to implement since it was meant to have the distributed network agree on any consensus and was not specific to leader election. Basic Paxos also had the possibility of a livelock, which would be especially undesirable for our application since the target user groups would not immediately recognize the problem and be able to fix it.

This left the Bully Algorithm as our design choice for implementation because of its simplicity and reliability.

Display Graphics Library

We came to the conclusion that we want to use Guizero to create our GUI. As it seemed the most reasonable because it has the necessary basic concepts we need for our UI and it seemed the easiest to download and use out of the 5 options. When looking at Tkinter it had a simplistic syntax but the big con for us was that it seemed very outdated and the implementation would be very difficult to do.When looking into PySide2 the pros came out to be that we can use QML and its relatively easy to implement the big con came out to be that there is lack of proper documentation so it would be hard to find examples to help us in the long run. Kivy's has very good pros but again the biggest con is that it has lack of proper documentation and examples so it would be very hard for us to find resources to help us in the long run. Although PySimpleGUI has less cons we found more guides and more tools to help us if we use Guizero. So all that led us to choose Guizero as our choice of library.

Guizero

- Pros
    - Has the basic concepts we need for our UI
    - Really easy to understand the code, seems like just type what you mean
    - With this website there is a lot of code given with proper examples so seems easy to understand enough to where we can implement what we want to the UI
- Cons
    - GUI development on the Raspberry Pi requires an OS version with the desktop. Efforts to develop a GUI interface using the OS Lite version (no desktop) have been fruitless.

- To test we need to run `sudo python3 myGui.py` but honestly doesn't seem that annoying might get annoying if we forget what the command is
- More Simple GUI

PySimpleGUI

- Pros
  - Easy to implement
  - Designed to take the benefits of TKinter, Remi, Qt, and WxPython
  - Several examples given to show how to implement it and a lot of recent support to make implementing it easier
- Cons
  - Needs a prior understanding of python

Tkinter

- Pros
  - It has simplistic syntax and is a default part of python
  - Makes use of simple but powerful widgets
    Lots of documentation and examples of people using it
- Cons
  - Seems to be a bit outdated. Looks like it was the recommend a lot around 2016ish
  - Dos not include some of the more advanced widgets
  - The implementation of widgets can make them hard to debug

Pyside2

- Pros
  - You can still use QML for the UI
  - Installation is relatively easy for the desktop(pip install pyside2)
  - It allows the usage of primary widget-based user interface resources.
- Cons
  - it requires C++ knowledge
  - Qt 3D documentation for the Python is still lacking, tricky to find how to import the namespaces
  - Lack of proper documentation, which leads to difficulty in learning all its insights. PySide requires knowledge of Qt C++ source code in order to understand how to use PySide

Kivy

- Pros
  - Based on Python, which is extremely powerful given its library rich nature.
  - Write code once and use it across all devices.
  - Easy to use widgets built with multi-touch support.
  - Performs better than HTML5 cross-platform alternatives.
- Cons
  - Non-native looking User Interface.
  - Bigger package size (because Python interpreter needs to be included).

- Lack of community support (Kivy Community isn't particularly large).
- Lack of good examples and documentation.
- Better and more community rich alternatives available if only focusing on Mobile Cross-platform devices i.e React Native.

4.3 Proposed Design

4.3.1 Overview

The microgrid pallet communication will be controlled by a Raspberry Pi (RPi) device mounted on each microgrid. This RPi accepts commands from a connected network according to the Tactical Microgrid Standard (TMS) and translates them into messages specific to the microgrid unit. These messages are then forwarded to the microgrid array.

The components used for each microgrid pallet requires communication over a proprietary network to synchronize properly, so the entire microgrid array is configured as a single unit with one leader RPi accepting TMS messages. The leader RPi is responsible for configuration of the entire array, and if it becomes unresponsive a new leader is selected.

The Raspberry Pi additionally maintains an attached display for viewing the status and updating configurations for a microgrid.

4.3.2 Detailed Design and Visual(s)

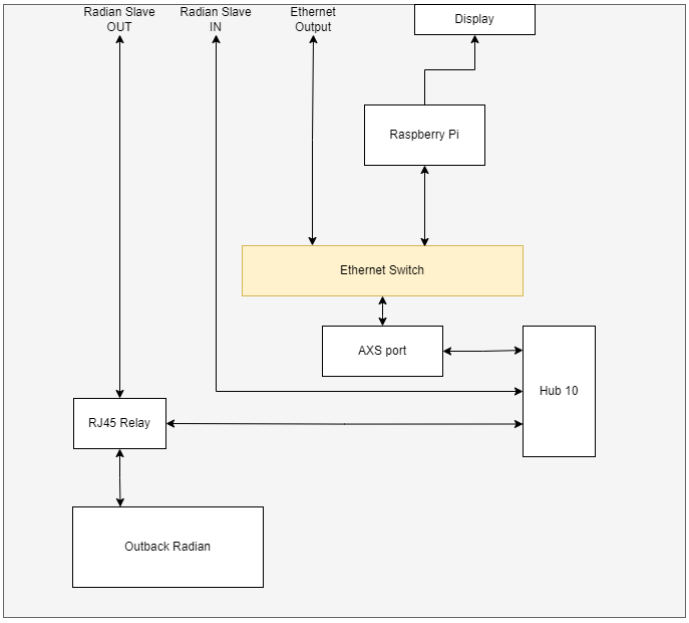

Figure 4.3.2.1: Simplified Controller View of a Single Microgrid

The diagram above shows controller components of a single microgrid unit. This diagram only concerns parts connected to and communicating with the Raspberry Pi and ignores the rest of the system. The Raspberry Pi will be connected to an Ethernet Switch which will allow it to

communicate with the rest of the microgrid and microgrid system. Additional hardware to the existing Microgrid are the Ethernet Switch, Raspberry Pi, Display, and Ethernet Output.

The Raspberry Pi will give commands to the microgrid it is a part of through the use of the AXS port. The AXS port translates our pysunspec2 API code into Outback's proprietary communication protocol to allow the Raspberry Pi to communicate with the rest of the system.

The Ethernet Output will connect the Raspberry Pi to an Ethernet network connecting RPis from each Microgrid in the array. Through this, the leader microgrid's Raspberry Pi will communicate with the others on what they must do while the follower microgrids' Raspberry Pi's will reply with their current status using heartbeat signals. This connection is shown in Figure 4.3.2.2.

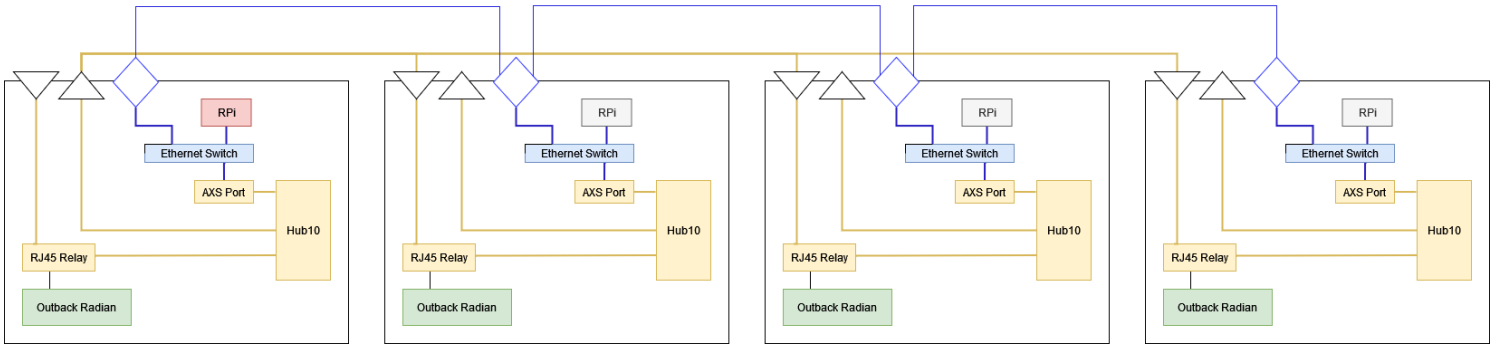

Figure 4.3.2.2: An Array of Four Connected Microgrids with Designated Leader (Left)

We have two established design plans for leader selection and configuration. For each method, a state diagram is shown to describe the leader election process. While in the Follower or Leader loops, the main software will be run. In the Follower state the RPi should receive heartbeats and update its display. In Leader mode, the RPi should follow the software model in Figure 4.3.2.5.

Hardware driven selection

- RPi detects Leader or Follower status based on physical connections
- RPis configure their inverter accordingly

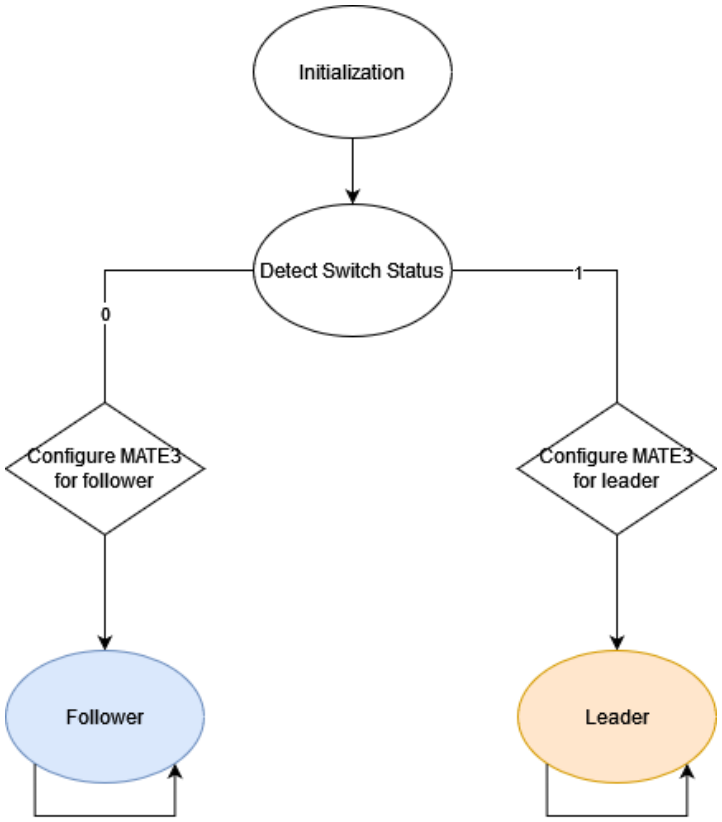

Figure 4.3.2.3: Raspberry Pi Software State Diagram, Hardware Defined Leader

Software driven selection

- RPi network runs the Bully Algorithm for leader selection
- Individual RPis use relays to form proper connections
- RPis configure their inverter accordingly

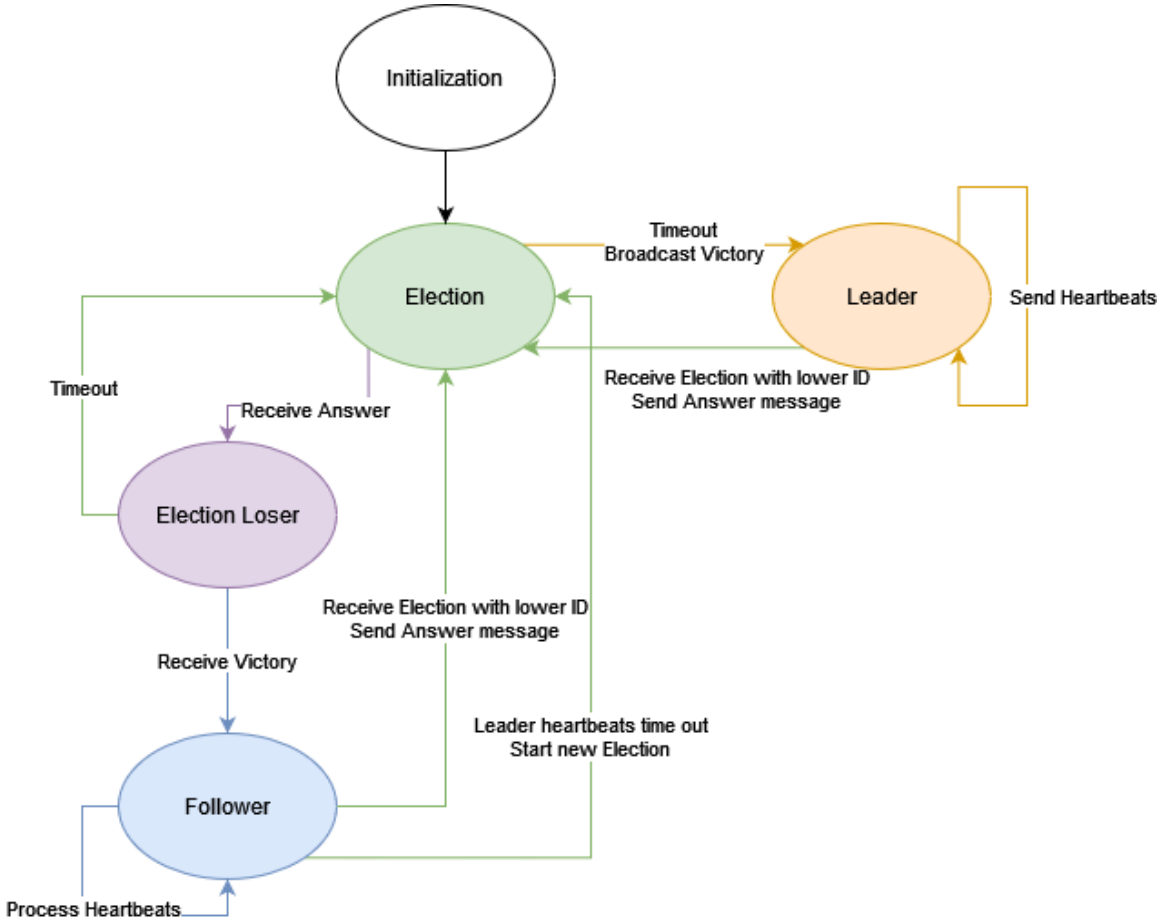

Figure 4.2.3.4: Raspberry Pi Software State Diagram, Software Defined Leader

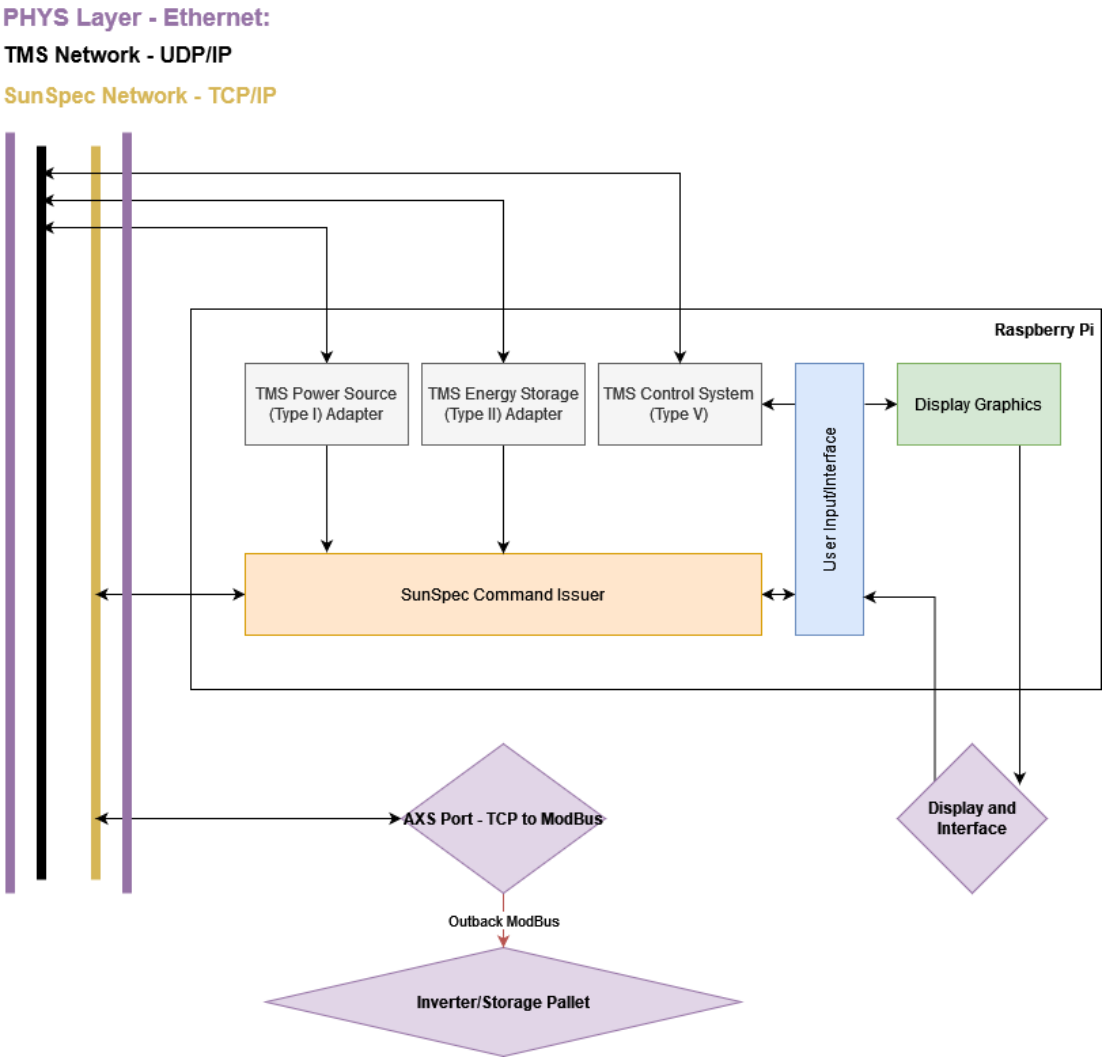

Figure 4.3.2.5: A diagram of the software that will be running on the Raspberry Pi

A significant portion of the software design will be for the leader state. A general breakdown of the necessary software components and interaction is shown in figure 4.3.2.5.

We have split the software into multiple sections to allow for each group member to contribute and work independently of each other. The TMS Power Source Adapter, TMS Energy Storage Adapter, and TMS Control System are all protocols that we must follow when designing this software. The purpose of these protocols is to standardize and meet powergrid system needs.

For the Display Graphics, we have been putting research into various python graphics libraries we could use. The one we are most likely going to use is GUIzero as it is simple and easy to implement which is important for us since most of us have not created applications like this before.

For the SunSpec Command Issuer, we will be using the pySunSpec2 API inorder to send and receive data from the Outback equipment.

MATE 3 Menu	MATE3 Value	Units	What the Value was	Possible Values	Sunspec Name	Sunspec Type	Sunspec Scale Factor	Contents	Sunspec Description
AC Input and Current Limit	Input Priority		Grid	Grid or Gen	GSconfig_AC_Input_Select _Priority	uint16		Programmable	0=Grid, 1=Gen
	Grid Input AC Limit	A	120		GSconfig_Grid_AC_Input _Current_Limit	uint16	GSconfig_AC_Current_SF	Programmable	Grid AC input current limit
	Gen Input AC Limit	A	120		GSconfig_Gen_AC_Input _Current_Limit	uint16	GSconfig_AC_Current_SF	Programmable	Gen AC input current limit
	Charger AC Limit	A	40		GSconfig_Charger_AC_Input _Current_Limit	uint16	GSconfig_AC_Current_SF	Programmable	Charger AC input current limit
Grid AC Input Mode and Limits	Input Mode		Grid Tied	Grid Tied, Generator, Support, UPS, Backup, Mini Grid, Off Grid	GSconfig_Grid_Input_Mode	uint16		Programmable	0=Generator, 1=Support, 2=Grid Tied, 3=UPS, 4=Backup, 5=Mini Grid, 6=Grid Zero
	Voltage Limit Lower Bounds	VAC	108		GSconfig_Grid_Lower_Input _Voltage_Limit	uint16	GSconfig_AC_Voltage_SF	Programmable	Grid Input AC voltage lower limit
	Voltage Limit Upper Bounds	VAC	132		GSconfig_Grid_Upper_Input _Voltage_Limit	uint16	GSconfig_AC_Voltage_SF	Programmable	Grid Input AC voltage upper limit
	Transfer Delay	Cycles	50		GSconfig_Grid_Transfer_Delay	uint16		Programmable	Grid Input AC transfer delay
	Connect Delay	Miuntes	0.2		GSconfig_Grid_Connect_Delay	uint16	GSconfig_Time_SF	Programmable	Grid Input AC connect delay
Gen AC Input Mode and Limits	Input Mode		Grid Tied	Grid Tied, Generator, Support, UPS, Backup, Mini Grid, Off Grid	GSconfig_Gen_Input_Mode	uint16		Programmable	0=Generator, 1=Support, 2=Grid Tied, 3=UPS, 4=Backup, 5=Mini Grid, 6=Grid Zero
	Voltage Limit Lower Bounds	VAC	108		GSconfig_Gen_Lower_Input _Voltage_Limit	uint16	GSconfig_AC_Voltage_SF	Programmable	Gen Input AC voltage lower limit
	Voltage Limit Upper Bounds	VAC	140		GSconfig_Gen_Upper_Input _Voltage_Limit	uint16	GSconfig_AC_Voltage_SF	Programmable	Gen Input AC voltage upper limit
	Transfer Delay	Cycles	50		GSconfig_Gen_Transfer_Delay	uint16		Programmable	Gen Input AC transfer delay
	Connect Delay	Minutes	0.5		GSconfig_Gen_Connect_Delay	uint16	GSconfig_Time_SF	Programmable	Gen Input AC connect delay
AC Output	Output Voltage	VAC	204		GSconfig_AC_Output_Voltage	uint16	GSconfig_AC_Voltage_SF	Programmable	AC output Voltage
	AC Coupled Mode NA		No	Yes or No	OutBack_Radian_AC_Coupled _Control_Mode	uint16		Programmable	0=Disabled, 1=Enabled
Low Battery	Cut-Out Voltage	VDC	36		GSconfig_Low_Battery_Cut_Out_Voltage	uint16	GSconfig_DC_Voltage_SF	Programmable	Low Battery Voltage Cut Out
	Cut-In Voltage	VDC	40		GSconfig_Low_Battery_Cut_In_Voltage	uint16	GSconfig_DC_Voltage_SF	Programmable	Low Battery Voltage Cut In
Battery Charger	Absorb Voltage	VDC	58		GSconfig_Absorb_Volts	uint16	GSconfig_DC_Voltage_SF	Programmable	Absorb Voltage Target
	Absorb Voltage Time		1		GSconfig_Absorb_Time_Hours	uint16	GSconfig_Time_SF	Programmable	Absorb Time Hours
	Float Voltage	V	54.5		GSconfig_Float_Volts	uint16	GSconfig_DC_Voltage_SF	Programmable	Float Voltage Target
	Float Voltage Time		1		GSconfig_Float_Time_Hours	uint16	GSconfig_Time_SF	Programmable	Float Time Hours
	Re-FLoat Voltage	VDC	50		GSconfig_ReFloat_Volts	uint16	GSconfig_DC_Voltage_SF	Programmable	ReFloat Voltage Target
Battery Equalize	Equalize Voltage	VDC	58		GSconfig_EQ_Volts	uint16	GSconfig_DC_Voltage_SF	Programmable	EQ Voltage Target
	Equalize Voltage Time		1		GSconfig_EQ_Time_Hours	uint16	GSconfig_Time_SF	Programmable	EQ Time Hours
Auxiliary Output	Status			Auto, Off	GSconfig_AUX_Control	uint16		Programmable	0 = Off; 1 = Auto; 2 = On
	Aux Mode		Fault	See Sunspec Description	GSconfig_AUX_Mode	uint16		Programmable	1=Load Shed, 2=Gen Alert, 3=Fault, 4=Vent Fan, 5=Cool Fan, 6=DC Divert, 7=Grid Limit/IEEE ,8=AC Source Status,9=AC Divert
Auxiliary Relay	Status			Manual, Off	GSconfig_AUX_Relay_Control	uint16		Programmable	0 = Off; 1 = On; 2 = Auto
	Relay Mode		Vent Fan with 56 VDC and 0.	See Sunspec Description	GSconfig_AUX_Relay_Mode	uint16		Programmable	1=Load Shed, 2=Gen Alert, 3=Fault, 4=Vent Fan, 5=Cool Fan, 6=DC Divert, 7=Grid Limit/IEEE ,8=AC Source Status,9=AC Divert
Inverter Stacking	Stack Mode		Master	Master, Slave	GSconfig_Stacking_Mode	uint16		Read Only	10=Master, 12=Slave, 17=B Phase Master, 18=C Phase Master
	Master Power Save Level		3		GSconfig_Master_Power_Save_Level	uint16		Programmable	Master inverter power save level
	Slave Power Save Level		1		GSconfig_Slave_Power_Save_Level	uint16		Programmable	Slave inverter power save level

Figure 4.3.2.6: Translation between MATE3 menu functions and SunSpec API values

Above is a table describing the relationship between the functions on the MATE3 and their counterparts in the SunSpec API. We will be using these values in order to implement the various functions that the MATE3 is able to perform and read the variables that the MATE3 can see. One concern about this information is that the stack mode for each inverter is a read-only value which means we would have to go about other means in order to change whether a microgrid is considered a master or a slave.

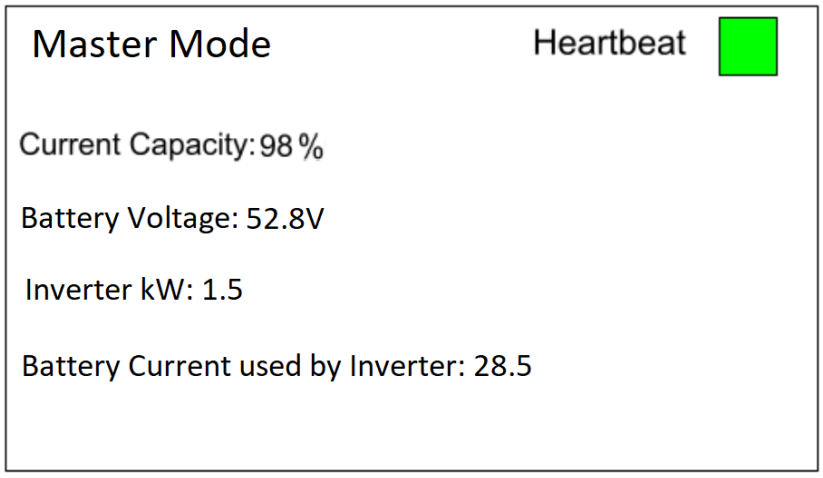

Figure 4.3.2.7: A mockup display menu of the microgrid

The above design is a mockup of a simple screen on the display of the microgrid. The design shows some basic information such as if the microgrids are in master or slave mode, the current battery capacity, battery voltage, inverter kW, and battery current used by inverter. In addition, it also displays the system's heartbeat to make the end user know that the microgrids are communicating correctly.

4.3.3 Functionality

Our design is intended to reduce the amount of user input and technical knowledge needed to operate the microgrids both as a single unit and as an array of units. With our design a user would only need to plug in cables connecting the microgrid units for the microgrids to operate as needed. After being powered on the Raspberry Pi in each microgrid will send signals to the microgrid using the SunSpec API to have it start using the selected mode. The display on each of the microgrids will then show useful information to the user on the state of the microgrid and what it is doing.

4.3.4 Areas of Concern and Development

The current design satisfies our functional requirements for how the microgrid array should function. Our main concern is how we are going to solve the problems that we have run into during the research and development of this project, as are discussed in 4.5.

With our current design, we are using the AXS port to communicate with the existing system through sunspec commands. It has recently come to our attention that we are unable to change the master/slave configurations of the inverters through the AXS port. The manufacturers state that being able to do so is a safety concern. In order to implement most of the proposed design, we would need to be able to make these assignments autonomously. One possible workaround we are discussing is modifying the existing MATE3 hardware to emulate button presses to navigate through menus and change configurations. If we were able to do this, but this functionality was deemed unsafe by the original manufacturer, it may not be ethically sound to take this approach as we are not experts in AC power delivery and power systems. Another option to solve this issue would be to use a relay to switch that would be connected to the Raspberry Pi. This relay would allow the Raspberry Pi to change the master and slave property of the inverter through software.

Another major concern is extensibility of our proposed solutions; under the assumption that we were able to configure a master or slave through the AXS port (we are not), we are limited by the proprietary hardware of the system. When is system is in a master configuration, all communication of that master and any slave is done through the HUB10 on the master unit. The HUB10 requires that the master of the array be plugged into port 1, and slaves plugged into the latter ports individually.

To successfully use a master/slave algorithm, all slave outputs on all pallets must also be connected to a slave input on every other pallet in the network. We are unsure if we can simply split the outputs to other units, or if we need specific hardware/switches to make this happen. Either way this drastically increases the complexity of hardware setup and configuration, which directly goes against the goals of this project.

4.4 Technology Considerations

We will be using a simple display to show the end user various pieces of information about the microgrid's current status. This display will show various variables such as the batteries' charge level, how much power is being output by the microgrid, and how hot the system is running. We will present this information using a Python graphical interface library such as GUIzero. Given that we were assigned to include the least amount of inputs possible for the end user, this display will solely be used to show information to the user and most likely not include any additional menus. The advantage of having no inputs is that it cuts almost all ability for the chance of the user accidentally configuring the microgrid incorrectly. The weakness and trade off of this design choice is that we will have to make sure our code for automating configuration is flawless. If there are any bugs found in our implementation after we finish this project, the user will not have the ability to try and mitigate it in any way unless an update for the software is released..

The Raspberry Pi we are using will communicate with the microgrid using the Sunspec API. This API allows us to view and modify most of the values included in the microgrid's hardware. This is great as it allows an easy way to automatically configure the pallets for the end user. One fault that we found with the API, however, is the fact that we can not toggle whether or not a microgrid is a master or a slave unit. Given that changing from master to slave is a crucial part of the project, this was a big roadblock that we walked into. To solve this issue, we are considering two methods:

- Currently the Microgrids have a dial that toggles a relay between the Master and Slave options by changing which port the inverter is plugged into on the Hub10, Slave or Master. We were thinking about implementing a similar relay into our design that, instead of its input being a dial, it would be connected to the Raspberry Pi to allow us to be able to change which option each microgrid is configured to through software. The advantage of this method is that it is pretty simple. While our group does not have much experience with relays, we have many resources to look at and it could be a good learning experience.
- The microgrid's current proprietary display, the MATE3, is able to change whether the microgrid is considered a master or slave unit within its menus. If we were to open the display and simulate button presses on the unit's buttons by remotely completing the circuit through the Raspberry Pi, we would be able to create software that could execute most every function that the AXS port could accomplish through the SunSpec API as well as being able to change the master/slave value that we currently can not. Problems arise when we realize that we can not read values that the MATE3 is displaying. The trade off of

this design is that we would not be able to read values from the system since the Hub10 only allows either a MATE3 or an AXS port to be connected to it and the MATE3 does not have any output except for its display.

4.5 Design Analysis

We have worked on initial implementations of some of the high-level algorithms running as a proof of concept. This includes figuring out how to communicate with the grids through the AXS port using sunspec commands, as well as master/slave arbitration. Most of the efforts so far have not been successful, and as we learn more about the hardware limitations, we are finding ourselves kind of put in a box.

With this new discovery in hardware limitations, our proposed design from section 4.3 would not work as we cannot configure the Master/Slave settings on the pallet through the AXS port. This is a gigantic hurdle that has only come up in the last couple of days. We are currently working with the manufacturer to learn more about this limitation, why it has been deemed unsafe, and if we can somehow still make these configurations. If we are unable, or it is deemed too unsafe to change it seems like it may jeopardize much of what we can do for the project.

The pallets, once initially configured, will keep their configurations saved through power cycles. So once they are set up there isn't anything that should need to be changed throughout the entirety of their deployment.The parts that make the configuration cumbersome are the connections between pallets, and using the MATE3 to make the proper configuration changes required. Without being able to change the configuration, a pallet would essentially be locked in a slave or master mode, defeating the need to automate anything in the first place.

The hardware that we need to add cannot run on Outback's proprietary network, and requires additional physical connections. These additional connections lead to more failure points, and the potential for user misconfigurations, increasing the complexity. To provide the connection redundancy necessary for a software assigned leader selection, many additional RJ45 ports and cables are necessary due to proprietary communications. This issue is shown in Figure 4.5.1.

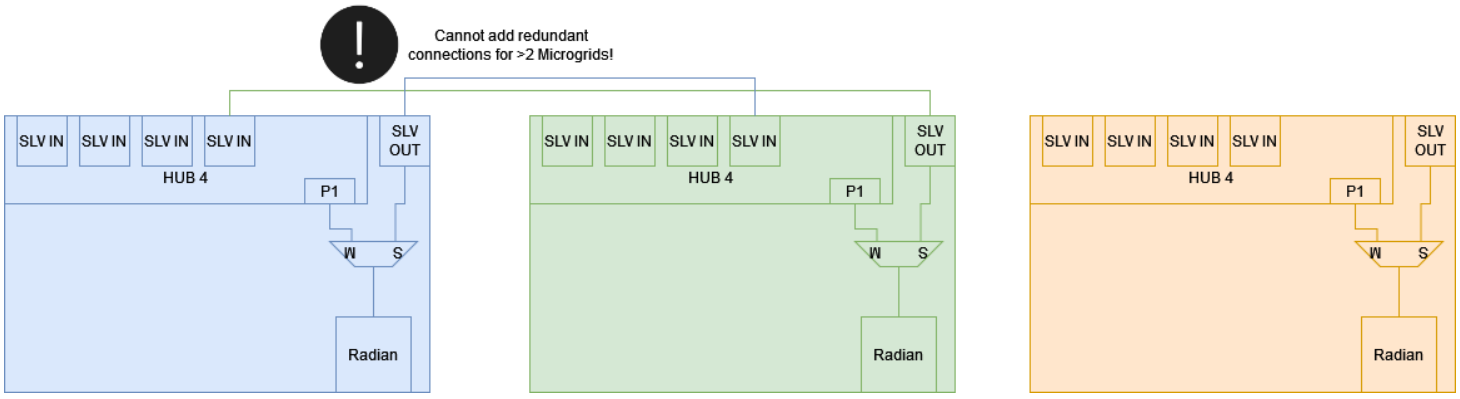

Figure 4.5.1: Unscalable Connection Scheme

5 Testing

5.1 Unit Testing

Three software units that need to be tested - Leader selection, Display & UI, Sunspec Communication

To test these software units, we are going to be using multiple test cases to confirm that our software is functioning correctly.

Unit	**Unit Testing / Method**
Leader Selection	Test algorithm with thread teams before implementing on our Raspberry Pis. After moving the algorithm to Pis, confirm one leader is selected for each possible system change (addition of a device, removal of a leader/follow, one device on the network).
Display & UI	Write unit tests for display software and input software. Correct signals are sent to the Sunspec communication software.
Sunspec Communication	Test various read/write sequences to a single inverter to verify that modifications made are accurate and reliable. Do sequences for each mutable register. Run SunSpec compliance tests as provided by Sunspec alliance.

Table 5.1.1: Software Testing Units

Hardware Units for testing - Network topology, Display & UI (Button input/ Display functional), Microgrid Outputs/Inputs

Unit	**Unit Testing / Method**
Network Topology	Ping devices across the network to ensure connections. Send a series of messages across the network and assert successful acknowledgement. Use basic command line pings.
Display & UI	Create software tests to probe input from individual buttons or other inputs. Confirm these are as expected. Example test: Connect button to RPi GPIO Press button, verify that correct response occurred (print message, etc.)
Inverter Configurations	Will discuss with PowerFilm contacts on how to test hardware to see if it works. Since we do not have much experience in power systems, it would be hard for us to come up with a testing environment for this hardware.

Table 5.1.2: Hardware Testing Units

5.2 Interface Testing

GUI Hardware Testing

We will likely implement a test screen where a user can manipulate the control hardware, and see whether or not it is working properly. We are unsure how to implement testing beyond using the physical hardware and seeing if the system responds appropriately.

Raspberry Pi to Raspberry Pi Communication (TCP/IP)

Here we will have clearly laid out parameters designating our packet information. We should be able to check incoming packets, and make sure they are valid before the microcontrollers take any action with the data. We may use something like modbus as a protocol to follow for this communication.

Raspberry Pi to Sunspec Command/AXS (TCP/IP)

The AXS port is meant to receive sunspec commands, if an invalid command is created we should get notified by the AXS port, and/or we will be able to observe improper/missing configurations. We will also write sanity checks to verify that messages being sent are of the correct format and in accordance with the sunspec API. The sunspec API will most likely have some of this functionality built in.

AXS Port to Inverter/Microgrid (Modbus/Sunspec)

This connection/communication is largely proprietary and will be difficult to test. The communication here isn't necessarily defined by our team, but we will want to make sure it is working properly by verifying that our messages are being received by the microgrid. We can do this through simply making configuration changes and reading these messages back. Modbus should

send responses with codes that will verify changes that are received and let us know if it was successful (or alert us if there are issues)

5.3 Integration Testing

The critical integration paths in our integration testing are making sure the User Interface fits well with the rest of the codebase. While we have been assigned to program all the features of the old system into a more modern design, we have also been assigned to make a very user friendly interface. That being said, the user interface seems to be one of the project's main actors and it's important that we put a lot of attention into making sure all of the features of the interface display correctly and call the correct methods. We must also make sure changes to configuration made through our UI are enacted across all applicable components of the system. The scope of configurations we will provide has not yet been specified, but we will most likely verify the results manually across parts of the system.

5.4 System Testing

System level testing is end to end testing making sure the whole system works. So in our situation making sure that the microgrids are able to talk to each other with a simple test. We would have to write up some unit tests that check the code, where it shows that both the microgrids will be able to talk with each other. We will also need to verify that the commands being sent are in fact being received and the proper configuration changes are being enacted.

For system level testing, our project goals are to make the microgrid work in tandem with minimal user interaction. To test this requirement, we will create a simple instruction set for connecting the pallets. We will ask people without experience with the project to attempt to set them up.

5.5 Regression Testing

To ensure new additions do not cause regressions, we will run tests on commits merged to the main branch making sure the application still works the same way. The most important features we will make sure are intact are the backend SunSpec Management methods and the potential TMS interface methods. These features are vital to the project and we must make sure that they are 100 percent functional. We will do this through building a Continuous Integration/ Continuous Delivery system into our GitLab.

5.6 Acceptance Testing

We will have branches corresponding to every feature that we will be adding to the project. Alongside that, we will be using GitLab issues and milestones to keep track of our overall progress in the project. The descriptions of these issues and milestones alongside any changes made to the original plan during development will be relayed to the client during our meetings with them. We will also provide images of any major milestones or problems encountered so that the client is up to date as much as possible. We will discuss all of what is mentioned above as well in our weekly meetings with the client so that we can answer any questions they might have about the topic.

5.7 Security Testing (if applicable)

Not applicable to this project.

5.8 Results

Our testing up until this point involves learning which functions we are able to execute using the SunSpec API, experimenting which graphics library would be best suited for the application, as well as, seeing which leader election algorithm would be the best to use in this scenario.

By reading and writing values through the AXS port and SunSpec API, we have found that we are able to do as much and more than what the current display, the MATE3, is able to do. The only problem we have run into is that we found that the inverter stacking mode variable provided by the SunSpec API is unable to be written to which means we are unable to change whether a microgrid in an array is considered the master or the slave through software. We are currently considering alternatives for this issue which have been described in section 4.4 of this document.

We have also run tests to determine the best choice for the leader election algorithm

6 Implementation

Describe any (preliminary) implementation plan for the next semester for your proposed design in 3.3. If your project has inseparable activities between design and implementation, you can list them either in the Design section or this section.

- Successful basic configuration of a single microgrid
- First test communication between two microgrids
- Reliable master/slave determination
- Fully SunSpec compliant
- Full stable, autonomous communication between two microgrids
- Fully TMS compliant

7 Professional Responsibility

This discussion is with respect to the paper titled " Contextualizing Professionalism in Capstone Projects Using the IDEALS Professional Responsibility Assessment", *International Journal of Engineering Education* Vol. 28, No. 2, pp. 416–424, 2012

7.1 Areas of Responsibility

Area of responsibility	Definition	NSPE Canon	Society-Specific Code	Difference between IEEE and NSPE
Work Competence	Perform work of high quality, integrity, timeliness, and professional competence.	Perform services only in areas of their competence; Avoid deceptive acts.	IEEE Code I.6	The IEEE code also covers undertaking technological tasks only if one has training or experience in the topic.
Financial Responsibility	Deliver products and services of realizable value and at reasonable costs.	Act for each employer or client as faithful agents or trustees.	IEEE Code II.2	The IEEE focuses less on the value of the product and more on the understanding of the product's value by others.
Communication Honesty	Reports work truthfully, without deception, and are understandable to stakeholders.	Issue public statements only in an objective and truthful manner; Avoid deceptive acts.	IEEE Code I.5	The IEEE code agrees with the NSPE while also talking about how one should seek out criticism and give credit for their ideas.
Health, Safety, Well-Being	Minimize risks to safety, health, and well-being of stakeholders.	Hold paramount the safety, health, and welfare of the public	IEEE Code I.1	The IEEE code also mentions to not hurt another's reputation and property as well as what the NSPE says.
Property Ownership	Respect property, ideas, and information of clients and others.	Act for each employer or client as faithful agents or trustees.	IEEE Code I.9	The IEEE code focuses as well on not damaging another's reputation or injuring them.
Sustainability	Protect environment and natural resources locally and globally		IEEE Code I.1	IEEE also talks about how making sure one engages in ethical practices.
Social Responsibility	Produce products and services that benefit society and communities	Conduct themselves honorably, responsibly, ethically, and lawfully so as to enhance the honor, reputation, and usefulness of the profession.	IEEE Code II.7	The IEEE focuses less on producing products that benefit society and more on making sure those products do not harm others.

Table 7.1: NSPE and IEEE Code of Ethics Breakdown

7.2 Project Specific Professional Responsibility Areas

Area of responsibility	Does it apply to the project?	How Important is it?	Why?
Work Competence	Yes	High	We need to not only make sure our project works but is high quality because, if not, we could end up damaging components or causing our system to not work when it is needed.
Financial Responsibility	Yes	Medium	We have been given a budget and are expected by the client to stick to that budget. We need to make sure that, when making a purchase for this project, we know that it will be impactful.
Communication Honesty	Yes	Medium	It is important that the stakeholders know not only what is going well with the project but also what isn't. Without the latter, our problems will only magnify and will show themselves sooner or later.
Health, Safety, Well-Being	Yes	High	Throughout this project, we are going to be working with high amounts of electricity. If we are careless, we could seriously end up hurting a team member or end user. To make sure that does not happen, we need to approach this issue with utmost caution.
Property Ownership	Yes	High	We need to make sure that we are working within the requirements and constraints set by the client or else there is no point in us working on this project in the first place.
Sustainability	Yes	High	Sustainability is what this project is all about. Microgrids can be seen as green alternatives to generators. The only barrier to entry is the setup. If they are made more user-friendly, we could end up replacing pollutant spewing generators and make the world more green.
Social Responsibility	Yes	High	The very idea of these microgrids is about helping those in need. Whether it be the military or disaster relief, microgrids purpose is to be helpful to society. With our interface automation, it'll make them that much more useful.

Table 7.2: Professional Responsibilities in our Project Context

7.3 Most Applicable Professional Responsibility Area

The most applicable professional responsibility area to our project is social responsibility. If we are making this product and it is not beneficial to the end user who will be using it, then it may as well be useless. These microgrids are going to be used to allow people to help wherever needed and save lives. If our configuration automation does not work in the end or works in a way that makes it not helpful to those who may need it, it would cost people a lot more than just time and money.

8 Closing Material

8.1 Discussion

Discuss the main results of your project – for a product, discuss if the requirements are met, for experiments oriented project – what are the results of the experiment, if you were validating a hypothesis – did it work?

For our project we have not made any material results as of yet. For this project so far we have created plans to implement the Raspberry Pi into the existing microgrid system and have shown that we are able to communicate with the OutBack Power systems. In addition to that we have begun to learn SunSpec and how to implement it to control the OutBack Power system.

We have discovered a number of roadblocks that interfere with the original design plan. These roadblock include:

- Proprietary OutBack Power communications require connection through their Hub4, which is more limiting than a switch. This leads to software assigned leader and follower roles to be non-scaleable with an increasing number of microgrids.
- For safety reasons, the stacking configuration of the inverters (Leader / Follower assignment) cannot be performed remotely through SunSpec (it is a protected Read Only value). We do not know if there is a work-around for this issue.

8.2 Conclusion

Summarize the work you have done so far. Briefly reiterate your goals. Then, reiterate the best plan of action (or solution) to achieving your goals. What constrained you from achieving these goals (if something did)? What could be done differently in a future design/implementation iteration to achieve these goals?

So the work we have completed so far is mainly design work. Our proposed design with our Raspberry Pis and how we are going to connect the microgrids. Our goals thus far would be having a clear cut plan on how we are going to properly implement our design. So we have done a good job on that part of our goal because our designs were thoroughly explained. The best plan of action was coming together each week and showing our work to our group. Everyone does a little check on the design and then we give the okay to make sure it should work, it was the divide and conquer strategy but we came together at the end to have a quick understanding of what everyone did. Nothing really constrained us from achieving the goals because there were 5 of us if one person couldn't really do as much work as they could have the rest of the group were able to keep the work

going. Better planning and making sure everyone knows their schedule ahead of time so that there won't be any confusion on when people are going to do their work.

8.3 References

Technical References: Nick David(Client), Mathew Wymore(Advisor)

C. Colohan, “Paxos Simplified,” Dec, 2017. [Online]. Available: https://www.youtube.com/watch?v=SRsK-ZXTeZo

L. Pan, “Paxos at its heart is very simple,” Nov, 2018. [Online]. Available: https://blog.the-pans.com/paxos-explained/

M. Bush, J. Corman, and B. Fox, “SunSpec Energy Storage Models.” .

“SunSpec DER Information Model Specification.” SunSpec Alliance, San Jose.

“SunSpec Device Information Model Specification.” SunSpec Alliance, San Jose.

“SunSpec Modbus IEEE 1547-2018 Profile Specification and Implementation Guide.” SunSpec Alliance, San Jose.

“SunSpec Technology Overview.” SunSpec Alliance, San Jose.

8.4 Appendices

Any additional information that would be helpful to the evaluation of your design document.

If you have any large graphs, tables, or similar data that does not directly pertain to the problem but helps support it, include it here. This would also be a good area to include hardware/software manuals used. May include CAD files, circuit schematics, layout etc,. PCB testing issues etc., Software bugs etc.

8.4.1 Team Contract

Team Members:

1) Andrew Frank 2) Christian Pinta

3) Austin Thoreson 4) Ben Eder

5) Saketh Jonnadula

Team Procedures

1. Day, time, and location (face-to-face or virtual) for regular team meetings:

   Weekly: Fridays 3:30 - 4:30pm (Discord)
   Bi-Weekly (With faculty): Thursday 2:00 - 4:00
   Meetings may be held in person as the project progresses, and we have physical hardware

2. Preferred method of communication updates, reminders, issues, and scheduling (e.g., e-mail, phone, app, face-to-face):

   Discord for main communication, Issues are tracked in GitLab, scheduling using when2meet.com. Documentation and communication with faculty may be sent through email.

3. Decision-making policy (e.g., consensus, majority vote):

   We will use the majority rule to make decisions when there are disagreements.

4. Procedures for record keeping (i.e., who will keep meeting minutes, how will minutes be shared/archived):

   Meeting minutes will be recorded each meeting, stored in shared google drive

Participation Expectations

1. Expected individual attendance, punctuality, and participation at all team meetings:

Expected attendance at each meeting unless specified 24 hours in advance in the Discord. If you have to miss a meeting, do a status update in Discord and read the meeting minutes afterwards.

2. Expected level of responsibility for fulfilling team assignments, timelines, and deadlines:

Team members will typically be assigned to tasks individually or in groups, and should expect to complete the task by deadlines determined by the team. If you won't be able to meet a deadline, let the rest of the team know ahead of time so we can figure out a backup plan.

3. Expected level of communication with other team members:

Establish communication with the group at least once a week.Communicate when tasks are completed or can't be completed, especially if it affects other timelines.

4. Expected level of commitment to team decisions and tasks:

Team decisions should be followed, and communication is necessary to change plans.

Leadership

1. Leadership roles for each team member (e.g., team organization, client interaction, individual component design, testing, etc.):

Section off project and assign it to team members as we learn more. To start, we will all communicate with the client and in the future may move to a single or few POC.

2. Strategies for supporting and guiding the work of all team members:

Put tasks up on the Git Board. Assign tasks during meetings. Set deadlines / milestones for tasks on assignment. Ask for help if you need it.

3. Strategies for recognizing the contributions of all team members:

Keep track of contributions through meeting minutes and Git board.

Collaboration and Inclusion

1. Describe the skills, expertise, and unique perspectives each team member brings to the team.

   CPRE: More experience with communication protocols and hardware.

2. Strategies for encouraging and supporting contributions and ideas from all team members:

Don't talk over other members during discussions and make sure everyone has a chance to speak during decisions.

3. Procedures for identifying and resolving collaboration or inclusion issues (e.g., how will a team member inform the team that the team environment is obstructing their opportunity or ability to contribute?)

If you have a problem, inform the team. If there is some internal team conflict, get in touch with Nick Fila.

Goal-Setting, Planning, and Execution

1. Team goals for this semester:
    a. Gain a good understanding of what needs to be done to complete the project and develop an action plan.

2. Strategies for planning and assigning individual and team work:
    a. Assign tasks and deadlines during team meetings and put it on the Git board.

3. Strategies for keeping on task:
    a. Reduce distractions during meetings. Create meeting agendas for each week (found on meeting minutes)

Consequences for Not Adhering to Team Contract

1. How will you handle infractions of any of the obligations of this team contract?

For a first offense, discuss with the person as a group. If issues are repeated or the offender goes MIA, bring it up with Matt Wymore.

2. What will your team do if the infractions continue?

Contact Nick Fila and Matt Wymore to discuss consequences.

****

a) *I participated in formulating the standards, roles, and procedures as stated in this contract.*
b) *I understand that I am obligated to abide by these terms and conditions.*
c) *I understand that if I do not abide by these terms and conditions, I will suffer the consequences as stated in this contract.*

1) Austin Thoreson	DATE: 09/16/2022
2) Christian Pinta	DATE: 09/16/2022
3) Andrew Frank	DATE: 09/16/2022
4) Ben Eder	DATE: 09/16/2022
5) Saketh Jonnadula	DATE: 09/16/2022