# Department of Computer Science & Engineering
# The University of Texas at Arlington


## Detailed Design Specification
## CSE 4317: Senior Design II
## Spring 2024

# CAN on Wheels
# Modular EV


Edwin Diaz
Carson Fabbro
Leonardo Ibarra
Destiny Rogers
Hector Sosa

## Revision History

| Revision | Date | Author(s) | Description |
|---|---|---|---|
| 0.1 | 2.20.2024 | DR | document creation |
| 0.2 | 3.07.2024 | DR | framework established |
| 1.0 | 3.26.2024 | LI, HS, ED, CF, DR | finished draft |

# CONTENTS

## List of Figures



## List of Tables

# 1 Introduction

The Modular Electric Vehicle is a platform designed for engineers to easily prototype and test the embedded systems inside of an electric vehicle by providing an easy method of adding and replacing various computer modules without rebuilding the entire machine. It is intended that engineers from many different disciplines use this platform to better understand the roles of their peers and test systems of various kinds with ease. The Intercommunication system (from here on referred to as "The CAN") of the vehicle will allow for computer interfacing with the machine and the connection of different modules, while other critical systems such as the Battery Management, Motor Control, and Security systems will serve as the baseline of the machine's functioning and provide access points for other modules.

These four systems will serve allow the vehicle to achieved all its major requirements: communication, safety, movement, and efficiency. The robust CAN bus communication allows these systems to communicate effectively without the risk of interference from external sources.

# 2 System Overview

The CAN-on-Wheels Modular Electric Vehicle contains four major systems: The Battery Management system (BMS), Intercommunication system, Motor Control system, and Security system. The BMS serves to manage power to the rest of the system. It controls the charging rate of the vehicle, protects the vehicle from power surges, and prevents the vehicle from overheating. The Intercommunication system (also called "The CAN") captures diagnostic data and allows for modules to be attached and detached from the vehicle. It is the main interface for developers to interact with the vehicle's built-in systems. The Motor Control system guides the motion of the vehicle. It uses speed data and user input to determine the rate at which to change the motors' velocity. The Security system keeps the vehicle from being stolen and allows for better control over the vehicle for safety purposes. It can control the vehicle remotely, make it immobile, lock and unlock compartments of the vehicle, and set off an alarm.

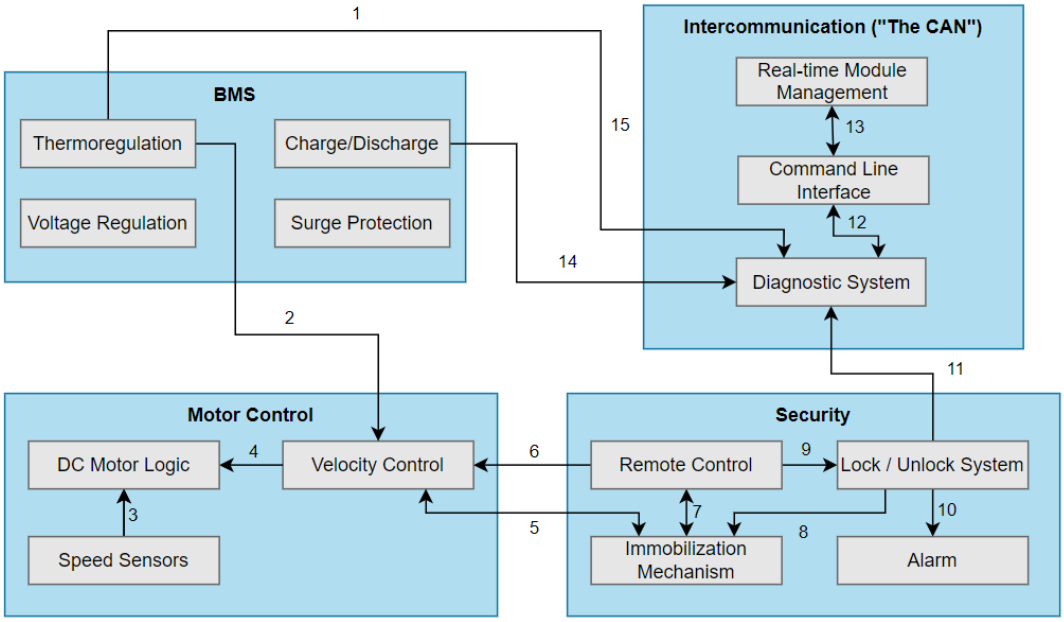


Figure 1: System Architecture

# 3 Intercommunication Layer Subsystems

The Intercommunication System Module, also known as the "CAN" layer, is the central CAN node that will monitor all other modules in real time and provide an interface for the user. The module has a software framework of multiple tasks that give the user the abilities to display critical information from the Modular EV, add or remove modules from the Modular EV, and interface with a specific module. The Interface Communication System uses the CAN Bus to communicate with other modules, so a CAN module driver for the TM4C123GXL LaunchPad Evaluation Board was designed to provide initialization, read, and write abilities for the internal CAN controller. The CAN driver also provides a CAN message structure which allows for programming a CAN message object along with storing any data to be sent or received.

## 3.1 Layer Hardware

The Intercommunication System Module runs on a Tiva C Series TM4C123GXL LaunchPad Evaluation Board with a TM4C123GH6PM MCU. The MCU is an ARM Cortex-M4-based microcontroller with 256 kB Flash memory, 32 kB SRAM, 80 MHz operation, and multiple peripherals including UART, CAN, PWM, and more. Additionally, the system uses an MCP2561 high-speed CAN transceiver and two 120 ohm termination resistors as the TM4C123GXL Evaluation Board does not provide an interal transceiver for the CAN controller.

## 3.2 Layer Operating System

The Intercommunication System runs on a custom-designed Real-Time Operating System for the TM4C123GXL Evaluation Board.

## 3.3 Layer Software Dependencies

A description of any software dependencies (libraries, frameworks, etc) required by the layer. As all modules communicate through the CAN Bus, the Intercommunication System uses CAN drivers designed based on TivaWare's C device peripherals library.

## 3.4 Command Line Interface

The Command Line Interface is the main interface that provides a set of commands and a terminal for the user to use to interact with the Modular EV. The interface wil run as a low priority task on the module using the UART peripheral and a unique data structure and parsing algorithm to transmit and receive command instructions. The set of commands will allow to user to view diagonistic information from the Modular EV, view the current Modules installed on the Modular EV, and handle certain interfaces from other modules.

### 3.4.1 Subsystem Hardware

The Command Line Interface does not implement any hardware as it is a task running on the TM4C123GXL Evaluation Board that has access to the UART peripheral.

### 3.4.2 Subsystem Operating System

A description of any operating systems required by the subsystem. The Command Line Interface is a single task running

### 3.4.3 Subsystem Software Dependencies

Because the Command Line Interface uses a UART-based communication with the user, drivers for the UART peripheral designed by Dr. Losh are used along with an open-source serial console such as PuTTY.

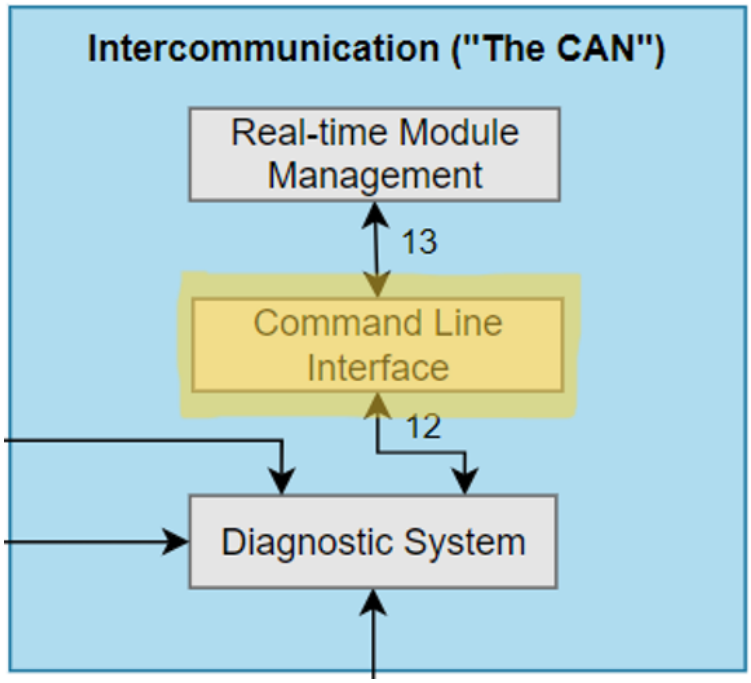


Figure 2: Command line interface subsystem description diagram

#### 3.4.4 Subsystem Programming Languages

The Command Line Interface is entirely written in C.

#### 3.4.5 Subsystem Data Structures

To avoid buffer overloads and to implement a light-weight interface, a data structure designed by Dr. Losh is used to contain the UART data retrieved from the console. The data stucture is a simple struct containing a string buffer the size of the maximum amount of characters that the console can type out, a field counter variable that counts how many fields were found in the buffer, a field position array containing the indicies that will point to the correct position of each field found, and a character array that indicates whether or not the field is a number or an ascii character. These buffers will be used for retrieving the data from the serial console and parsing the data into fields without the use of any C string libraries.

#### 3.4.6 Subsystem Data Processing

Because the interface needs to be light-weight and compact, a simple strategy designed by Dr. Losh is used to retrieve and format every single character from the terminal console, store it into the data buffer, parse the data into several fields, and verify that the command is valid.

The process for getting the command from the user begins with a counter variable initialized to 0, indicating that the string buffer is empty. An infinite loop is set where the first character from the console is retrieved. If the character retrieved is a backspace and the string is not empty (counter is greater than 0), delete the character by decrementing the counter. If the character retrieved is a carriage return or line feed, add the null terminator byte and return. If the character is a printable character (an ascii value greater than or equal to 32), store the character into the data buffer and increment the count. However, if the maximum amount of characters has been reached, add the null terminator byte and return.

The parsing algorithm begins with the idea of grouping characters into 3 groups: Alphabetical characters, numbers, and delimiters. The start of the parsing sets the field counter to 0 and the previous character as a delimiter. The algorithm then loops through the entire string buffer and begins to break the buffer into different fields by converting the delimiters into null terminator bytes. The strategy is if the current character is an alphabetical character and the previous character was a delimiter, then it is recognized as a field and its position in the string buffer and its field type is saved using the data structure fields. The same idea is used for number characters. This process is done to convert the string buffer into multiple strings that can be accessed anytime.

The command validation is done using a custom-made compare function in which it returns true if each character on the command string matches with the requested command by the user. The validation can be switched to be case-sensitive depending on the specifications of the project.

## 3.5 Diagnostic System

The Diagnostic System is a specific task in the Intercommunication System that stores and formats vital information about the Modular EV. It receives the vital information from several modules via the CAN bus and can send the contents of it's table to the user via the command line interface to be displayed.

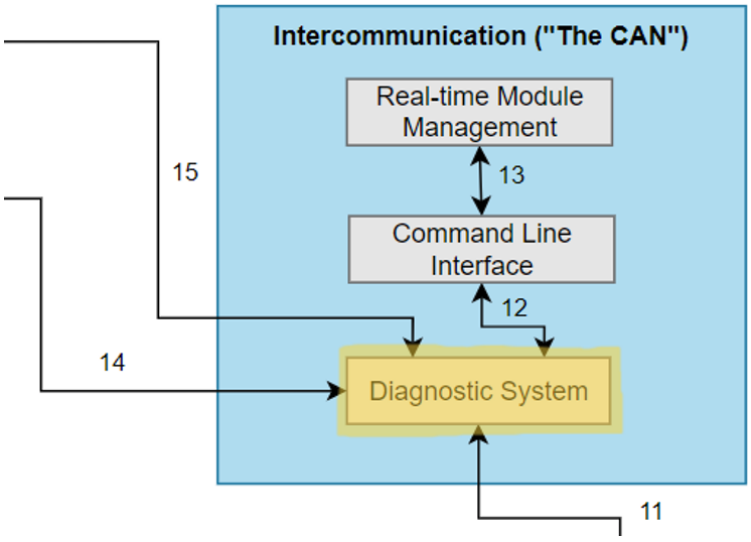


Figure 3: Diagnostic system subsystem description diagram

### 3.5.1 Subsystem Hardware

This subsystem does not implement any hardware as it is entirely software dependent run on the TM4C123GXL Evaluation Board.

### 3.5.2 Subsystem Operating System

The Diagnostic System is a assortment of tasks running on the custom Real-Time Operating System being used for the TM4C123GXL Evaluation Board.

### 3.5.3 Subsystem Software Dependencies

The Diagnostic System depends on the reading CAN bus task which is in charge of reading the CAN bus and returning the data read to all dependable sublayers of the Intercommunication System.

### 3.5.4 Subsystem Programming Languages

The Diagnostic System is entirely written in C.

### 3.5.5 Subsystem Data Structures

The Diagnostic System simply uses a table of 32 bit words similar to how the internal EEPROM of the TM4C123GH6PM MCU operates. The table is initialized at the start of the program with a fixed size which can be changed before runtime by the user.

### 3.5.6 Subsystem Data Processing

Because the Diagnostic System receives it's data via the CAN bus, it needs to process the returned message object structure from the reading task. The data returned will be stored in the message object structure's data field along with the length of the data. Depending on the data type, the returned data will be stored in a single entry of the table or split into multiple entries in the table.

## 3.6 Real-Time Module Management

The Real-Time Module Management subsystem is a specific set of tasks in the Intercommincation System's RTOS that is in charge of monitoring the status of all running Modules and allowing the user to interface with these modules. This task will be able to send messages to other modules with administrative instructions such as stopping the selected Module's activities or adding new modules to the system.

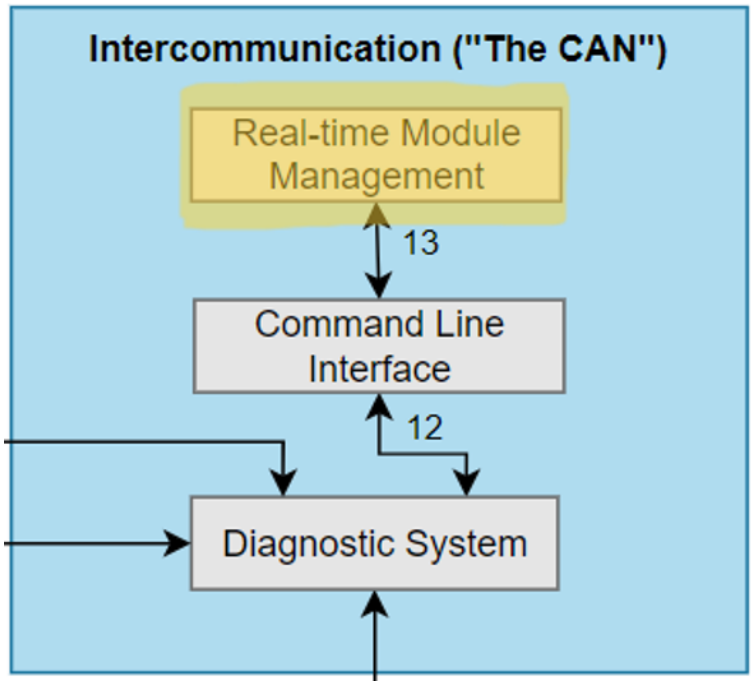


Figure 4: Real-time module management subsystem description diagram

### 3.6.1 Subsystem Hardware

This subsystem does not implement any hardware as it is entirely software dependent run on the TM4C123GXL Evaluation Board.

### 3.6.2 Subsystem Operating System

The Real-Time Module Management subsystem is a set of tasks running on the custom Real-Time Operating System being used for the TM4C123GXL Evaluation Board.

### 3.6.3 Subsystem Software Dependencies

The Real-Time Module Management subsystem is entirely dependent on the read and write tasks that are in charge of sending CAN messages to other modules. As this subsystem will both send and receive messages from other modules, it also depends that each module is using the same framework used by the Intercommunication System.

### 3.6.4 Subsystem Programming Languages

The Real-Time Module Management subsystem is entirely written in C.

### 3.6.5 Subsystem Data Structures

The Real-Time Module Management subsystem does not use any specific data structures except for the provided CAN message object structure which allows the subsystem to send messages through the CAN bus along with reading CAN messages.

### 3.6.6 Subsystem Data Processing

Because the Real-Time Module Management subsystem handles the addition and removal of modules, its data processing is relatively simple as it either processes a CAN message from the receiver task or constructs a CAN message and sends it over to the transmit task.

# 4 Battery Management Layer Subsystems

The Battery Management system is the power and temperature handling hub of the machine. The implementation idea is to periodically and uniformly read and monitor all things related to power. This includes temperature readings of all main motors, batteries, and compartments. Additionally, the BMS also reads and monitors voltage levels from the batteries and current readings from the motors. After this information has been gathered, the battery life is calculated using the current readings and subtracting the energy consumption from the current amount of energy. A distress message is sent to the Motor Control system if high heat is detected to reduce the load and lower the temperature of the overheating motor. Temperature and battery data is also reported to diagnostics. Lastly, charging and discharging of the batteries is handled by utilizing voltage and temperature readings.

## 4.1 Layer Hardware

The Battery Management system module's hardware specifications include a suite of analog-to-digital converters, temperature sensors, thermocouples, a switching circuit, and voltage divider circuits. The analog-to-digital converters are used to get voltage values from the batteries to access the charging state and, paired with the thermocouples and temperature sensors, to get temperature readings of the 3 compartments of the vehicle, all batteries, and all driving motors. The switching circuit is used to enable or disable load power as needed and lastly, the voltage divider circuits are used to scale down voltages to be able to serve as inputs to the analog-to-digital converters as needed.

## 4.2 Layer Operating System

This module utilizes a lightweight, custom real-time operating system with the intention of efficient and effective task executions.

## 4.3 Layer Software Dependencies

Software dependencies for the Battery Management system are the expectation of valid command inputs sent from the CAN network to the Battery Management system. Subsystems should act predominantly independently excluding the BMS reporting to Motor Control. Additional dependencies include the calibration of the buck converter, which provides a stable 5 volt source voltage to all our systems. The vehicle depends on the voltage deviation staying within +/- 0.5 volts or risk abnormal or faulty behavior. The system depends on the summation of all module logic's current draw to be under 20 Amps which is a limitation of the buck converter.

## 4.4 Voltage Regulation

The Voltage Regulation subsystem provides stable power sources when translating between voltage levels. For example, a key role of voltage regulation is to be able to drop down voltages to a compatible range such as 5 volts for logic-level devices. It is a piece of hardware based on the LM2596S.

### 4.4.1 Subsystem Hardware

Hardware includes the sole utilization of a buck converter based on the LM2596S chip series.

### 4.4.2 Subsystem Software Dependencies

Dependencies of the voltage regulation subsystem are valid input voltages and proper device calibration by means of adjusting the voltage regulators' potentiometer. The voltage regulators internally adjust the output voltage by changing the impedance of the circuit resulting in a linear chance in voltage. Calibration includes measuring output voltage and adjusting the potentiometer until 5 volts is measured. The voltage must not drift past 0.5 volts and depends on the user to monitor if voltage drifts occasionally and to recalibrate if necessary.

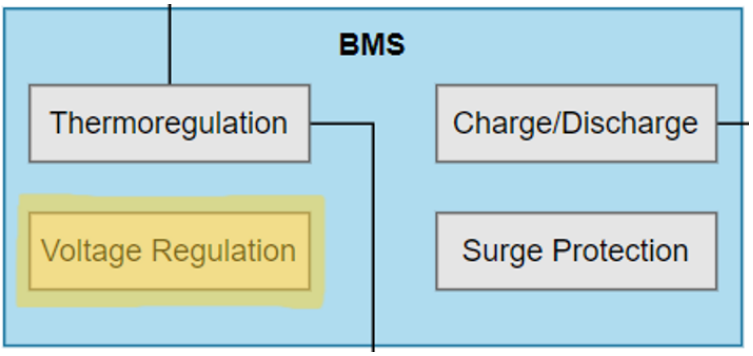


Figure 5: Voltage regulation subsystem description diagram

## 4.5 Surge Protection

The BMS protects against power surges by isolating the different voltage levels such as motor loads, and the different battery levels. In addition we also have diodes in place to respond to currents and fuses to prevent worst case scenarios where the power produced is harmful to the system as a whole.

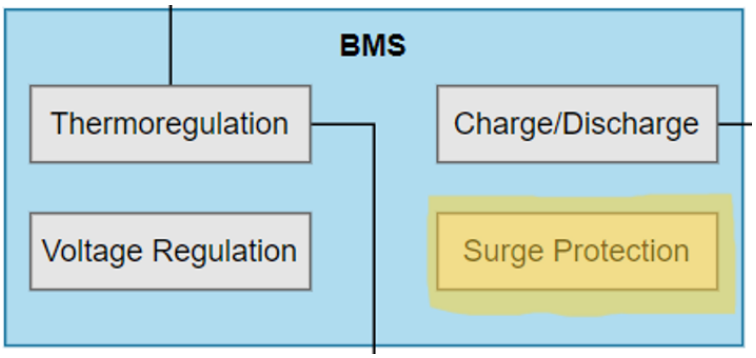


Figure 6: Surge protection subsystem description diagram

### 4.5.1 Subsystem Hardware

Hardware included to apply protection against surges is the use of diodes and the electrical isolation of circuits with drastically different loads, in this case logic level power and load level power coming from the logic side battery and power side battery. In addition there is also added bypass capacitors to reduce surge noise on the logic side. The vehicle also relies on the motor drivers to remove any significant current surges to enter the logic side of the circuit. A similar approach is made to isolate the power side and logic side battery sources, different from the motor isolation, with an optocoupler and a MOSFET. Lastly, applied fuses are placed at the main power entry point to protect the system from extreme surges.

## 4.6 Temperature Regulation

The point of the Temperature Regulation subsystem is to simply protect all modules from extreme cases of overheating. This subsystem is the guardian of thermal regulations and handles and reports situations potentially harmful for the system in regards to thermal threats.

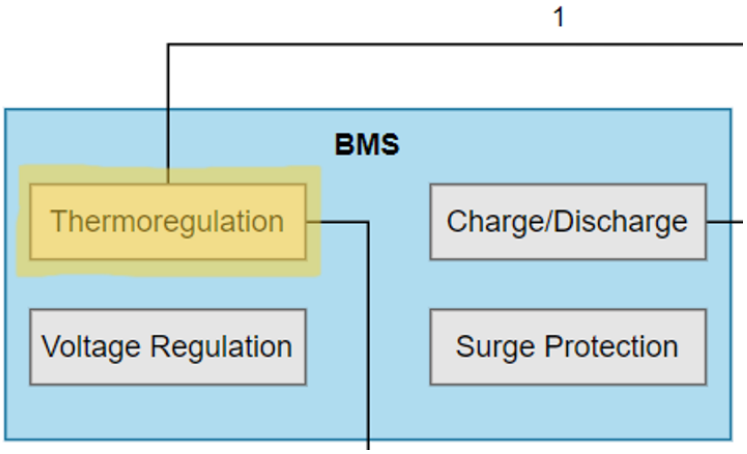


Figure 7: Temperature regulation subsystem description diagram

### 4.6.1 Subsystem Hardware

The temperature regulation subsystem utilizes a suite of ADC readings paired to thermocouples. Additionally, the TMP36 temperature sensors are also implemented to gauge compartment temperature and serve as a offset reference to our thermal couples because the thermal couple look up tables are referenced to 0 degrees Celsius and the result can be linearly shifted to gauge reliable result.

### 4.6.2 Subsystem Operating System

This module utilizes a light weight custom real-time operating system with the intention of efficient and effective task executions.

### 4.6.3 Subsystem Software Dependencies

The temperature regulation subsystem depends on analog-to-digital converter readings from thermocouples to monitor key areas including the two main batteries and the 4 main motors.

### 4.6.4 Subsystem Programming Languages

The temperature regulation subsystem is written entirely in C.

### 4.6.5 Subsystem Data Structures

The main data structure employed in the temperature regulation module is the K-type thermocouple's lookup table. The lookup table is utilized to efficiently gauge temperature readings, saving time and energy on computation as well as promptly responding to severe thermal situations. Additionally, data is structured and put together to be sent to the diagnostic system, such as motor and battery temperatures and a similar practice is in place to notify the motor controller if the motors reach substantially high temperatures.

### 4.6.6 Subsystem Data Processing

The temperature regulation subsystem processes data in a watch dog like fashion. It gauges the situation, in this case the temperature levels gauged from motors and batteries, and then alerts and responds if the system encounters trouble, such as overheating of a system.

## 4.7 Charging/Discharging

The objective of the charging/discharging subsystem is to be able to gauge the state of the battery. Total battery life is calculated and based off of tested readings, then based on the total measured current draw

per uniformed sample, energy loss is calculated. Once battery levels reach an established end point, the system is shut down. Additionally the system recognizes when the vehicle is finished charging based on voltage, temperature readings, and calculated battery percentage. Use of the vehicle while charging is not supported and risk is on the user in the case that this occurs.

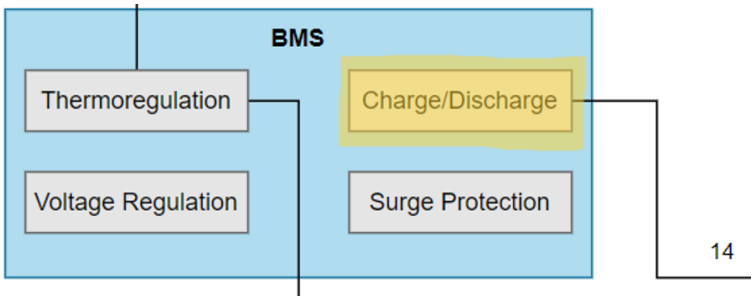


Figure 8: Charging/discharging subsystem description diagram

### 4.7.1 Subsystem Hardware

The charging/discharging subsystem utilizes ADC's, thermocouples, and TMP36's to gauge voltage and temperature of the battery in an effort to properly handle the charging and discharging of the batteries.

### 4.7.2 Subsystem Operating System

This module utilizes a light weight custom real-time operating system with the intention of efficient and effective task executions.

### 4.7.3 Subsystem Software Dependencies

A strong dependency of the charging/discharging subsystem is the reliability and timely nature of the ADC voltage readings that are used in software to calculate battery life.

### 4.7.4 Subsystem Programming Languages

The charging/discharging subsystem is written entirely in C.

### 4.7.5 Subsystem Data Structures

The charging/discharging subsystem does not hold any data structures but it does assemble and send data structures to the diagnostic system. The data it assembles and sends is changes in the calculated battery life percentage in terms of integers.

### 4.7.6 Subsystem Data Processing

ADC readings are periodically gauged to points measuring the motors' current draw. The information is then accessed and battery life is then calculated from the data processed. This information is predominantly utilized by the discharging portion of this subsystem to know when to soft shut down the system, but this measured calculated battery life is also used preemptively in an attempt to preserve battery health. However, the calculated battery life is also used by the charging portion of this subsystem to decide when to stop charging, and this is paired with temperature and voltage reading to establish a good stopping point.

# 5 Motor Control Layer Subsystems

## 5.1 Layer Hardware

The Motor Control System runs on a Tiva C Series TM4C123GXL LaunchPad Evaluation Board with a TM4C123GH6PM MCU. The MCU is an ARM Cortex-M4-based microcontroller with 256 kB Flash memory, 32 kB SRAM, 80 MHz operation, and multiple peripherals including UART, CAN, PWM, and more. In addition, the system uses an MCP2561 high-speed CAN transceiver to handle and translate the differential signals of the CAN bus to logical signals for the MCU.

## 5.2 Layer Operating System

The Motor Control System runs on a self-designed Real-Time Operating System.

## 5.3 Layer Software Dependencies

The Motor Control System uses CAN drivers based on TivaWare's C device peripherals library. Additionally, the Real Time Operating System running on the device was implemented on a framework developed by professor Losh. Finally, many libraries for basic commands were developed by professor Losh.

## 5.4 Velocity Control

The Velocity Control subsystem is a group of tasks running on the Motor Control microcontroller. The purpose of this subsystem is to manage the velocity of the car by telling the DC Motor Logic subsystem how to drive the motors. This module is responsible for telling the car to move forward, backward, turn, and stop. Additionally, it is responsible for receiving input from a remote control and telling the motors how to react. Finally, this module is responsible for tracking the speed of the car by taking input from optical interrupter sensors mounted on the wheels, whose number of interrupts per second have a direct impact on the speed of the vehicle.

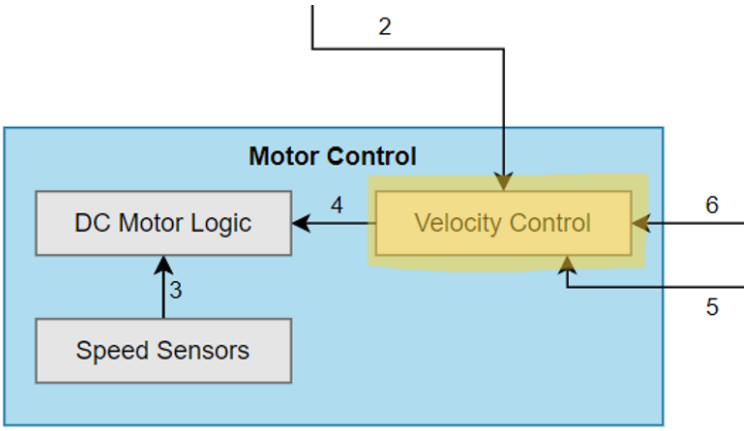


Figure 9: Velocity control subsystem description diagram

### 5.4.1 Subsystem Hardware

This subsystem does not implement any hardware, and simply uses the hardware built into the TMC123GXL board. This hardware includes timers and GPIO pins.

### 5.4.2 Subsystem Operating System

The Velocity Control subsystem is a group of tasks running on a self-designed Real-Time Operating System.

### 5.4.3 Subsystem Software Dependencies

This system will rely on the Optical Interrupters from the Speed Sensors Subsystem, as well as CAN messages sent from other systems.

### 5.4.4 Subsystem Programming Languages

The Velocity Control Subsystem is written entirely in C.

### 5.4.5 Subsystem Data Structures

The Velocity Control Subsystem does not utilize any data structures.

### 5.4.6 Subsystem Data Processing

The Velocity Control Subsystem has an algorithm that averages the number of interrupts (speed) of each wheel with a motor using a simple ring buffer filter. The system consists of multiple interrupt-driven timers which increment unique variables corresponding to the number of interrupts from each wheel. Additionally, another timer running at a 1 second interrupt rate averages all of the ring buffers and multiplies them by a predefined distance per interrupt value, which is a mechanical constant derived from the hardware dimensions. Ultimately, this will allow for an up-to-date speed reading for the Motor Control System. Another algorithm running on this system will be a listener, which listens for a message and updates the motors' speeds accordingly. This task waits on a semaphore that is posted to when the specific message is received. When the semaphore has a value, the task consumes the resource, and updates the motor speeds using the DC Motor Logic Subsystem.

## 5.5 DC Motor Logic

The DC Motor Logic Subsystem is a group of functions on the Motor Control microcontroller. The purpose of this subsystem is to provide a way for the controller to communicate with the motors and manipulate their angular velocity. The DC Motor Logic Subsystem consists of 5 DC Motors, 4 of which control the speed and 1 of which controls the direction by mechanically turning the wheels. The Subsystem will also be responsible for equalizing the speed of each motor using a PID controller.

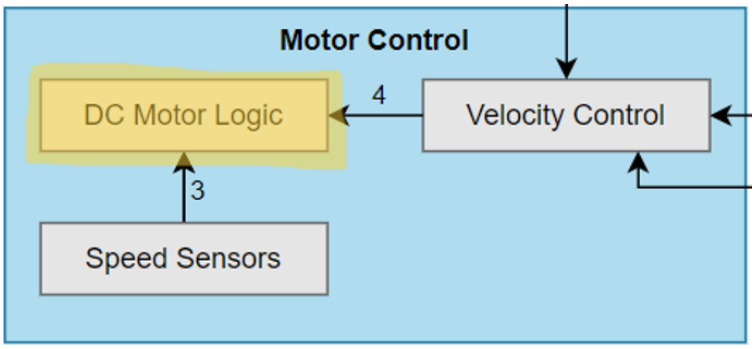


Figure 10: DC motor logic subsystem description diagram

### 5.5.1 Subsystem Hardware

This Subsystem consists of 5 DC motors that are all connected to their own BTS7960 43A H-bridge modules. The logic inputs of the H-bridge are connected to the Motor Control microcontroller to the voltage supply, GPIO pins, and PWM outputs to manipulate the motors. Finally, the subsystem contains a power distribution board that splits the 12V power supply of the battery to each motor module, supplying the high voltage to the motors.

### 5.5.2 Subsystem Operating System

The DC Motor Logic subsystem is a group of functions running on a self-designed Real-Time Operating System.

### 5.5.3 Subsystem Software Dependencies

This system will rely on the battery voltage and direction from the Velocity Control Subsystem. It will also rely on the Speed Sensors Subsystem for PID operation.

### 5.5.4 Subsystem Programming Languages

The DC Motor Logic Subsystem is written entirely in C.

### 5.5.5 Subsystem Data Structures

The Velocity Control Subsystem does not utilize any data structures.

### 5.5.6 Subsystem Data Processing

The DC Motor Logic Subsystem will utilize one main algorithm which implements a PID Controller. This algorithm will run at a 100kHz rate and will take in the current speeds of each wheel from the Speed Sensors Subsystem, processed in the Velocity Control Subsystem, and work to drive the differences in their speeds to zero. The algorithm uses coefficients that were optimized empirically. The algorithm then updates the effected PWM outputs, which then ensures the wheels are moving at the same speed.

## 5.6 Speed Sensors

The Speed Sensors Subsystem is a collection of optical encoders mounted on each wheel. The purpose of this subsystem is to provide the car with accurate speed data that will allow the Velocity Control subsystem to monitor the speed of the car as well as allow the DC Motor Logic Subsystem to equalize the speed of all the wheels.

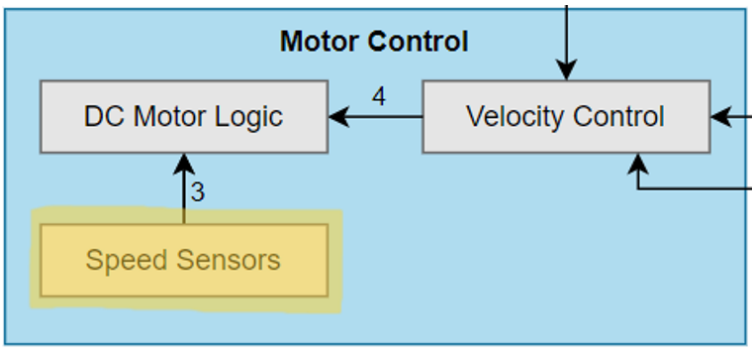


Figure 11: Speed sensors subsystem description diagram

### 5.6.1 Subsystem Hardware

The optical encoders in the Speed Sensors Subsystem are simply optical interrupters placed around a rotating disc with evenly spaced holes that is mounted to the rotor of the motors. The mount as well as the disc are 3-D printed parts and yield a specific distance/interrupt measurement. These optical interrupters are powered by the Motor Control microcontroller.

### 5.6.2 Subsystem Operating System

This subsystem does not utilize an operating system.

### 5.6.3 Subsystem Software Dependencies

This system does not contain any software, as it just consists of the sensors themselves. The other subsystems handle the data collection and interpretation.

### 5.6.4 Subsystem Programming Languages

No programming in this subsystem.

### 5.6.5 Subsystem Data Structures

The Speed Sensors Subsystem does not utilize any data structures.

### 5.6.6 Subsystem Data Processing

The Speed Sensors Subsystem does not process any data.

# 6 Security Layer Subsystems

## 6.1 Layer Hardware

For this module, all the software is being run on a ARM M4F Microcontroller which is on a TM4C123GXL Evaluation Board that contains all the peripherals needed to interface with other components. Additionally, an MCP2561 high-speed CAN transceiver is used to handle and translate the differential signals of the CAN bus to logical signals for the MCU.

## 6.2 Layer Operating System

The operating system used is the custom Real Time Operating System (RTOS) that was specifically optimized to work on this microcontroller. The framework of the RTOS, which includes the overall function prototypes and software layout, was provided by Dr. Losh, meanwhile the implementation of the RTOS was done by the CoW team.

## 6.3 Layer Software Dependencies

In order to make use of all the peripherals available on the development board, drivers were needed which were provided by Dr. Losh. However, some specific drivers were developed by the team and others individually.

The CAN driver was developed by the team to allow communication between microcontrollers using the CAN lines.

The wireless driver to allow communication of the remote control and the receiver was developed by a single team member.

## 6.4 Remote Control

The Remote Control system is two microcontrollers with the ability to communicate with each other wirelessly. One microcontroller will obtain user data using push buttons, switches, and joysticks. The user will be able to control the car because the data will be sent to the receiving microcontroller on the car itself, which will then send messages to other car modules to execute commands accordingly.

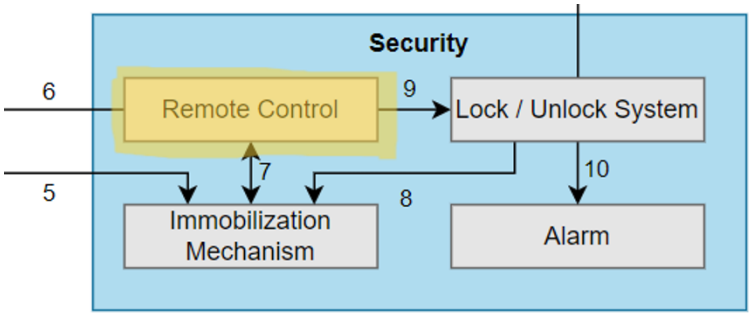


Figure 12: Remote control subsystem description diagram

### 6.4.1 Subsystem Hardware

The wireless module used is the nRF24L01+ wireless transceiver which will enable wireless communication. Also, two joysticks will be used.

### 6.4.2 Subsystem Operating System

The subsystem itself is a task in the receiving microcontroller within the custom RTOS. The primary transmitter will be running on an open loop configuration.

### 6.4.3 Subsystem Software Dependencies

The remote control will heavily rely on the ADC libraries provided by Dr. Losh and the custom wireless developed by the team member. The circuit makes use of two joysticks which are essentially two potentiometers, of which voltages can be interpreted as speed and direction.

### 6.4.4 Subsystem Programming Languages

All subsystem software is written in C with the exception of ARM Assembly for portions of the RTOS.

### 6.4.5 Subsystem Data Structures

The data transmitted will be a 5 byte data structure of which the first two bytes represent the speed, the second two bytes the direction, and the last byte is status of buttons and switches in the remote control for wireless transmissions.

For the CAN messages there will be a single CAN message that transmits the two byte speed information of the motor control system.

### 6.4.6 Subsystem Data Processing

Light signal processing will be used to ensure that the data voltages read by the ADC are not noisy and are interpreted correctly by the receiving microcontroller.

## 6.5 Lock / Unlock System

The lock and unlock system is a task in the wireless controller that will actuate relays. The relays will be located on the compartments of the car, which will allow it to secure the microcontrollers in the vehicle. If the user needs to access the compartments of the car, it will send an unlock command via the remote control, and the compartments will be unlocked for the user.

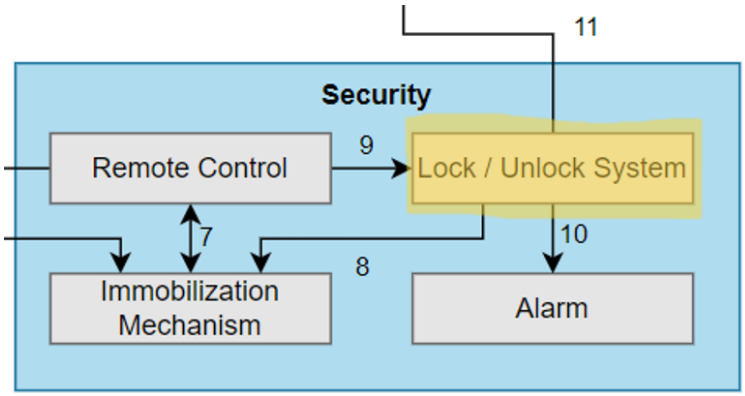


Figure 13: Lock / Unlock System subsystem description diagram

### 6.5.1 Subsystem Hardware

The lock and unlock system will make use of small relays that will lock the compartments of the car that hold the sensitive modules that are essential for the vehicle.

### 6.5.2 Subsystem Operating System

The subsystem itself is a task in the receiving microcontroller within the custom RTOS.

### 6.5.3 Subsystem Software Dependencies

The remote control will heavily rely on the GPIO libraries provided by Dr. Losh to actuate the relays to lock and unlock the compartments.

### 6.5.4 Subsystem Programming Languages

All subsystem software is written in C with the exception of ARM Assembly for portions of the RTOS.

## 6.6 Immobilization Mechanism

The immobilization mechanism is the system that prevents the car from moving unless a valid user is near the car. If the car recognizes an authorized nearby user, it will then permit the motors to run. However, if it does not detect a valid user, then the motors will not run.

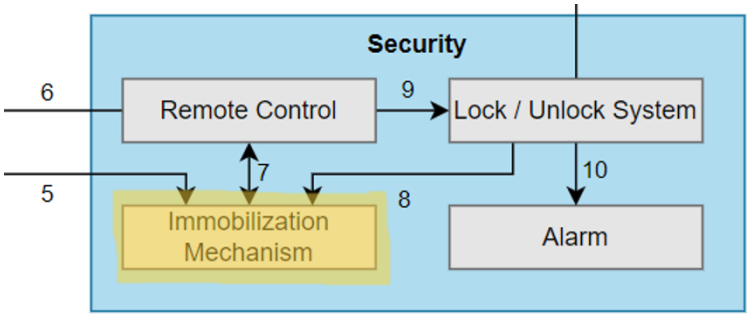


Figure 14: Immobilization mechanism subsystem description diagram

### 6.6.1 Subsystem Operating System

The subsystem itself is a task in the receiving microcontroller within the custom RTOS.

### 6.6.2 Subsystem Software Dependencies

The immobilization mechanism will rely on the CAN drivers to allow for communication with the other modules of the car.

### 6.6.3 Subsystem Programming Languages

All subsystem software is written in C with the exception of ARM Assembly for portions of the RTOS.

### 6.6.4 Subsystem Data Structures

The data transmitted on the CAN bus line will be a single byte of information that contains the immobilization status of the car. This message will then be received by the Motor Controller Layer.

## 6.7 Alarm System

The alarm system on the electric vehicle is going to be a rudimentary system that flashes LED's on and off periodically and plays a tone that sweeps up in frequency to grab the user's attention. The alarm system will be activated if there is a sudden sound that is above the ambient noise floor, which may indicate a problem.

### 6.7.1 Subsystem Hardware

The alarm system will make use of a grove speaker set with an amplifier to make the sounds. Also, the system will make use of the embedded lights in the car to actuate them to turn on and off. Lastly, the CMC-9745-44P model electret condenser microphone is going to be used to detect noises.

### 6.7.2 Subsystem Operating System

The subsystem itself is a task in the receiving microcontroller within the custom RTOS.

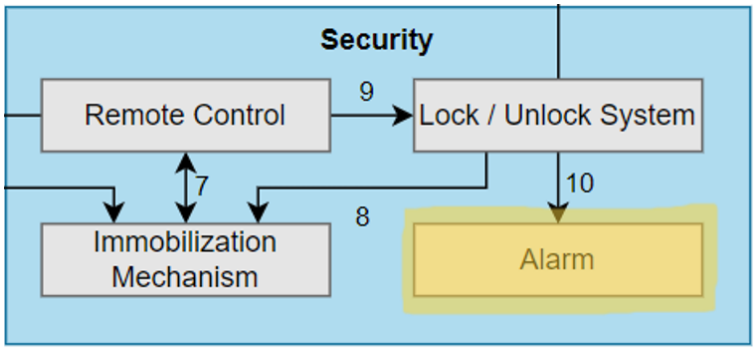


Figure 15: Alarm subsystem description diagram

### 6.7.3 Subsystem Software Dependencies

The microphone will rely on the ADC libraries provided by Dr. Losh. The actuation of lights will be accomplished by using the GPIO library. Lastly, the PWM library will be used to drive the speaker system to play the different tones.

### 6.7.4 Subsystem Programming Languages

All subsystem software is written in C with the exception of ARM Assembly for portions of the RTOS.

### 6.7.5 Subsystem Data Processing

Light signal processing will be used to ensure that the data voltages read by the ADC are not noisy and are interpreted correctly by the receiving microcontroller. This will be implemented using a circular buffer to filter the high frequency noise.

# 7 Appendix A

## 7.1 BMS Circuit Design

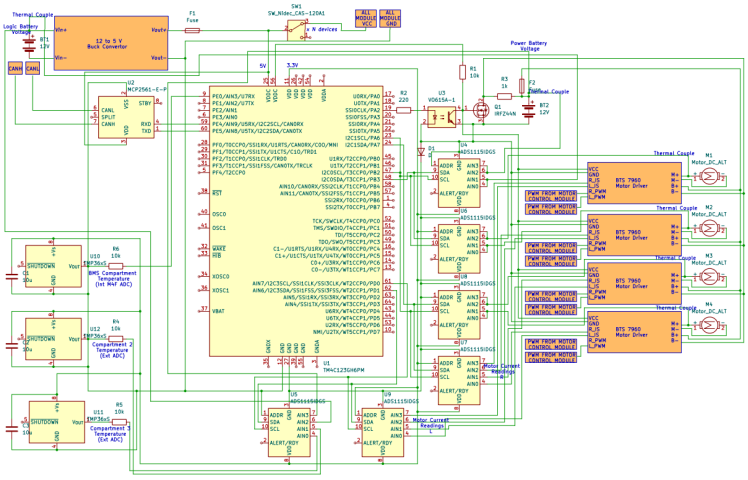

Figure 16: BMS Circuit Design

# References