12-2022

Modeling and Control of Battery Management Systems with High-Frequency AC Link Coupled Multiport Series Resonant Converters for 2nd Life Battery Applications

Brooks Jace Maughan

Utah State University

MODELING AND CONTROL OF BATTERY MANAGEMENT SYSTEMS WITH HIGH-FREQUENCY AC LINK COUPLED MULTIPORT SERIES RESONANT CONVERTERS FOR 2ND LIFE BATTERY APPLICATIONS

by

Brooks Jace Maughan

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Electrical Engineering

Approved:

Hongjie Wang, Ph.D. Major Professor	Regan Zane, Ph.D. Committee Member
Donald Cripps, Ph.D. Committee Member	D. Richard Cutler, Ph.D. Vice Provost of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2022

ABSTRACT

MODELING AND CONTROL OF BATTERY MANAGEMENT SYSTEMS WITH HIGH-FREQUENCY AC LINK COUPLED MULTIPORT SERIES RESONANT CONVERTERS FOR 2ND LIFE BATTERY APPLICATIONS

by

Brooks Jace Maughan, Master of Science
Utah State University, 2022

Major Professor: Hongjie Wang, Ph.D.
Department: Electrical and Computer Engineering

With the emerging solutions of Active Cell Balancing to enable second life use for Battery Management Systems, there is a need for large systems that can recycle large amounts of batteries. At Utah State University, a system has been designed and tested to enable up to 8 cells to be recycled using active cell balancing but has not been tested with larger amounts of battery cells. This project entails the design, production, and testing of a larger system up to 96 cells that would all be balanced to similar State of Health (SOH). This is done efficiently using calculated ratings and safety limits and permits the hot swapping, replacement, of a battery cell including disconnection preceding the replacement of a particular cell while the system is still running, of any module in the system.

(151 pages)

PUBLIC ABSTRACT

MODELING AND CONTROL OF BATTERY MANAGEMENT SYSTEMS WITH HIGH-FREQUENCY AC LINK COUPLED MULTIPORT SERIES RESONANT CONVERTERS FOR 2ND LIFE BATTERY APPLICATIONS

Brooks Jace Maughan

While the use and production of Electric Vehicles becomes more prevalent, it is also important to make this economical and ensure the reduction of a carbon footprint. Second-life batteries can satisfy both problems as batteries can be used in a second-life application for lower power purposes such as supplementing the grid so the infrastructure needed to charge the expanding fleet of Electric Vehicles can be easily supplied. This thesis goes through the process of Active Cell Balancing which will produce equal capacities, or similar batteries, that can be more efficiently used in these and other types of second-life applications. The process is expanded through series connection of the converters and modules of batteries so the process can be used for up to 96 Nissan Cells at a time. This project at Utah State University will also entail the removal and replacement of these Nissan battery cells while running to ensure quick but smooth balancing of hundreds of batteries.

To my wife and family.

ACKNOWLEDGMENTS

I would like to express my appreciation to the professors and coworkers at ASPIRE that have helped me throughout my thesis work and contributed to my learning. The experience and accumulation of knowledge throughout my time at ASPIRE has gotten me to the point where I am now and I am a better engineer because of it.

Brooks J. Maughan

CONTENTS

LIST OF FIGURES

ACRONYMS

SRC	Series Resonant Converters
BMS	Battery Management Systems
SOH	State of Health (in regard to BMS)
HF	High Frequency
SOC	State of Charge (in regard to BMS)
ACB	Active Cell Balancing
AC	Alternating Current
DC/DC	Direct Current (in reference to Power Converters)
CV	Constant Voltage
PCB	Printed Circuit Boards
EOL	End of Life
BESS	Battery Energy Storage System
SPI	Serial Peripheral Interface
MISO	Master In Slave Out
EV	Electric Vehicle
POC	Proof of Concept
OCV	Open Circuit Voltage
ZVS	Zero Voltage Switching
IC	Integrated Circuit
PI	Proportional-Integrated (in reference to feedback controllers)
ADC	Analog-to-Digital
CAN	Controller Area Network
PWM	Pulse Width Modulation
FPGA	Field Programmable Gate Array

CHAPTER 1

INTRODUCTION

1.1 Second Life Battery Applications of Lithium-Ion Batteries

Lithium Ion batteries are becoming more prevalent due to the expansion in vehicle electrification across all fronts. These batteries are extremely efficient and can maintain charge for years without degradation of the battery State of Health (SOH) or State of Charge (SOC). Electric Vehicles (EV) commonly use these types of batteries for their efficient characteristics and impressive chemistry. The degradation of a battery is an important question that has been analyzed and discussed in literature aplenty [3]. Electric Vehicles however only use the first 20% of the battery life-cycle which is then discarded for later use.

This thesis focuses on preparation of Li-Ion Batteries for Second Life Applications. This second life can best be defined by the State of Health after First-Life EV use, also known as the End-of-Life (EOL) stage, and before the battery degrades rapidly around 30% SOH, also known as the ageing knee. An example of a Lithium-Ion battery SOH curve with these critical points is represented in Figure 1.1. Second-Life Applications are typically much lower power density needs with lower C-rates (Current-rates), especially when compared to First-life. Many of these applications have to do with series connections of these Battery Systems used for grid supplement and protection, or for storage directly from the grid or an alternative power source. These series connections are best suited and last longer when the battery pack is homogeneous in State of Heath with the other batteries connected in the system. Thus the need for balanced batteries which can be best prepared using Active Cell Balancing techniques discussed in this thesis.

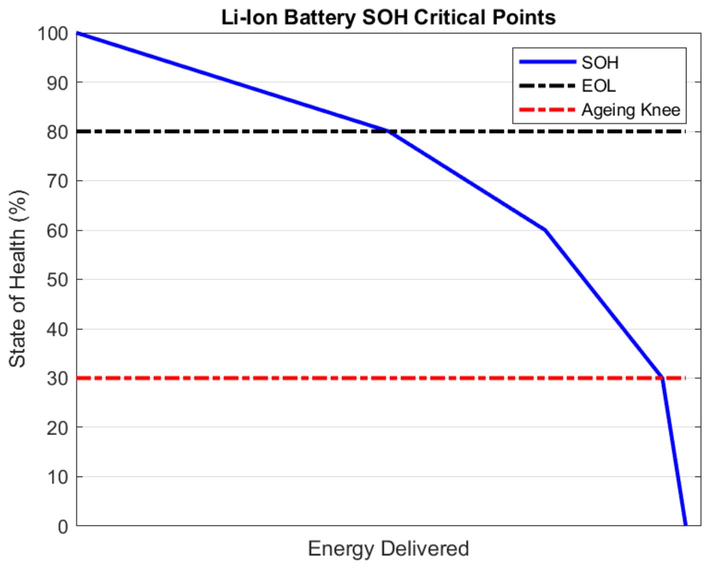


Fig. 1.1: State of Health curve over a period of time and energy provided

1.2 Active Cell Balancing with a High Frequency Link Transformer

Cell balancing techniques can be done either passively or actively. Passive Cell balancing consists of various types of algorithms which use bleeder resistors to gradually degrade the capacity of cells with higher SOH than other batteries in the system [4]. This can be a long, arduous process and the system can not be scaled to balance n amount of cells at a time. To contrast this type of balancing, there exists Active Cell Balancing which uses power converters to cycle through discharging and charging of batteries until the capacity of each cell in the power converter has become equal.

Active Cell Balancing will be demonstrated in this thesis using a Series Resonant Converter system with series output connection of various converters. This type of converter and connection will allow for a large amount of batteries to be balanced at a time, limited to the number of converters in the system multiplied by the number of battery input ports per converter. These series resonant converters are designed using a novel High Frequency Link transformer to limit the resistance of the series connection of output transformers by using one transformer as the secondary for each of the primaries. This topology and converter will be discussed in the following chapters along with the design and control of the system.

CHAPTER 2
BACKGROUND AND LITERATURE REVIEW

In modern technology, Battery Management Systems (BMS) are used to operate all types of technology. As cells degrade in State of Health (SOH) and nominal capacity, active cell balancing is required if these cells are to be used again in a fully assembled battery pack. Magnetically coupled isolated topologies to perform this active balancing are usually limited to three ports due to challenges in controlling the hardware and the plenitude of switches and components needed to couple more than three ports [5] [6].

2.1 Battery Management Systems

In a topology similar to the one proposed in this proposal, a High-Frequency (HF) transformer based isolated DC/DC converter is introduced that provides sufficient speed (1 kHz) for voltage matching. The secondary winding is common among each of the transformers, meaning that the secondary side transformer for each of the primary ports could be either connected in series, or have a magnetic field that would be scalable enough to handle the power transfer for each of the primary ports with just the one secondary side transformer [7]. This behavior is shown below in Figure 2.1. With this output regulation on three ports, the cells connected to each primary port can be continuously balanced through active or passive cell balancing, and the amount of input ports can be scaled to aggregate three or more primary input ports. Active cell balancing distributes energy among each of the cells to balance State of Charge (SOC), while passive balancing is less efficient due to dissipation of energy across balancing resistors [8].

In comparison with other three port power converters, this HF link uses a resonant LC tank to transfer the energy in a stable, more efficient manner than other topologies such as the non-resonant dual active bridge. This topology would account for many advantageous approaches, such as: 1) Where m is the number of input primary side ports the reduced

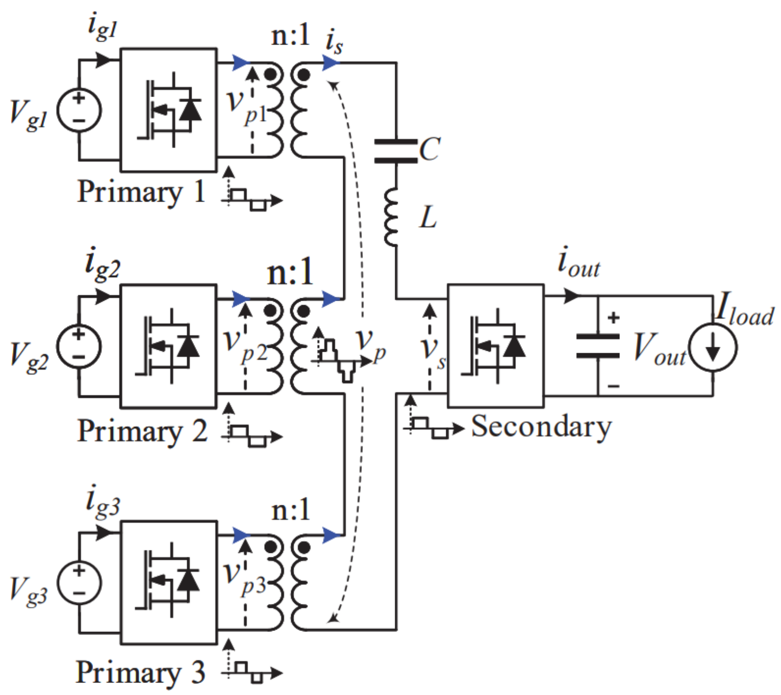


Fig. 2.1: Three-port power transfer using one series-connected secondary transformer

component count of switches and other components needed per secondary side transformer would only amount to m + 1 rather than 2*m. 2) Efficiency goes up due to the winding length being much shorter because the windings are either coupled into one winding or are wound in series, reducing length opposed to separate secondary side transformers for each input. 3) Each of the cells would have a common ground reference as well as a common bus bar which would negate the need for isolation of each bus.

2.2 Second-Life Applications of Battery Cells

Battery Management Systems typically produce high voltage systems due to the series connection between batteries that are nominally under a lower voltage. These systems typically take on the State of Health (SOH) of the battery with the lowest SOH. Because of variances in their nominal capacities and SOH, these systems degrade rapidly and need replacing of the cells with more similar cells [9]. For this reason, it has become an emphasis of research to enable these batteries for second-life applications to reduce electric vehicle costs and to make the process of using these BMS more economical [10]. As cells are recycled in preparation for their second-life, it enables them to be used in future BMS under similar SOH and nominal capacities that the cells are recycled with. To do so, it requires to balance the cells in a common system using either Passive or Active Cell Balancing described hereafter.

2.3 Active Cell Balancing of Battery Management Systems

Throughout literature there have been various methods of control for the active cell balancing, as well as the varying load types that have been applied to the systems. The output can be loaded with a fixed output, fixed voltage, or fixed resistance where each comes with their own limitations. Fixed current requires input voltage to be regulated, which is a much more tedious, and long process as opposed to regulating input current in a system [11]. Regulation of input current requires a fixed voltage system, which is a common approach used in much of literature but a downside being the need to share output voltage. This is known as droop current and is further researched and explained in this proposal so it may be implemented in the project. Previous authors have also regulated output voltages to achieve SOC balancing [12] which would require constant input current at each of the ports. Overall, the limitations seem to favor using droop current and more research on the subject would affirm that fact.

To generate a full duty cycle (50 percent) a four-switch H-Bridge is commonly applied to the input ports to generate square waves through gate drivers that will result in AC input voltage. This AC input voltage will be applied to the SRC where it will go through another four-switch H-Bridge on the secondary side to produce DC voltage, thus affirming the DC analysis of the phasor transformation. Referring to the Active Cell Balancing methods, the duty of each of the ports will vary depending on the SOC and SOH of each cell [7]. By changing the duty cycle between ports, it allows for the input current of each of the cells to differ while only controlling one "base" current, defined as the current of the best cell. The "best cell" is found using a voltage map [13], as shown below in Figure 2.2 which takes both SOH and SOC into account to map out which cell has the highest (best) order of magnitude according to the gain provided by the voltage map labelled as K_{map}. This base current is a central part to controlling the SRC converter for this project but is also referred to as the “reference current” because it is the controlled current value that each input primary port will refer to. The results from these author’s tests show the control of the input current effects the output current, but does not directly control the output current unless attached to a constant voltage (CV) load.

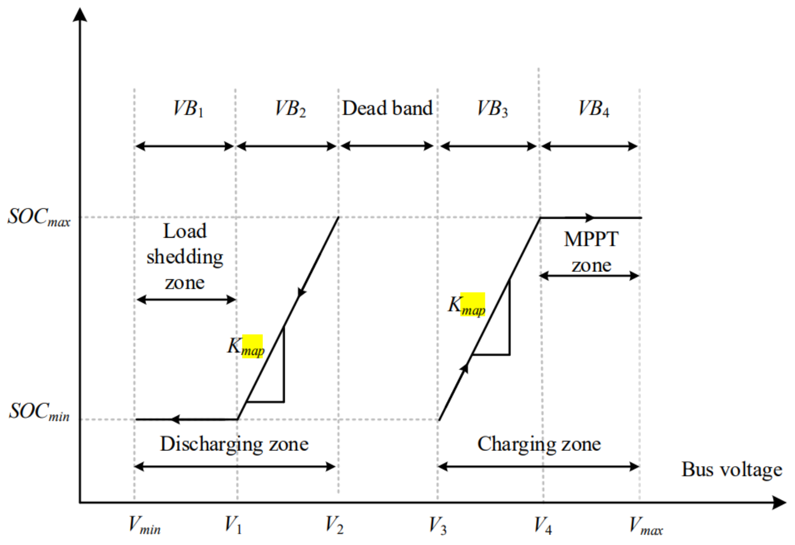


Fig. 2.2: Voltage mapping for active cell balancing

To alter the common reference current, the phase between the primary and secondary windings is altered to account for the change in current across each primary winding. This current regulation is important because it allows the outputs of the converters to be connected in series or parallel [13]. With the input cells providing a somewhat constant voltage, this reference current can be used to calculate the input power, using the equation shown below, which is then transferred over the converter to the output. By measuring the output current and considering the CV load on the output, the efficiency can be analyzed by calculating the secondary power using the same equation.

$$P = VI \tag{2.1}$$

2.4 Phasor Transformation of Series Resonant Converters

To regulate the input current, knowledge of the phasor currents and its input variables are required. Phasor transformations have been done for all types of converters including Buck, Boost, Buck-Boost, as well as SRC. Authors have analyzed the behaviors of SRC converters and have found it is difficult to analyze Alternating Current (AC) signals, but Direct Current (DC) signals can be simplified to give an approximate signal [14]. The math of this approximation can be difficult, but by using DC analysis and unity gains (1:1 conversion ratio of input to output) between input to output, it is proposed and discovered that input current can be controlled by the phase shift between input and output as well as the duty cycle of the bridges on each primary port. The equivalent circuit of an SRC used for this DC analysis is shown below in Figure 2.3.

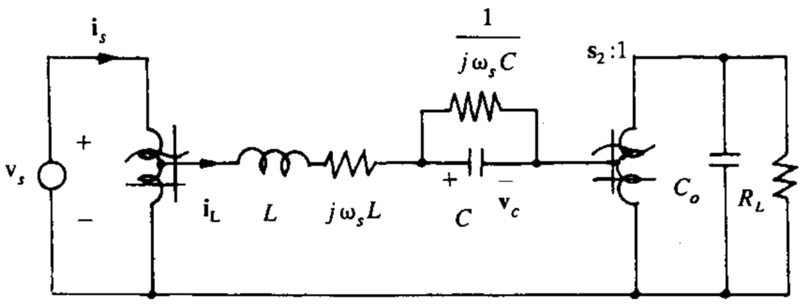


Fig. 2.3: Series Resonant Converter equivalent circuit input-output

2.5 Voltage Sharing of Output Series Connected Converters

Power sharing and regulation has been implemented through many of these power converters. Due to the limitations of just three primary ports in previous models, the series connection of output was the only method of balancing and controlling more than three battery cells at a time. This power sharing has come by way of both series and parallel connection, but parallel capabilities are degraded when implementing the input current regulation as researched. The major problem of output series connections of these type of converters is when connected in series the output voltage is extremely hard to control due to the resonant capacitance on the output terminals [13].

In [15]. the author noted that when a couple of SRC outputs were connected in series while in current regulation mode, the output voltage became mismatched and at times applied only to a single converter in an uncontrollable manner. Voltage sharing is a necessity among output series connected SRC and this author developed a Multi-Mode control that would implement a digital slope R_i which would determine the maximum voltage a converter could take according to the current of the output using equation 2.2. The author of [13] approached the problem in a similar manner and implemented a "droop" method that would achieve natural voltage sharing in the same manner. The performance of this droop slope is to be calculated, but an equivalent plot is shown in Figure 2.4. as well as the implementation of droop within the full system is shown in Figure 2.5. This droop accounts for any differences and maintain a similar voltage across each of the outputs (slightly offset due to uncontrollable discrepancies such as voltage loss, measuring errors, etc.).

$$V = IR \tag{2.2}$$

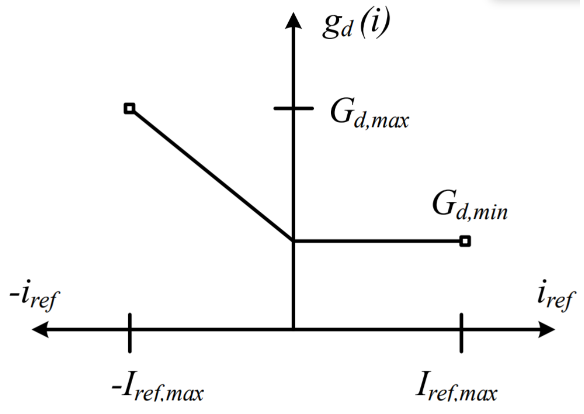


Fig. 2.4: Droop slope needed for voltage sharing

This droop control allows for voltage sharing across N modules as shown above. This figure also shows the signals (namely I_{all}) being sent from one controller to each of the converters. It is vital that the communication between each of these controllers be syn-

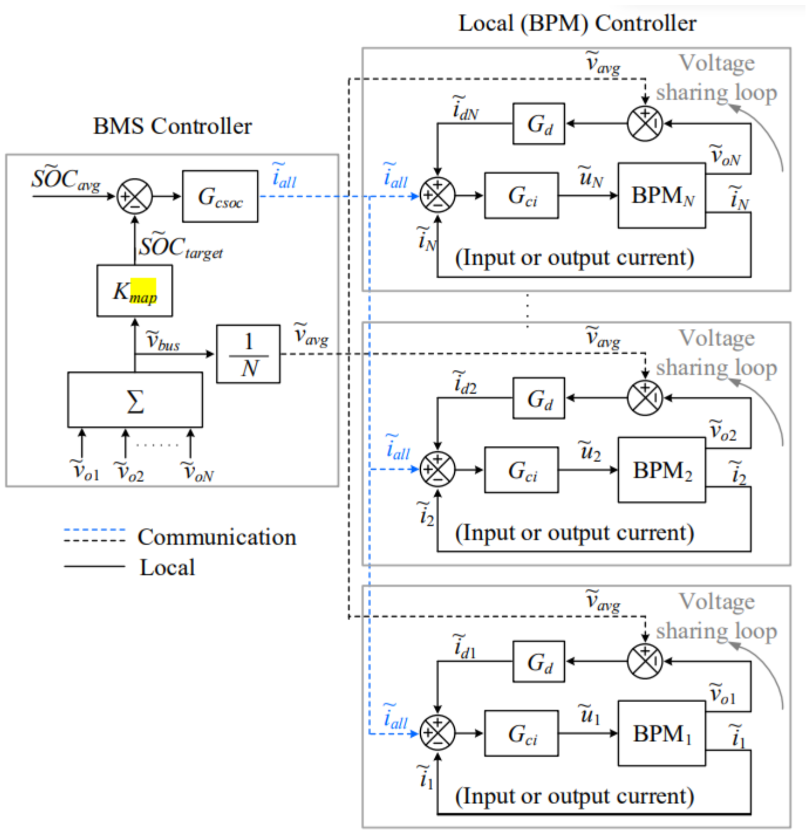


Fig. 2.5: N-port system for current regulation using voltage sharing by droop control

chronous. For this reason, the signals needed to start conversion and control parameters such as reference current and duty cycle are sent through a string controller to ensure timely communication [9]. Research has been done using String Controllers using various types of communication including SPI, I2C, and CAN. In [9]. the String Controller used CAN communication which made the system scalable to N modules and ensured that each module could control reference current at the same frequency.

2.6 Hot Swapping

When using series or parallel connected converter, it is viable to disconnect one of the N batteries while the system is running and connect a new battery in the place of that primary port. This is called hot swapping [16]. Hot swapping has mostly been done for parallel-connected Battery packs, but the idea remains the same. There are various problems with hot swapping so it must be done with care, otherwise the battery may be damaged due to the peak current exceeding the nominal current of the system when disconnecting or replacing the battery [13]. Thus, the system must be able to ramp down the input current of one module while still processing the active cell balancing of the other N-1 systems. This method is an ongoing research discussion.

2.7 Proposed Approach

With the emerging solutions of Active Cell Balancing to enable second life use for BMS, there is a need for large systems that can recycle large amounts of batteries. At Utah State University, a system has been designed and tested to enable up to 8 cells to be recycled using active cell balancing but has not been tested with larger amounts of battery cells. This project would entail the design, production, and testing of a larger system up to 96 cells that would all be balanced to similar SOH. This is to be done efficiently using calculated ratings and safety limits and will allow for hot swapping, replacement of a battery cell including disconnection preceding the replacement of a particular cell while the system is still running, of any module in the system.

Design and testing of this battery system used for series control of a SRC will include but not be limited to these procedures. The SRC will need both inductors and a system control board to be designed and the efficiency will be analyzed and idealized. This board will need to be tested and controlled for active cell balancing of Nissan Cells to enable second life applications for battery cells. The secondary side of each of the boards will be connected in series to output up to 750 V using said Nissan Cells. For this series connection to function accurately, a PI droop compensator will need to be designed so each converter can share voltage nearly equivalent across each of the output ports. Modules containing these converters will be connected in series on the secondary side and communication between each of these modules will need to be synchronous at a frequency of nearly 1 kHz. This system will also require being able to disconnect and reconnect battery cells to fulfill the term "hot swapping."

CHAPTER 3

BATTERY CELL RECYCLING FOR SECOND LIFE APPLICATION

3.1 Second Life Applications

Lithium-Ion batteries are commonly used in the EV sector, but prices of these batteries keep rising due to the limit that these batteries hit within the first 20% of their lifetime as their capacities degrade. This capacity limit of 70-80% is known as the 'ageing knee' and is typically when a battery is discarded from EV operation and deemed at its End of Life (EOL) for Electric Vehicle purposes [17]. As prices keep rising in the Lithium-Ion industry due to demand in the EV sector, optimization of these batteries is required for longer lifetimes of these batteries. The term 'second-life' refers to the batteries that have reached their EOL stage and are no longer applicable for Electric Vehicle use and are typically discarded, creating a carbon footprint that will be discussed later in this report. Instead of discarding these batteries, they can be used in applications that are less demanding when it comes to performance, volume, and weight limitations.

These applications for Lithium-Ion batteries can enable longer life of these batteries which will have environmental benefits as well as economical benefits. Studies have shown that Second-Life applications could drop the price of Lithium-Ion batteries by nearly 50% over the next 5 years as the production of batteries gets larger to accommodate for more Electric Vehicles and the use of these batteries are prolonged [18]. Bloomberg New Energy Finance estimates that the market for second-life batteries could account for 26 GWh by 2025 and will continue growing as the EV sector continues to grow. As the EV sector continues to grow, the grid will need to provide up to 30% more power in peak power times and these second-life batteries could be used to supplement the grid and be used in a power smoothing application for grid-scale power plants as shown in Figure 3.1.

So what happens to batteries during their first life that causes the End of Life stage

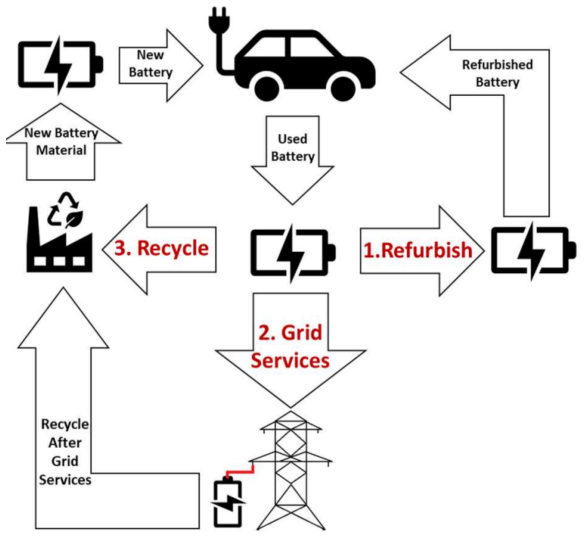


Fig. 3.1: Lithium-Ion battery recycled life-cycles for power smoothing of power plants

that limits these batteries from being able to provide power to Electric Vehicles? As shown in Figure 3.2 below, a Lithium Battery has three significant instances in its lifetime from 0-100%. The First-life EOL threshold is due to a change in slope of the capacity that usually occurs at around the 70% mark due to higher internal resistance and lower nominal capacitance. This State of Health (SOH) is no longer capable of powering a high-voltage application such as an Electric Vehicle, so Li-Ion batteries are removed from first-life application at the 80% mark, or close to it, to avoid that initial drop in slope of SOH. The next drop in slope occurs at about that 60% mark, but varies according to cell chemistry and capacity degradation that the cell has seen earlier in its lifetime, including during first-life. The final instance is the 30% threshold that signifies the Second-life EOL threshold. As seen in the below figure, the slope drastically drops to a point where the cell is no longer usable, even for second-life applications.

Second-Life applications are required to be less-demanding than applications such as Electric Vehicles. Performance, volume, and weight must not be critical to the function of the system, and the system must operate with low energy density to avoid high charge/discharge of the battery with high currents. The Figure 3.3 below shows a study done in [18] that indicates a fade in SOH at higher temperatures, which is fine for first-life, but to op-

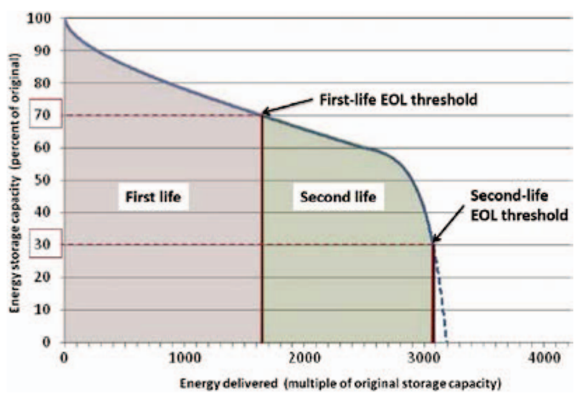


Fig. 3.2: Lithium-Ion battery SOH curve

timize the capacity and SOH curve for second-life applications, this must be used at lower temperatures, as indicated by the small slope in capacity fade from the red data.

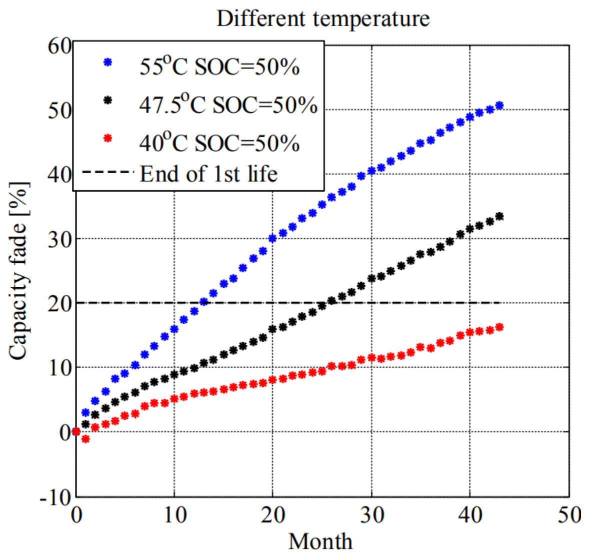


Fig. 3.3: SOH and capacity fade depending on temperature

Another reason for second-life applications requiring lower energy density can be seen in Figure 3.4 from [19] which indicates the resistance and capacity fades becoming more rapid in latter stages of life, and that slope should continue to degrade as Li-Ion cells are used in second-life applications. Higher internal resistance promotes more resistive losses in the connection between the internal battery in the battery pack, and the application to which it is connected. Another study in [20] models a Li-ion battery with the circuit shown in Figure 3.5. This circuit is used to demonstrate Open Circuit Voltage curve development and tests were run to find the OCV curve of a Li-Ion cell at varying rates of SOH. The data shown in Figure 3.6 demonstrates the OCV curves developed through charge (the top waveforms) and discharge (the bottom waveforms) cycles that indicate lower capacity and degraded OCV curves as a battery is degraded in SOH. The study also found confirmed that lower C-rates of these EOL batteries resulted in higher efficiency of each Battery Management System as a whole with slower degradation rates.

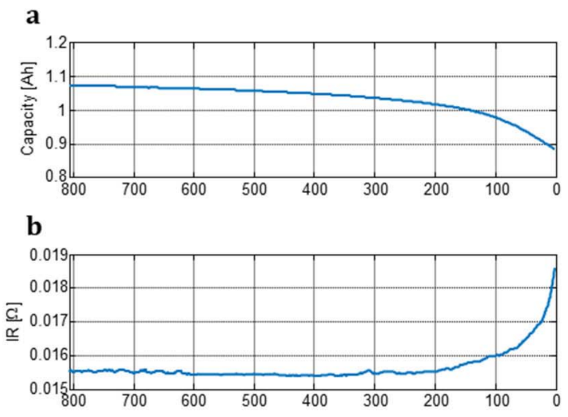


Fig. 3.4: Resistive and capacitive fade of Li-Ion batteries throughout second life

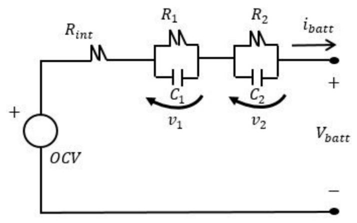


Fig. 3.5: Equivalent circuit of a Li-Ion battery

To optimize second-life performance for lower density applications, a study was done in [21] to detail the Useful life of a Li-Ion battery in its second-life. These Remaining-Useful-Life (RUL) Tests were typically used on batteries that were monitored their whole life, which is not the case with battery packs in electric vehicles, but the technique remained

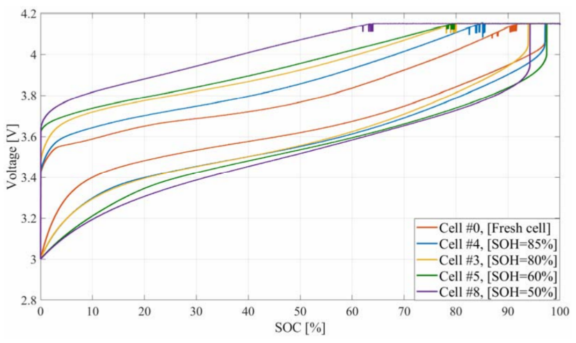


Fig. 3.6: Open Circuit Voltage curve at varying SOH

the same to for the estimation algorithm of RUL for second-life batteries. The estimation requires large amounts of tests which take away from the RUL, so a healthy medium is established in this paper and the success rate is shown in Figure 3.7. Figure 3.8 shows the results of the RUL estimation techniques used on second-life Li-Ion batteries after only 150 cycles were performed for the algorithm.

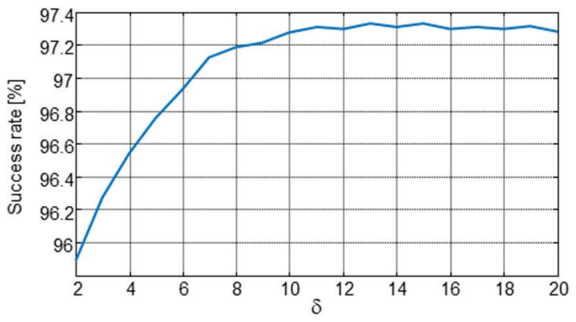


Fig. 3.7: RUL Estimation success rate

Optimization of Li-Ion batteries in second-life can also result from how the Battery Management Systems makeup of batteries is decided. A study in [22] represents the internal resistance and capacity of cells in a pack and the degradation through a number of cycles that they call FEC (Full Equivalent Cycles: $(Ah_{CHARGE} + Ah_{DISCHARGE})/Q_{NOM}$. As shown in Figure 3.9, the degradation of both internal resistance and of capacitance, or SOH, is much more rapid in Figure 3.10 than Figure 3.10. This is due to the homogeneity of the cells in the pack. The cells that are heterogeneous differ in SOH and capacity and

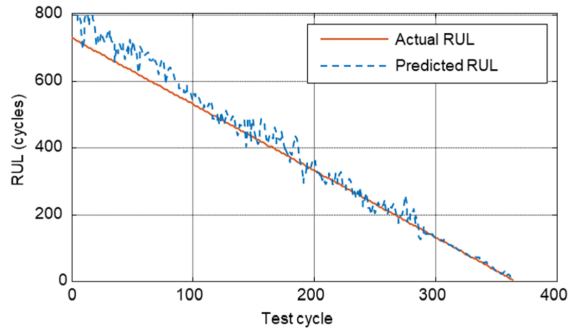


Fig. 3.8: RUL Estimation compare to actual RUL

result in much faster degradation than those cells that have been balanced. After first-life use in Electric Vehicle application, cells have degraded in both resistance and capacitance in different manners. Even battery cells in the same battery pack during first-life become heterogeneous in cell chemistry. For homogeneity in cells to be used in a second-life BMS, cell balancing is required which can balance a number of cells through either active or passive cell balancing [23]. The makeup of the homogeneous cells result in a much better chemistry in the battery pack which would be well utilized in second-life applications.

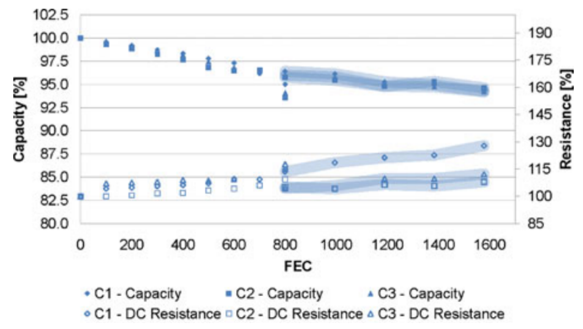


Fig. 3.9: Degradation of a homogeneous collection of Li-Ion cells

As stated before, second-life battery applications require low energy density and relatively low C-rates compared to their use in first life. These applications include but are not limited to: Battery storage Systems that can increase household energy self-consumption and can power a small building or a home with the energy stored; Low-voltage grid reinforcements that can provide small amounts of voltage to account for any instability provided

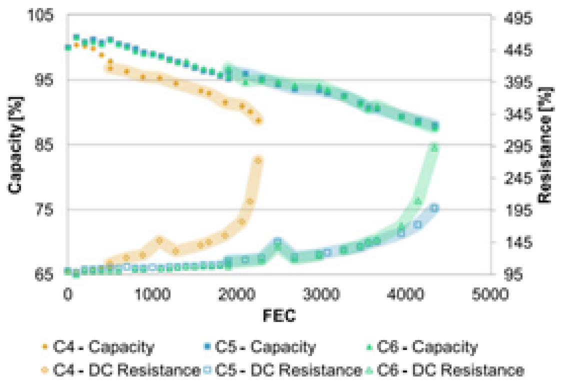


Fig. 3.10: Degradation of a heterogeneous collection of Li-Ion cells

from applications being connected to the grid spontaneously; Grid reinforcements that can improve energy efficiency in the grid and can also match peak power generation consumption through storage systems. An application with an emphasis on being explored for second-life applications are grid reinforcements so EV batteries can be used to supplement the power grid increasing through these EV applications. The study in [24] explains the 30% additional power rating that the grid will require over the next two centuries, and they propose and test infrastructure of Battery Energy Storage Supplements for the grid. This process is shown below in Figure 3.11 and was validated through testing this process. These grid supplements would enable a better market and economy of Li-Ion batteries with prolonged use and would be able to stabilize grid problems that could arise from full electrification of transportation.

Research continues to be furthered in the area of Second Life Battery Technology. Second-Life batteries in general are becoming a bigger topic due to the nature of Lithium-Ion batteries and the need to find a more sustainable way of using them both environmentally and economically. With the expansion of the electrified transportation market, there grows a bigger need to determine how these batteries will be used once they have been deemed at their End-of-Life (EOL). Efforts are being made to maximize the life of the Second-Life batteries by balancing cells that will be accumulated into a battery pack to prolong the life of their next use.

Balancing of the cells can be taxing on the battery due to the charge/discharge of the

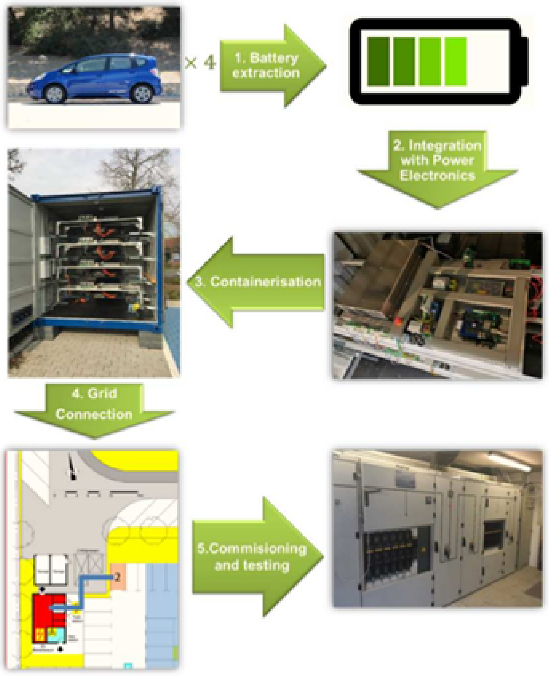

× 4
1. Battery extraction
2. Integration with Power Electronics
3. Containerisation
4. Grid Connection
5.Commisioning and testing

Fig. 3.11: Lifecycle of EV battery to grid supplemental battery

battery taking away from future cycles they could have with their second-life application. Active Cell Balancing, however, provides a longer lifetime for the batteries in their battery pack due to homogeneity of the capacities and State of Health of the batteries that would otherwise be heterogeneous. This homogeneity in cells will prolong each of these cells' lifetime considerably and thus is an ongoing area of interest to maximize lifetime of Li-Ion batteries. Active Cell Balancing entails the balancing of cell capacities using SOC Estimation algorithms and balancing techniques depending on which is the best cell and worst cell, as shown below in the Figure 3.12. Passive Cell Balancing is also a topic of interest for the same reason but has not shown quite the same progression in research as Active Cell Balancing.

There are various other techniques being used to maximize cell lifetime and efficiency by monitoring the cells during their first-life, or determining their Remaining Useful Life (RUL) using similar algorithms but without the information of the first-life [21]. Applications for battery packs with non-taxing needs continue to be researched and discovered giving more meaning for second-life application and cell balancing for their new applications.

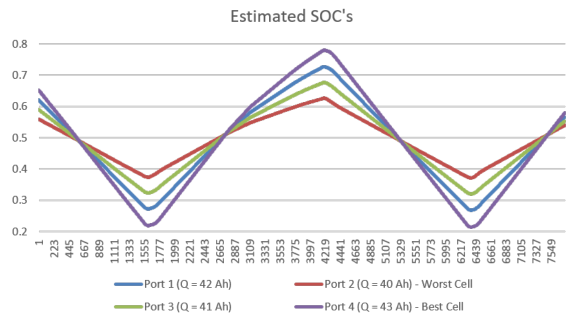
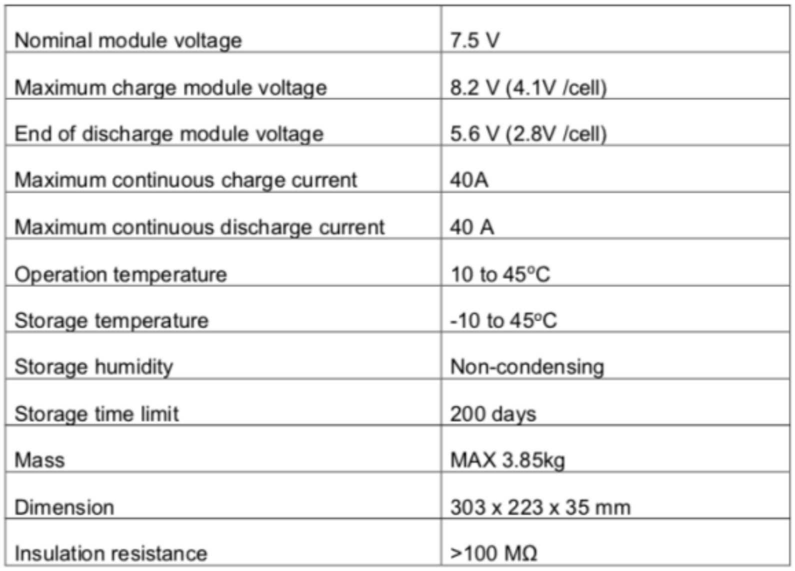


Fig. 3.12: Active Cell Balancing charge and discharge cycles depending on best/worst cells of BMS

Secondary batteries in EVs are also being considered to help cell life. By having a secondary battery to extend range the cells in the primary battery can be operated at healthier states of charge (SOCs). These techniques are especially effective in long range applications despite the increased complexity [25]. Dual battery systems could be an effective use to improve battery usage.

3.2 Arbin Cyler

Batteries, such as the Nissan Cells being used for active cell balancing and second life applications, have various characteristics such as capacities (nominal and actual), State of Charge (SOC), and State of Health (SOH) which can denote the overall performance of a specified battery as well as determine what uses and limitations it may have for certain applications. For First-life applications, such as the Electric Vehicle (EV), the cells are carefully assembled with specified characteristics and chemistries that are designated beforehand and some values are provided by the manufacturer's of the battery. Below in Figure 3.13, some specifications are shown detailing the nominal voltage and current range of the battery. These are the specifications for the Nissan Cells used in this project when they were initially created and put into production for EV use. The modern limits and nominal values will be described later in this section, as well as characterized capacities.



Nominal module voltage	7.5 V
Maximum charge module voltage	8.2 V (4.1V /cell)
End of discharge module voltage	5.6 V (2.8V /cell)
Maximum continuous charge current	40A
Maximum continuous discharge current	40 A
Operation temperature	10 to 45°C
Storage temperature	-10 to 45°C
Storage humidity	Non-condensing
Storage time limit	200 days
Mass	MAX 3.85kg
Dimension	303 x 223 x 35 mm
Insulation resistance	>100 MΩ

Fig. 3.13: Nissan Cell Characteristics from Datasheet

The capacity and SOC of a battery can be determined using Coulomb counting to develop an Open Circuit Voltage (OCV) curve. This process uses current to charge/discharge a battery at various temperatures to produce an average waveform while measuring

Capacity, in Amp-hours (Ah), on the x-axis, and the voltage, in Volts (V), on the y-axis. OCV curves can be derived by charging and discharging the battery between 0% and 100%.. The nominal capacity is derived from the amount of Ah it takes to drop the voltage from it's highest voltage, also known as the full State of Charge, to it's drop-off point to 0 V, which in this case begins at about 2.5 V. Using this OCV curve, the State of Charge can be determined by replacing the x-axis from capacity to SOC with 100% as the peak voltage and 0% SOC as the drop-off point. It is important to know the State of Charge of a battery so as to not overcharge or over-discharge a battery beyond it's limits. Typically a battery is not used past the 80% or 20% SOC due to a rapid decline/incline of the slope at these points.

The State of Health of a battery is also important, especially when contemplating second-life applications and the limitations a battery may have. Upon conception for first-life use, a battery is deemed to be at 100% State of Health and degrades the more it is used. The cutoff between first-life and second-life is typically around 80% and the Nissan Cells are taken away from their Electric Vehicles and discarded. This project maintains the ability to use these cells in second-life applications depending on the density and SOH curve of the cell. Although not explored in this thesis, research has shown that rapidly degraded first-life cells also degrade more rapidly during their second life, which can be predicted using these SOH curves during first-life (Technical Liability, Martinez-Maserna). These second-life applications are deemed successful for these recycled batteries until about the 30% SOH mark where the battery is discarded due to the rapid decline in SOH seen in aforementioned battery life usefulness tests, also known as the ageing knee. The same paper mentioned the performance enhancements of second-life cells used with homogeneous cells in regards to their SOH and internal resistance.

The Arbin Cycler is a useful tool that can determine these characteristics of batteries using coulomb counting and other processes. The Arbin was used heavily in this project as a means of a power source and to find the capacities needed for coulomb counting for the individual Nissan cells. Due to the Nissan cells' first-life degradation, the cells being used needed to be re-characterized to determine the proper OCV curve and find the actual capacity of the cell as opposed to the nominal capacity shown in the first OCV curve. The capacities of the cells were a necessity for this project to enable the coulomb counting and best/worst cell selection for the active cell balancing which will be discussed hereafter.

The Arbin is shown below in Figure 3.14 and is a battery life-cycle emulator that has various capabilities, all pertaining to batteries. It can be used to power a battery and charge it up, or as a load to discharge a battery and send power to the grid. The software developed uses switches to safely connect the 12 Arbin channels to 12 battery cells and has limits that can be set to ensure safe use of the batteries being characterized. It is capable of characterizing most battery cells, but does have limitations as it cannot exceed ±5 V and has a limit of ±300 A. Its high current capability helps to be able to run these characterization tests such as life-cycle tests, Open Circuit Voltage tests, dynamic and static characterization tests to develop an equivalent model of the battery, and many others. Tests that were not run on this project but could have been useful include the life-cycle test which charges and discharges a battery from 100% SOC to 0% SOC, or any other predetermined level of charge, to find it's first-life ageing knee where the SOH drops below 80% SOH, as well as full testing of equivalent circuit models for the Nissan Cells. The static and dynamic characterizations of the cells were performed in prelude to the equivalent circuit model analysis but were never realized fully. The results of these tests result in proper parameters for the full order cell model shown in Figure 3.15 as described in [26].

Open Circuit Voltage tests were performed on the Nissan Cells to determine new OCV curves and find the actual capacities of each cell in a pack. A detailed Open Circuit Voltage test requires a thermal chamber and Arbin to develop curves at different temperatures until an average waveform is found. The equipment needed for this varying temperature test was not available for this time of characterization. A characterization of a Lithium-Ion cells OCV curve is shown below in Figures 3.16 and 3.17 to demonstrate the small margin of difference between the test performed at room temperature, to the actual OCV curve of the cell, and of the cell characterizations at different capacities. For this reason, the Nissan

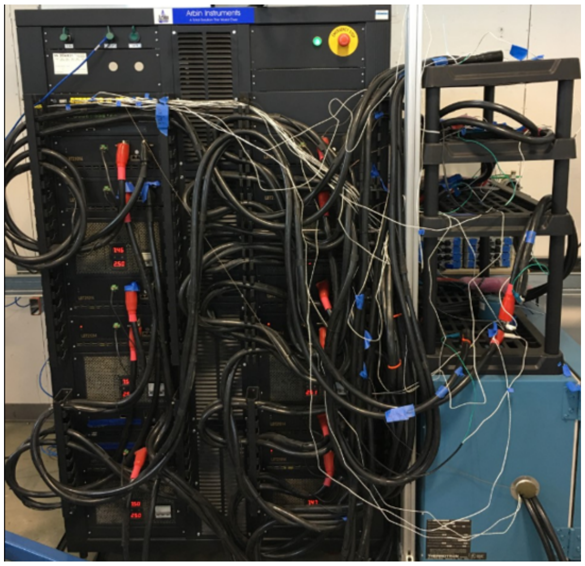


Fig. 3.14: Arbin Cycler Instrument for Battery Cycling

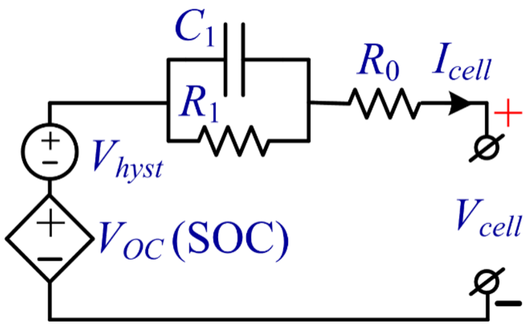


Fig. 3.15: Lithium-Ion Full Order Cell Model

Cells were tested purely at room temperature to determine the OCV curve and capacity of these second-life cells. This OCV Curve of a Nissan Cell is portrayed in Figure 3.18.

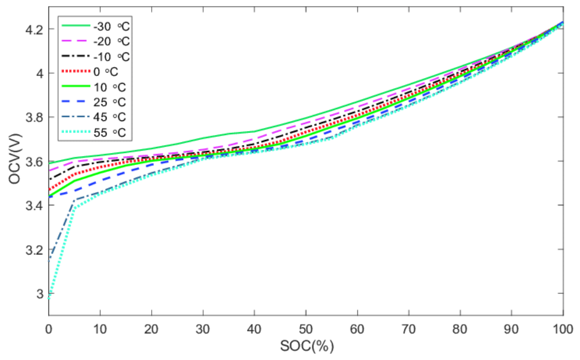


Fig. 3.16: OCV Curve developed through various temperatures [1]

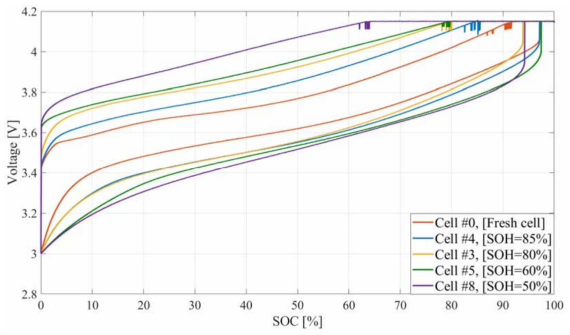


Fig. 3.17: OCV Curve developed through various capacities [2]

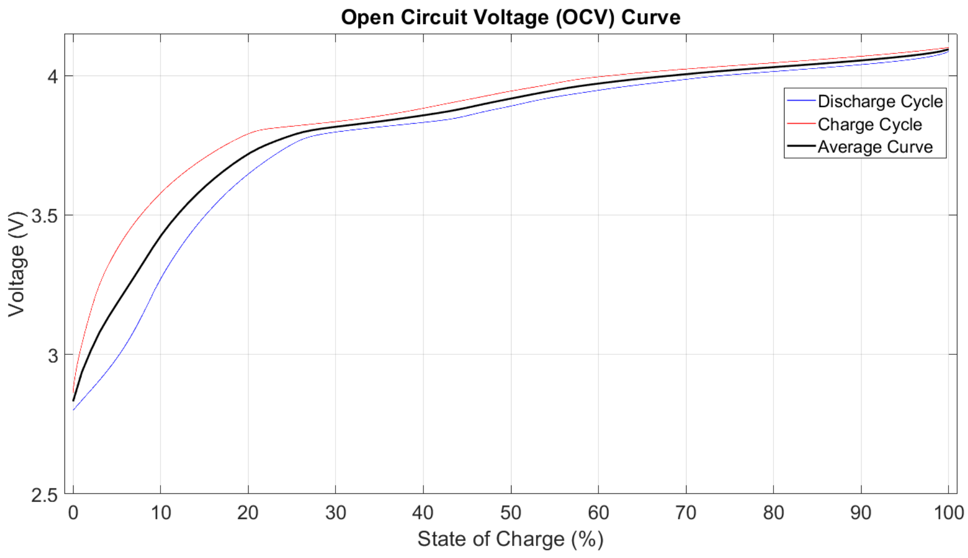


Fig. 3.18: Open Circuit Voltage Curve of a 2nd-life Nissan Cell developed by Arbin Cycler

Before running testing of the converters and circuits with the Nissan Cells, preliminary testing was done to validate the design of the circuitry to ensure connection to the Nissan Cells could be done without damage and that the software worked correctly. For this reason, the project required 4 voltage sources with high current capability, namely ±40 A, which could be provided from the Arbin Cycler. However, when the Arbin made connection to a system it would begin in discharge mode to verify the connection to a battery. This could prove harmful to the system and fuses, so circuitry was developed as shown below in Figure 3.19 to discharge two resistors in series before closing switches that would connect the converter boards to the Arbin Cycler. This ensures that the Arbin Cycler would work as a voltage source to the system without causing any of that initial harm. The attachments and components assembled for this circuitry is shown in Figure 3.20.

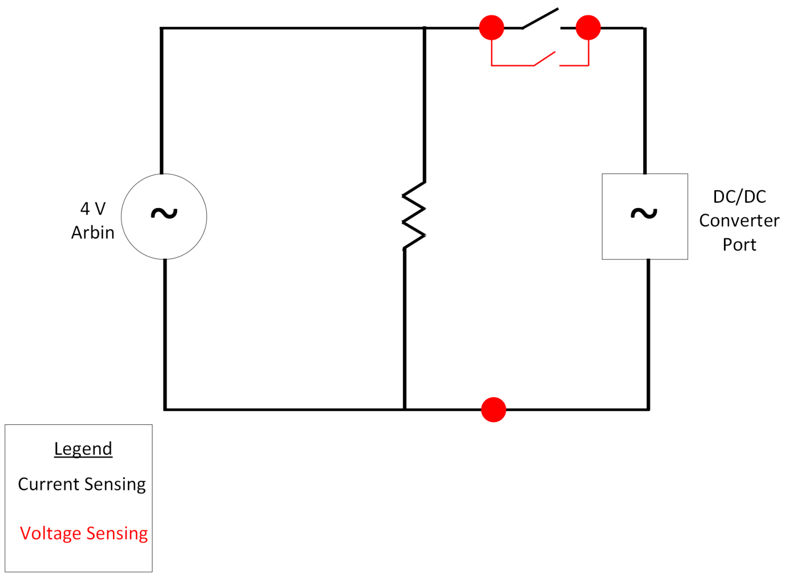


Fig. 3.19: Circuit design for isolation from Arbin to system at startup

3.3 Active Cell Balancing

Series connection of battery cells require similar chemistry and capacity of cells to reduce the imbalance between cells and possibility for instability of the full system. Even without instability, series connected battery systems with heterogeneous cells pose threats to the longevity of the battery pack and the possibility of extending the life of the battery pack during second life would be diminished. Chemistry of battery cells are inevitably different thus the batteries will degrade differently and have different capacities and State of Health. Homogeneity is a condition of the batteries that will have to be imposed on each

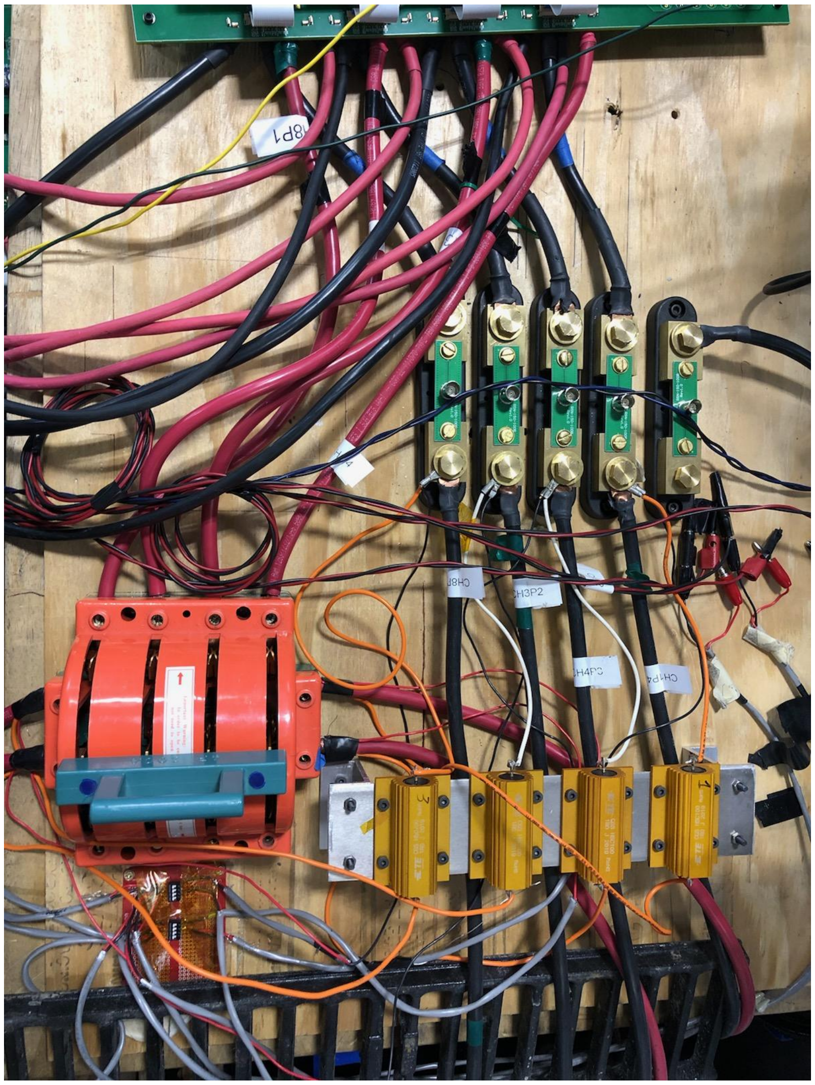


Fig. 3.20: Circuit setup for isolation from Arbin to system at startup

cell through cell balancing.

There are two types of cell balancing, one being passive cell balancing and the other active cell balancing. Passive cell balancing are slower in process and less efficient than active cell balancing. It consists of various algorithms that have been studied [27] but the concept remains the same of using bleeding resistors for batteries with higher capacities in order to discharge the battery more rapidly and to degrade the capacity until it can become similar to other batteries. This process tends to be complex and slow unlike the active cell balancing techniques that require DC/DC converters to cycle the battery cells through various charge and discharge cycles [7] until the batteries achieve homogeneity in SOH.

Active cell balancing consists of DC/DC converters that can cycle various cells through charge/discharge cycles depending on the State of Charge of the cells connected to the converter. This thesis work consists of the Series Resonant Converter connected to a High Frequency (HF) link which is a novel idea introduced in [7] and requires galvanic isolation which is provided by the HF transformer that is common for each of the primaries. Most of the DC/DC converters used for active cell balancing consist of 3 or less ports, while this thesis work will enable cell reconditioning of 4 channels per converter with various converters connected in series for a modular connection which can be connected in series with other similar modules.

Active Cell reconditioning will achieve homogeneity of the cells by discharging/charging the cells with the highest capacity (the best cell) at the highest C-rate while the cell with the lowest capacity (the worst cell) will be charged at the lowest C-rate, as shown in Figure 3.21 [28]. This will enable little to no degradation of the SOH of the worst cell while the best cell will have rapid and major degradation in SOH and nominal capacity. This method was used in [29] and validation of the algorithms are shown below in Figure 3.22 while the next section will provide some insight into the algorithm used for the cell balancing. The figure provides the necessary waveform for SOC to be charged and discharged each cell proportional to its capacity. This was done using the Proof-of-Concept (POC) prototype to validate its use for the full scale system.

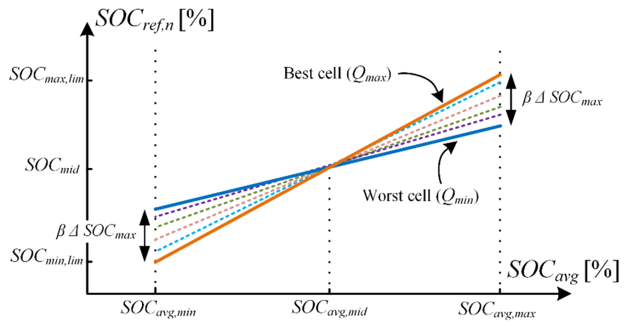


Fig. 3.21: Lithium-Ion Battery Reconditioning Objective Map

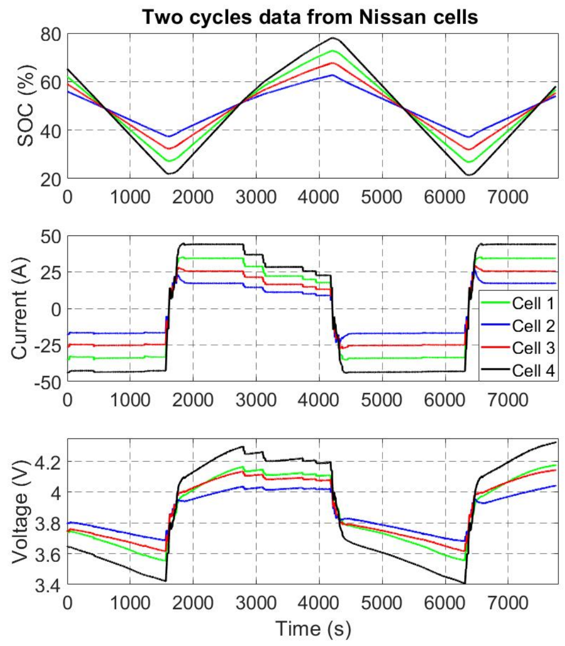


Fig. 3.22: Active Cell balancing charge and discharge waveforms using POC concept

3.4 SOC Estimation and Objective Map Calculations

SOC Estimation requires initially the data showing the actual capacity of each of the cells and the OCV curve to denote State of Charge from the voltage at the terminals, as described previously on OCV curves. While the OCV curve is helpful for SOC calculations when there is negligible current coursing through the battery cells, when current is applied the voltage needs to settle before using this again. This does not bode well for measuring SOC during cell balancing, so an algorithm was developed that can estimate SOC based on the C-rate and nominal capacity of the batteries, called objective mapping. This algorithm is known as the Sigma-point Kalman filter (SPKF) derived in [30] and the gains were calibrated to the Nissan Cells being connected to the system. This calibration and validation can be seen in Figure 3.23 where the SOC is estimated in MATLAB over various intervals. This initially has a margin of error that gets corrected over time and through more cycles, which serves well for the cell balancing for SOC estimation.

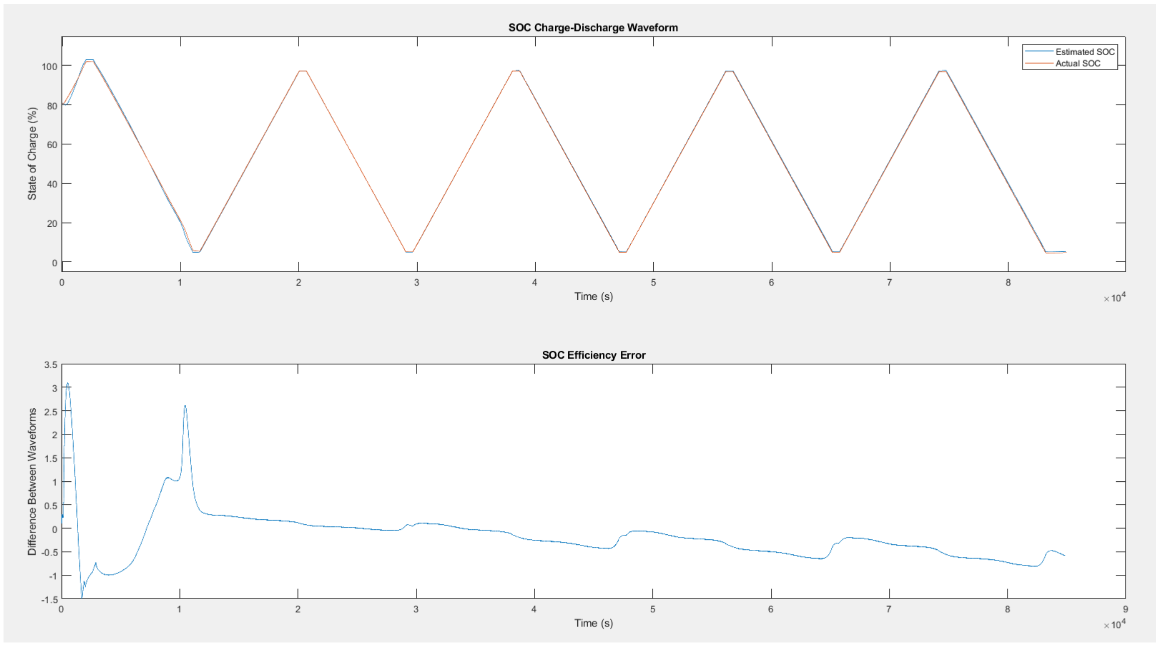


Fig. 3.23: Objective Map calibration for Nissan Cell chemistry

CHAPTER 4
MATHEMATICAL ANALYSIS AND DESIGN

4.1 Series Resonant Converters

Active Cell balancing requires a DC/DC Converter to cycle through charge and discharge cycles of batteries in a coordinated manner as described in the previous chapter. This is to be done continuously and efficiently which was an important factor in the decision to use a LCC Series Resonant Converter (SRC) for maximum power transfer and minimal resistive and conductive losses in the system which could cause the temperature of the system to exceed safety bounds needed for the cycling and use of battery cells, especially Nissan Cells.

A SRC is highly efficient due to the resonant frequency between reactive and resistive components in the system. Upon resonance, the reactive components of both primary and secondary impedances cancel each other out. This results in minimal electrical energy losses resulting in heat because the switching, resistive, and inductive losses are negligible during the resonant frequency and the system can perform more efficiently.

The switching frequency of the system was chosen to be 190 kHz, meaning that the switches would alternate in the H-Bridge every 5 us. The resonant frequency then could either be chosen to be above or below switching frequency depending on the desired switching characteristics of the system. Any resonant frequency above 200 kHz would result in Zero Current Switching and would result in the tank current increasing as the output current of the system after the rectifier decreased. With the system operating at currents up to 40 A, but also ramping current from 40 A to 0 A to -40 A, this was not desirable.

Instead, the resonant frequency was chosen to be below switching frequency, anywhere around 160 kHz for each converter unit. This resulted in Zero Voltage Switching (ZVS) and would result in the tank current decreasing as the output current of the system of the rectifier decreased. This ZVS would also result in more efficient performance of the switches and higher efficiency of the system. Resonant Frequency can also be measured as $f_r = 1/(2 * pi * sqrt(L * C))$ where C is the DC blocking capacitor in the secondary and L is the secondary transformer inductance. The Power Converter design will be discussed in the upcoming sections, but have a DC blocking capacitance value of 1uF which would require an inductance of nearly 1uH for the resonant frequency to be met. The following Figures 4.1- 4.2 show the calibrated values for the secondary of one of the converter units. These values were obtained using a frequency sweep of the LCR meter to view the resonant frequency plot and the inductance value at the switching frequency of the system.

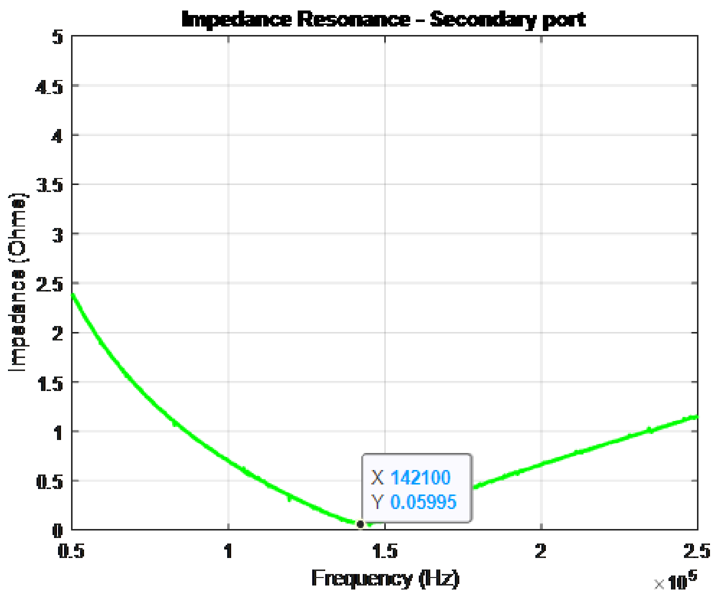
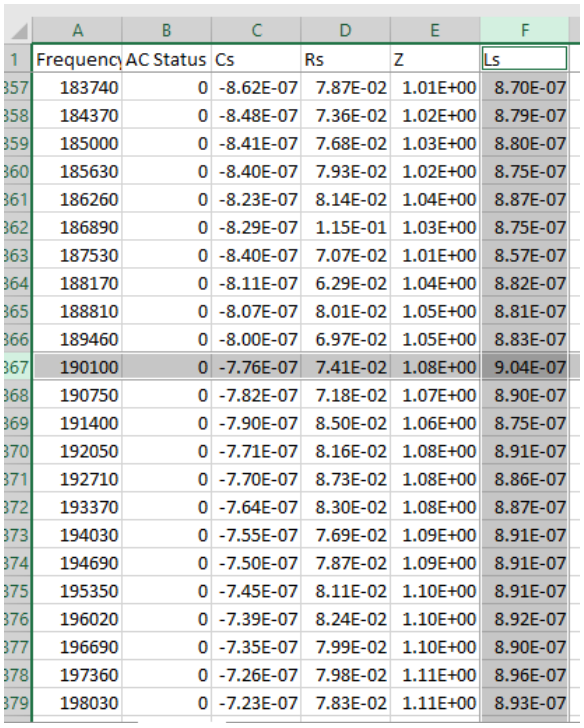


Fig. 4.1: Frequency plot of the impedance frequency sweep portraying the resonant frequency peak of the system

4.2 Control Board Design

Printed Circuit Board's were designed for the needed controller boards and power boards. The boards involved in this project include the following: Power Board, Control Board, Module Controller Board, and the String Controller Board. While the Power Board, Module Board, and String Board were previously designed, the Control Board was needed



	A	B	C	D	E	F
1	Frequenc	AC Status	Cs	Rs	Z	Ls
357	183740	0	-8.62E-07	7.87E-02	1.01E+00	8.70E-07
358	184370	0	-8.48E-07	7.36E-02	1.02E+00	8.79E-07
359	185000	0	-8.41E-07	7.68E-02	1.03E+00	8.80E-07
360	185630	0	-8.40E-07	7.93E-02	1.02E+00	8.75E-07
361	186260	0	-8.23E-07	8.14E-02	1.04E+00	8.87E-07
362	186890	0	-8.29E-07	1.15E-01	1.03E+00	8.75E-07
363	187530	0	-8.40E-07	7.07E-02	1.01E+00	8.57E-07
364	188170	0	-8.11E-07	6.29E-02	1.04E+00	8.82E-07
365	188810	0	-8.07E-07	8.01E-02	1.05E+00	8.81E-07
366	189460	0	-8.00E-07	6.97E-02	1.05E+00	8.83E-07
367	190100	0	-7.76E-07	7.41E-02	1.08E+00	9.04E-07
368	190750	0	-7.82E-07	7.18E-02	1.07E+00	8.90E-07
369	191400	0	-7.90E-07	8.50E-02	1.06E+00	8.75E-07
370	192050	0	-7.71E-07	8.16E-02	1.08E+00	8.91E-07
371	192710	0	-7.70E-07	8.73E-02	1.08E+00	8.86E-07
372	193370	0	-7.64E-07	8.30E-02	1.08E+00	8.87E-07
373	194030	0	-7.55E-07	7.69E-02	1.09E+00	8.91E-07
374	194690	0	-7.50E-07	7.87E-02	1.09E+00	8.91E-07
375	195350	0	-7.45E-07	8.11E-02	1.10E+00	8.91E-07
376	196020	0	-7.39E-07	8.24E-02	1.10E+00	8.92E-07
377	196690	0	-7.35E-07	7.99E-02	1.10E+00	8.90E-07
378	197360	0	-7.26E-07	7.98E-02	1.11E+00	8.96E-07
379	198030	0	-7.23E-07	7.83E-02	1.11E+00	8.93E-07

Fig. 4.2: LCR meter readings of a frequency sweep indicating the inductance of the secondary

for communication between the Module Controller's microcontroller and the Power Converter Board. The Controller Board would be overlaid on top of the Power Converter Board to allow gate drive signals and ADC readings to be transferred between boards through Molex connector's. Other connections necessary included the auxiliary power supply, for the 12 V power needed for powering some of the Integrated Circuits (IC's) of the Power Board and Control Board, and the RJ-45 ethernet connector's allowing communication between the Microcontroller on the Module Controller Board to the FPGA on the Control Board. Shown below is an image of the final version of the Control Board in Figure 4.3.

A difficulty in this project was component selection due to the shortage of components created from the COVID-19 pandemic. The DC/DC Converter's were replaced between the conceptual version for testing of functionality, and the actual version for full system testing as well as the ADC's and digital isolator's. The only real problem caused by these adjustments were the timing of the digital isolator's as the signal was delayed by one clock

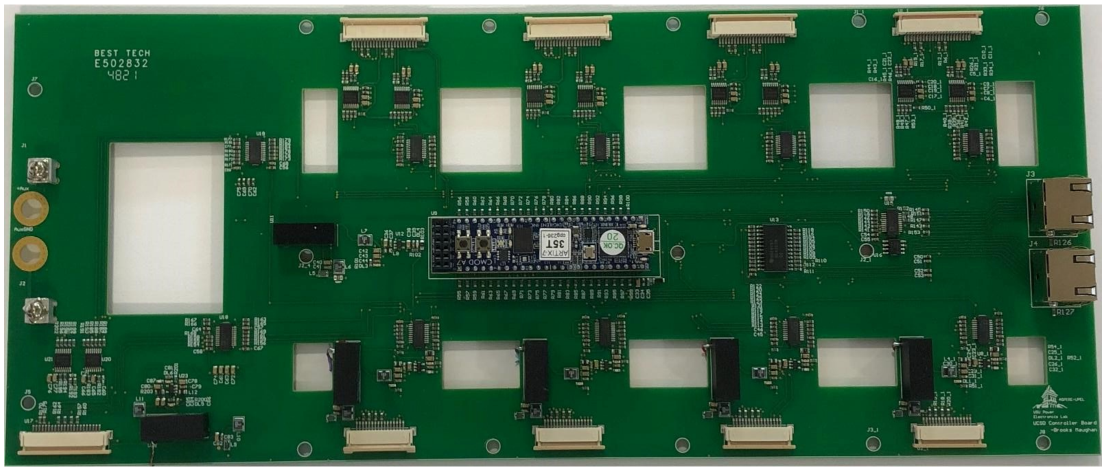


Fig. 4.3: Final design of control board for FPGA communication

signal, as seen in the oscilloscope waveform from Figure 4.4 , which was easily corrected in the FPGA code. The ADC's had no problem with the new implementation, but the signal's being sent from the current sensor had a problem. Both Current Sensor's had the same pin-out, shown below in Figure 4.5. Due to a mistake in product delivery and storage, the manufacturer provided a current sensor with a different pin-out so a small workaround was made in the soldering of the component to correctly connect the pins to the corresponding pads. The workaround on the new pin-out is shown in Figure 4.6. Other than the shorting of pins with NC pins, the IC had to be shifted slightly upward to account for the GND pin being misplaced slightly.

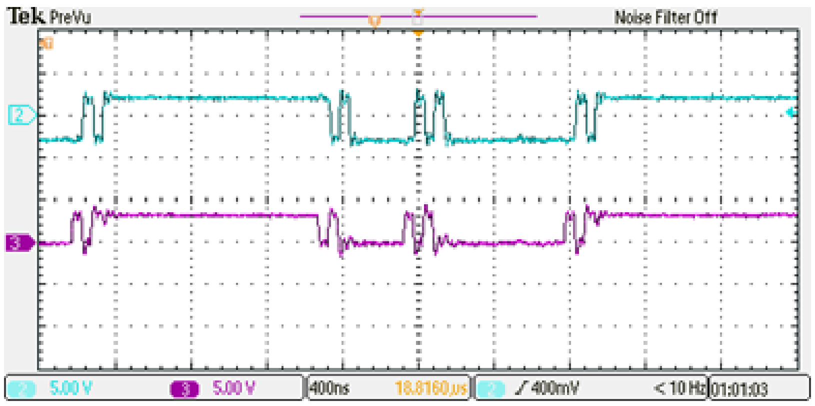


Fig. 4.4: ADC signals being delayed by one clock cycle with the new ADCs

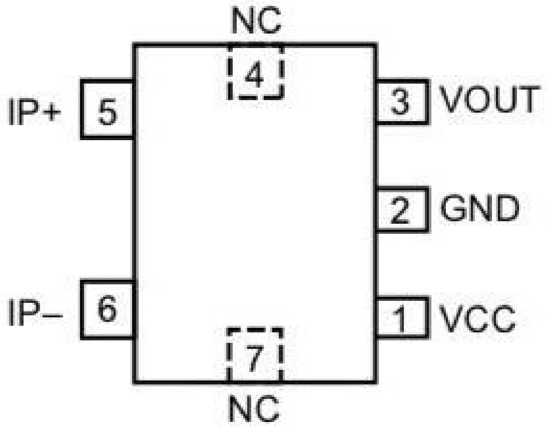


Fig. 4.5: Pin-out of the previous and new current sensor's provided by manufacturer

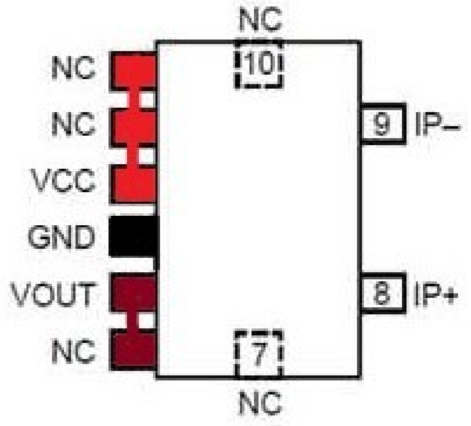


Fig. 4.6: Pin-out and workaround of the new current sensors sent by mistake

The board was designed using Altium Designer where a circuit schematic and a PCB layout was configured. Grounds were configured to allow for the 600 V operation on the secondary. Other ground and signal interference was negligible due to the use of digital isolation between each ground signal. The FPGA was powered by the 12 V auxiliary after processing the voltage through a 12 V to 5 V DC/DC converter and a 3.3 V linear regulator. A multiplexer was also needed for cycling through the thermistor's needed for the Module Control Board which also held isolation on the Control Board along with an op-amp for cycling through using Chip Select. This communication was done through Serial Peripheral Interface (SPI) with a MISO (Master In Slave Out) line for communication from the Module Controller's Microcontroller to the FPGA and SIMO for the communication from the FPGA to the Microcontroller. Digital isolation was also provided between the ground of the Microcontroller and the ground of the FPGA. More of this communication will be discussed in the Module testing of the Results section.

4.3 Boards Layout

Each Power Board has a Control Board with an FPGA to communicate signals and process data between controllers and Power Converter. The combination of these two boards interlaced together will be known hereafter as a Converter Unit. The layout is shown below in Figure 4.7 where there will be four converter units for each Module Controller Board and there will be six Module Controller Boards for every String Controller Board. This will enable the main communication to come from a String Controller to simultaneously send signals to 6 Module Boards, each of which will be communicating with 4 Converter Units. The communication needs to be synchronous so each converter unit can power up at the same time as well as regulate input current and output voltage simultaneously with the other boards.

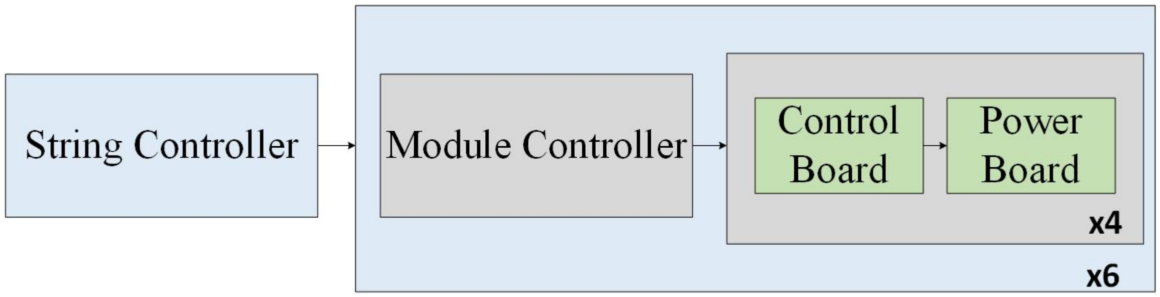


Fig. 4.7: Simple Block Diagram portraying the Boards layout

Timing between each board is set so every 1 ms the signals will be communicated between boards to operate as a 1 kHz frequency. The signals between each controller are clearly spelled out in Figure 4.8 and the method of communication each board will be using to send data from one board to the next. The main methods of communication are over a CAN bus between the String Controller and Module Controller (also the Site Controller, or the PC in this case) as well as through SPI communication using SIMO and MISO between Module Controller and Control Boards. Any analog signals, namely between control board and power board, will be traced together using Molex connectors that will allow the signal to go from one board to the next without any delay. The signals being communicated through these two boards include the gate signals, thermistor values, and the ADC readings of the input and output current and voltage. These ADC readings allow for limits and regulation of the Power Board from the FPGA. The Gate Signals are sent from the Control Board to control the switches on the Power Board which will enable power transfer from the Primary to the Secondary side of the Series Resonant Converter transformer.

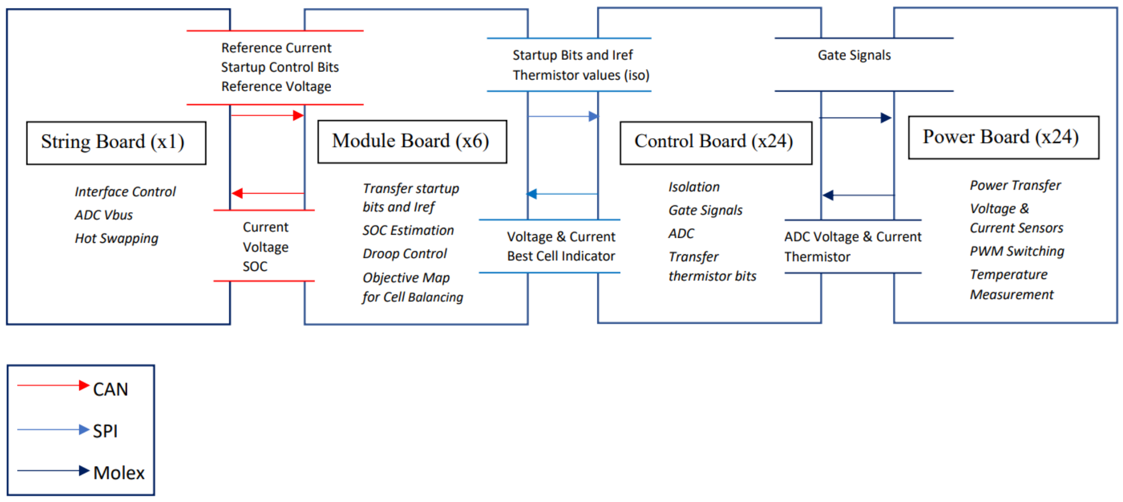


Fig. 4.8: Signals being communicated between each of the four boards. (Signals from the String Controller are coming from a Site Controller controlled by the user)

The main duties of each board are also spelled out in Figure 4.8 where the purposes of the Power Board are to allow Power Transfer from Primary to Secondary, Pulse Width Modulation (PWM) according to the signals given by the Gate Drivers, and to measure the Voltage, Current and temperature of the primaries and secondary. The Control Board is meant to drive the switches of the Power Board using gate signals, as well as isolate the grounds of signals coming between FPGA, Microcontroller, and Power Board to prevent any high voltage processing. The Control Board will also convert the analog signals to digital signals read by the FPGA which is needed for current regulation power transfer regulation, even so that it may open the switches if values are beyond a predetermined, set value for limits of the voltages and currents to ensure no power is being transferred thus protecting the system from any excessive power or transients.

Each Control Board will also send and receive signals to and from the Module Board using Serial Peripheral Interface (SPI) through Ethernet cables connected by RJ-45 connectors. The signals are also shown in the above figure, such as the voltage and current digital values read from the Analog-to-Digital-Converter (ADC) and the Best Cell Indicator so the

FPGA can regulate the current according to current regulation loops that will be discussed in the following chapter. These signals from the FPGA are read to the Microcontroller where Active Cell Balancing will occur using an objective map, as well as SOC Estimation and Droop Control, also known as Voltage Sharing, all of which will also be detailed in the next chapter of this thesis. The thermistor values are also read from FPGA and sent to FPGA for isolation. Other signals being sent are the startup bits and reference current which are both being received from the user interface to determine how the system will operate during balancing at a prescribed limit set by the user.

The String Controller Board here will be controlled by a Site Controller, namely a PC using MATLAB to send signals to the Microcontroller using SPI, but for the sake of simplicity is overlaid into one block in the Figure 4.8. The purpose of the String Board is to send signals to start up the system and control the software of every board in the system. Commands are sent from this board to begin operation, close the loop, begin differential control for active cell balancing, and many others that will be explained in the string controller testing phase. The droop control required for voltage sharing is also initialized in this controller to allow voltage sharing between modules and converter units, thus the need to receive the output voltage signal and read the output voltage through a voltage sensor and an ADC. A reference voltage will then be sent to the other boards for voltage sharing regulation. Figure 4.9 shows the setup for the preceding proof-of-conception (POC) model to validate function of the converter and communication between each board. While the converters are being powered by Arbin, they are being communicated to by a Site Controller, or the PC, which sends the startup commands and reference current to the String Controller using CAN communication. These CAN busses are also communicating to both module controller boards (only having one converter unit per module) where the Module Controller has a Microcontroller communicating with the FPGA of the Converter Units.

The setup of the overall system is also shown below in Figure 4.10 with the voltages pertaining to each system. The Nissan Cell Leaf will have a variable voltage depending on the State of Charge of the cell but the outputs will be regulated to nearly 25 V on each Converter Unit/Power Board, 100 V on each Module, for a total of 600 V on the total system through the String Controller Board. This will allow for 600 V regulation for the final system setup.

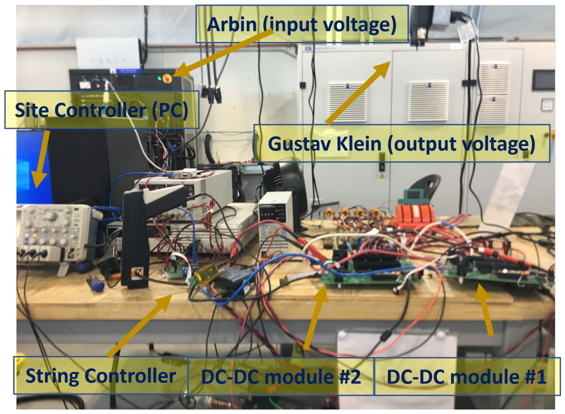


Fig. 4.9: POC setup for verification of string to module to converter unit communication

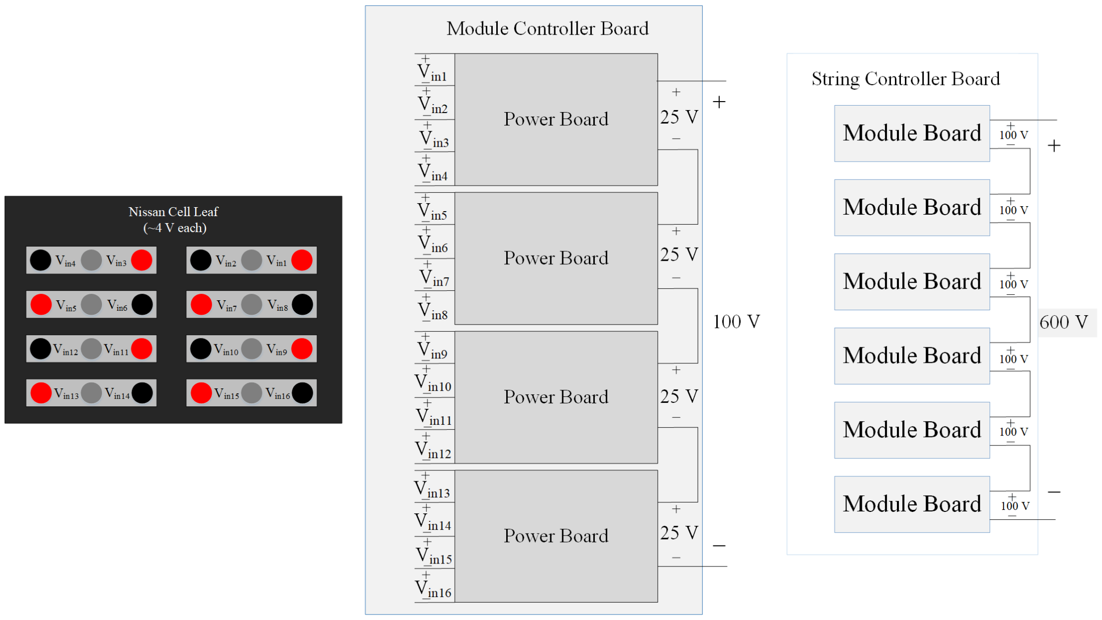


Fig. 4.10: Block Diagram portraying voltage ratings of the system for each unit

4.4 Control Loop Analysis

Active and Passive Cell Balancing can be done using input current, output current, or output voltage. This thesis bases input current regulation off of the paper done by Mohamed Ahmed [13] which allows for each cell to be regulated proportional to its capacity. The Pulse Width Modulation of the switches will provide current based on the resistivity of the output connected load, but will slightly differ based on circuitry and hardware differences between converter units. The circuit below in Figure 4.11 shows all of the feedback loops for the system. Each reference value depends on the initialization and enabling of the value or else the system will operate without any of the feedback regulation required for that reference value. The three feedback loops include: reference current regulation, droop current regulation, and differential current regulation. The control loops for each of these feedback controllers occur in the module controller where the microcontroller then sends the values of reference current to the FPGA's for each of the input ports to receive. These control loops, however, could be implemented in different locations depending on memory and processing power of each controller.

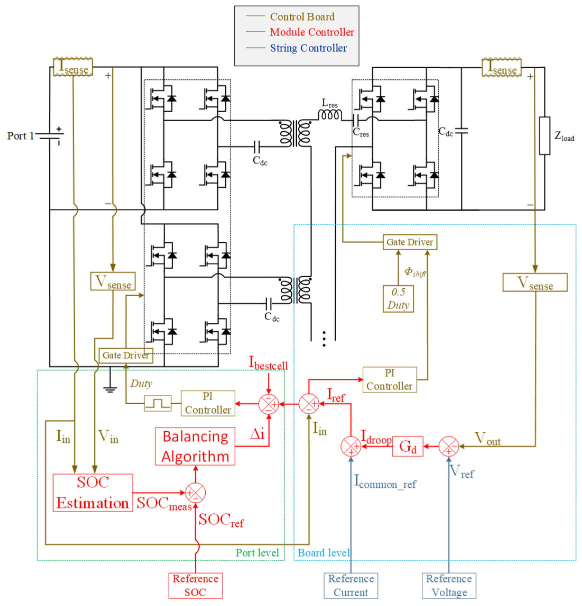


Fig. 4.11: Software diagram for the control of the Series Resonant Converter's of the system through String, Module, and FPGA Controllers

4.4.1 SOC Estimation

The SOC Estimation Loop takes the Voltage and estimates the SOC based on the OCV curve developed previously. When charging, however, State Of Charge becomes more difficult to estimate due to the current and temperature being applied at the terminals of the Nissan cell or any other battery that is being charged/discharged. The chemical change within the battery changes the state of charge but can not be directly measured by the terminal voltage to base along the OCV curve. Coulomb counting must then be used to approximate the SOC based on the capacity of the cell, the starting terminal voltage for the OCV curve starting measurement, and the temperature while in operation. This SOC Estimation code and the balancing algorithm used comes from a project from within Utah State University that has been modified to perform for balancing these Nissan Cells. The implementation of these algorithms are shown in Figure 4.12.

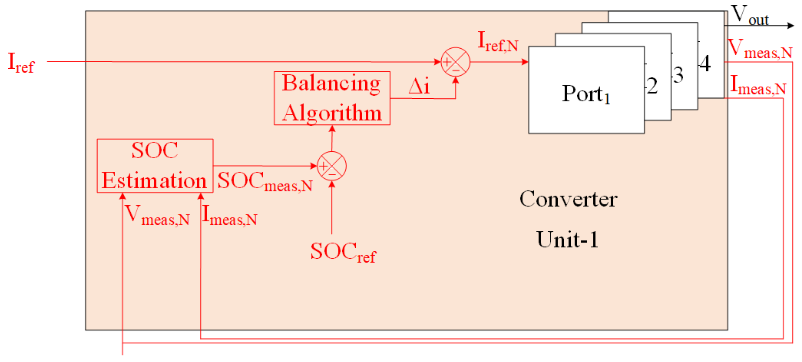


Fig. 4.12: Differential Current Control Loop based on Active Cell Balancing Algorithm

The estimated State of Charge of each port is then used in a feedback loop to regulate to a specific SOC reference that can be modified within each module controller. The balancing algorithm developed then balances each cell according to the active cell balancing process described earlier in this thesis. That algorithm will output a variation in reference current for each of the ports known as the differential current or δi. This allows for differential currents between each of the ports to perform the active cell balancing in a correct and efficient manner by sending the common reference current to the best cell (the highest capacity) and the lowest reference current to the worst cell (the lowest capacity) which is further described earlier in the Active Cell Balancing section.

4.4.2 Differential Current

The differential current control loop is portrayed below in Figure 4.13. This regulation is done within the FPGA, coded in Verilog, and each block is a sub-function inside of the top module of the Verilog code. As shown, the best cell will determine which cell has the highest reference current, and the ErrRatio variable will be determined by the difference between the reference and primary current. The duty will then be regulated in a sine waveform, thus an inverse sine function is needed to determine the duty of the primary.

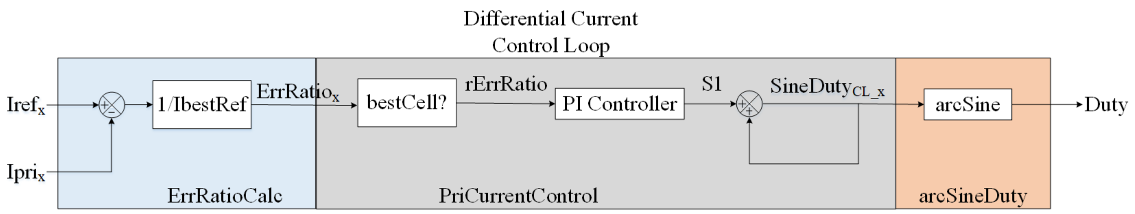


Fig. 4.13: Software diagram for the differential control of ports within FPGA

The best cell will output 0.5 as the duty and will have the highest phase shift at all times as well as a higher current which will be equivalent to the common reference current decided on by the user from the Site Controller . The phase shift of each of the other cells, between their primaries and the common secondary, will be determined by δi and will attribute to the difference between each port as well as the duty cycle due to the duty cycle being constantly 0.5 within the secondary ports of each converter unit. This process occurs for each of the four converter units within each of the six module units.

4.4.3 Droop Control for Voltage Sharing

The full voltage sharing, and current sharing loops are shown below in Figure 4.14. This figure shows each of the controller board's signals and processes being done for the communication. The current regulation loop is performed within the Module Controller where the ADC signals are being received and processed to send a reference current to the FPGA that is determined from the common reference current I_{ref}, the measured current for each port I_{meas}, and the droop current I_d; which is sent to each board and altered based on the δ_i of each port resulting from the differential control loop described in the preceding subsection. This droop current comes from the voltage sharing loop which regulates the output on each converter to the output of the overall bus voltage divided by the Number of Converters (N), also known as V_{ref}, which is sensed and computed within the String controller.

The reasoning for this droop control loop is due to a system instability of voltage sharing during the charging mode. When entering charge mode without any droop or PI control, the control will go unstable and the entire bus voltage being applied to the series output connections will only go to one of those modules. This stresses the Series Resonant tank and the output voltage ratings so it requires a PI controller to stabilize the voltage sharing across each individual module. The block G_d is a PI converter that takes the error between the voltage reference, described previously as the n-module string voltage averaged over n, and the actual voltage on the output of the converter and converts it into a droop current I_d which will end up regulating the reference current being given to each port in the converter unit.

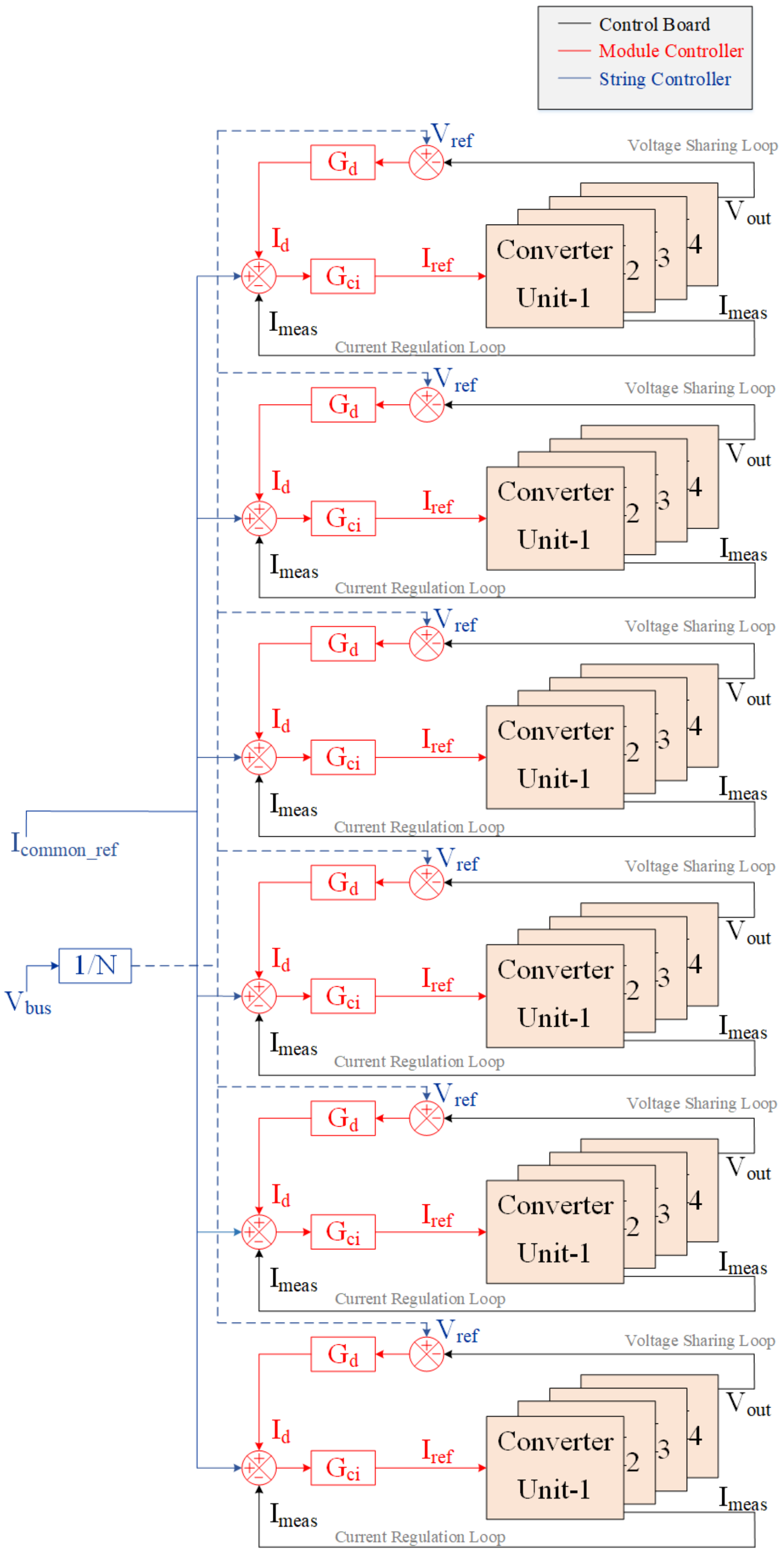


Fig. 4.14: Software diagram for the string and module level feedback controllers

CHAPTER 5
TESTING AND RESULTS

5.1 Preliminary Testing

5.1.1 MOSFETS and Gate Drivers

Converter efficiency and full functionality is necessary for full power capability. Each unit converter needs to be carefully inspected and tested so regulation can be precise and SOC estimation can be accurate in its capability to balance the cells actively. Tests were run to ensure that the switches would receive perfectly timed gate signals, and that the current sensors would output correct voltages for the Analog-to-Digital Converter's to read in with extreme bit precision. Figure 5.1 below shows the test setup used to calibrate the ADC's and to test the synchronicity of the gate drive signals.

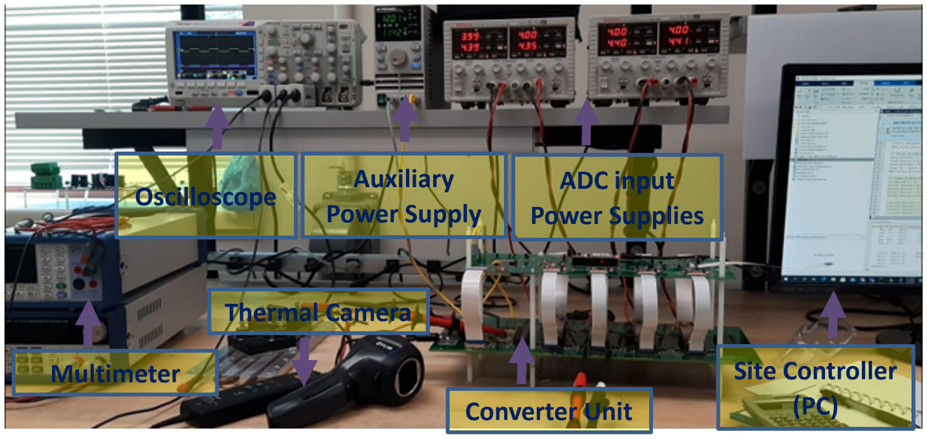


Fig. 5.1: Test supplies and setup for board calibration

The gate drive signals are directed from the FPGA directly through the Molex connectors to the switches. The return path for each signal needed to be similar for each signal to ensure timing was exact for each switch including the synchronicity of each port so the

GND return path for each gate drive signal was overlaid with the gate drive signal in design of the control board. This is because for full H-bridge functionality, the switches A and C must close at the exact time that switches B and D open for a full 0.5 duty cycle to occur, as shown in Figure 5.2. Anything less will cause the power transfer to diminish as the phase between primary and secondary will be affected negatively. Verification was done to ensure synchronicity at the same clock instance for each gate signal. This was done for both H-Bridges on the primary ports.

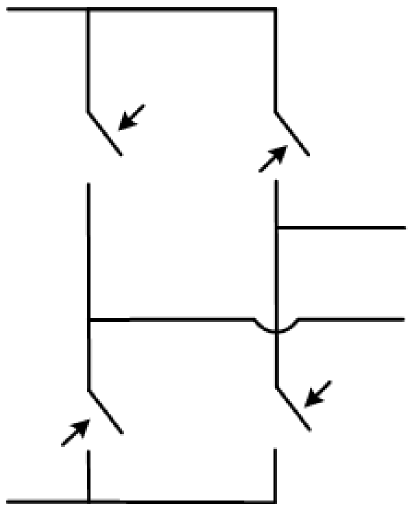
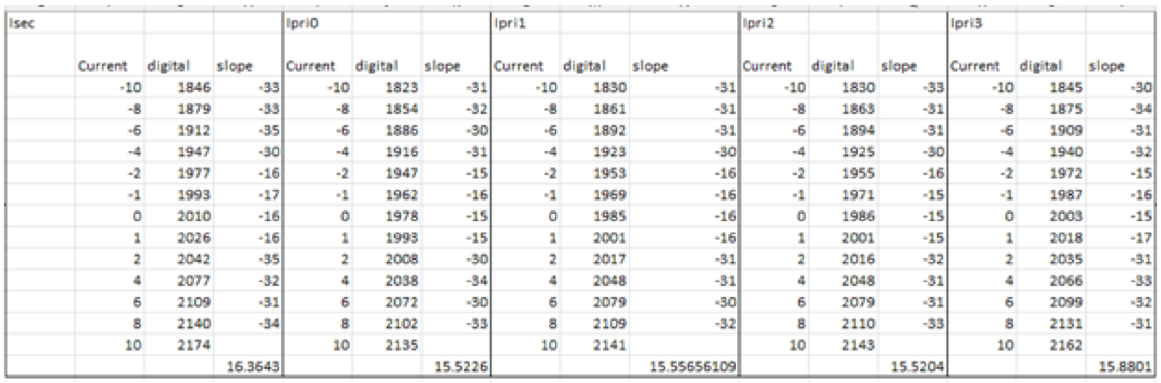


Fig. 5.2: Small level diagram of an H-bridge for DC/AC voltage conversion

5.1.2 Analog-to-Digital Converter Calibration

As discussed in previous sections, the measurements of Primary current and output voltage are used for regulation in different feedback control loops. The ADC readings are needed to calibrate each port and to be able to correctly identify the measurements for SOC Estimation and active cell balancing as well as voltage sharing using droop control. To do this, each port was calibrated for current and voltage on every board. The input current was especially important to have the correct slope of each current sensor so there were many points identified using a Sorenson Power Supply for accurate measurement up to the limit of the Power Supply of 10 A. The results are shown below in Figure 5.3. It is important to note that due to lack of supply of components, these current sensors were barely in stock and were not very consistent in slope from sensor to sensor. Previous sensors used for the proof-of-concept prototype were much more consistent and only required calibration of one port for each voltage divider provided (primary and secondary namely). These new current sensors provided the need to calibrate port by port and at times have a current sensor replaced due to faulty/irregular gains seen.



Isec				Ipri0			Ipri1			Ipri2			Ipri3		
	Current	digital	slope	Current	digital	slope	Current	digital	slope	Current	digital	slope	Current	digital	slope
	-10	1846	-33	-10	1823	-31	-10	1830	-31	-10	1830	-33	-10	1845	-30
	-8	1879	-33	-8	1854	-32	-8	1861	-31	-8	1863	-31	-8	1875	-34
	-6	1912	-35	-6	1886	-30	-6	1892	-31	-6	1894	-31	-6	1909	-31
	-4	1947	-30	-4	1916	-31	-4	1923	-30	-4	1925	-30	-4	1940	-32
	-2	1977	-16	-2	1947	-15	-2	1953	-16	-2	1955	-16	-2	1972	-15
	-1	1993	-17	-1	1962	-16	-1	1969	-16	-1	1971	-15	-1	1987	-16
	0	2010	-16	0	1978	-15	0	1985	-16	0	1986	-15	0	2003	-15
	1	2026	-16	1	1993	-15	1	2001	-16	1	2001	-15	1	2018	-17
	2	2042	-35	2	2008	-30	2	2017	-31	2	2016	-32	2	2035	-31
	4	2077	-32	4	2038	-34	4	2048	-31	4	2048	-31	4	2066	-33
	6	2109	-31	6	2072	-30	6	2079	-30	6	2079	-31	6	2099	-32
	8	2140	-34	8	2102	-33	8	2109	-32	8	2110	-33	8	2131	-31
	10	2174		10	2135		10	2141		10	2143		10	2162	
			16.3643			15.5226			15.55656109			15.5204			15.8801

Fig. 5.3: Analog to Digital Conversion tables for each current sensor - one board only

The Analog-to-Digital Converters (ADCs) used were new to the project and new to Utah State University. The component was chosen due to the low price and provided a different challenge to other ADCs previously used. While the ADCs had a default range of -12 to 12 V for the 4095 bits, the current sensors used provided a default voltage of 2.5 V with bias voltages that only ranged from 0-5 V so there were only 2-3 bits of precision between each tenth of an amp. This was not sufficient for current regulation for the input current feedback loops. The ADCs were also used to translate analog voltage values to digital values for the FPGA to read, none of which required any more than 5V after the voltage dividers of each circuit. The range could thus be changed using the SDI line, which was unique to these ADCs and provided more bit precision, using pre-programmed messages when CS went low as prescribed in the diagram shown in the datasheet in Figure 5.4. With a V_{ref} of 3.3 V, the correct message to configure the Range_Selection_Register was 1.25 x V_{ref} x 1.25 = 1011b. The exact timing and detail of the message being sent took several days to configure through FPGA code and to generate bitstreams. With the correct timing, the final message being sent was 32'b11010000000101000000000000001011 which can be seen in the FPGA code for the ADC in the appendices. Figure 5.5 below shows the message being sent from FPGA to the ADCs to modify the range selection array for more accurate readings between ADC to FPGA.

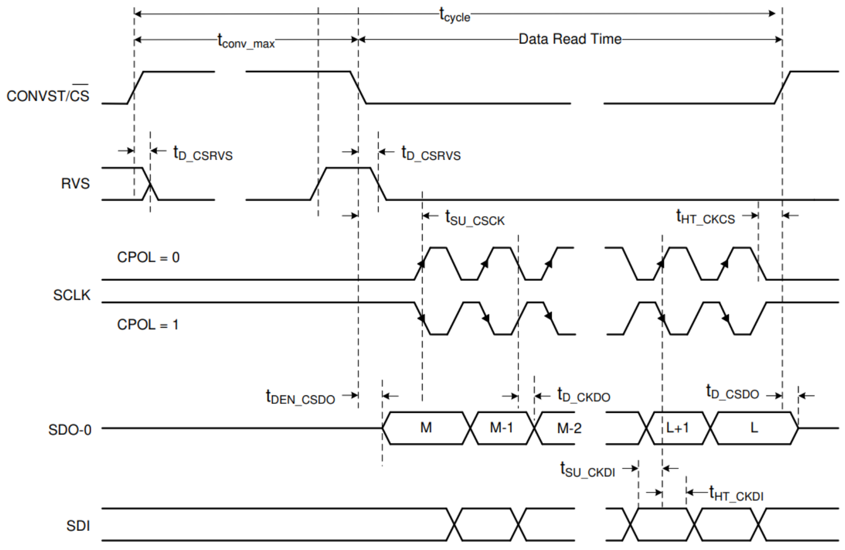


Fig. 5.4: Timing Diagram of the SDI line used for Range Selection of the ADC

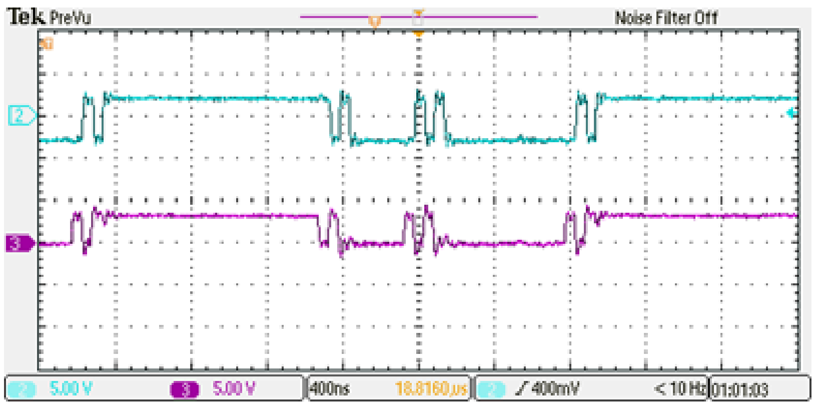


Fig. 5.5: SDI message being sent from FPGA to ADC

5.2 Converter Unit Testing

Upon full fabrication, population, assembly, and calibration of the boards, including the design and tuning of the magnetics for the Series Resonant Converter discussed earlier, each board was ready for power transfer and needed to be tested for full integration to the system. The boards were tested under maximum power (at 500 W per board) and various software control loops were tested with the board architecture. Rather than immediately stepping into the full software of the system as discussed for Figure 4.11, control loops were gradually enabled by the FPGA using controlled parameters and references within MATLAB. The test setup is shown below in Figure 5.6 with the Arbin Cycler initially being the power source for the Converter Units before connecting the Nissan Cells. The Arbin Cycler will be suitable for validating the function of the Converter Units, and the Gustav Klein will function as the constant voltage source on the output.

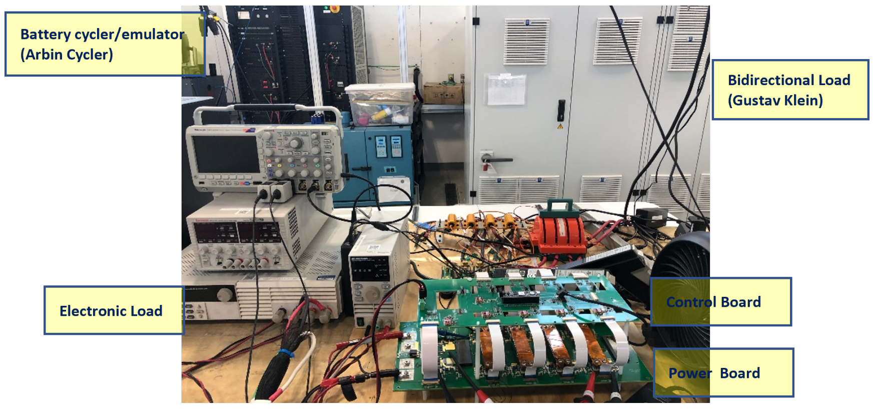


Fig. 5.6: Test Setup for Converter Unit functionality verification

5.2.1 Open Loop Control

The boards were first tested in Open Loop Control using the software diagram shown in Figure 5.7. These tests were to verify that the current of the system could be regulated in buck/discharging mode (positive input current) as well as in boost/charging mode (negative input current) of the battery cells. This was done by shifting the phase of the secondary so the current of the primaries would go up and down. At 0 degree phase shift the input current was measured by the ADCs to be nearly 4 A but slightly differed, as discussed in the previous chapter. Once phase shifting the secondary phase validated the variable nature of the input current and the tank current was consistently in buck or boost operation within prescribed limits, the closed loop testing of the verification could begin.

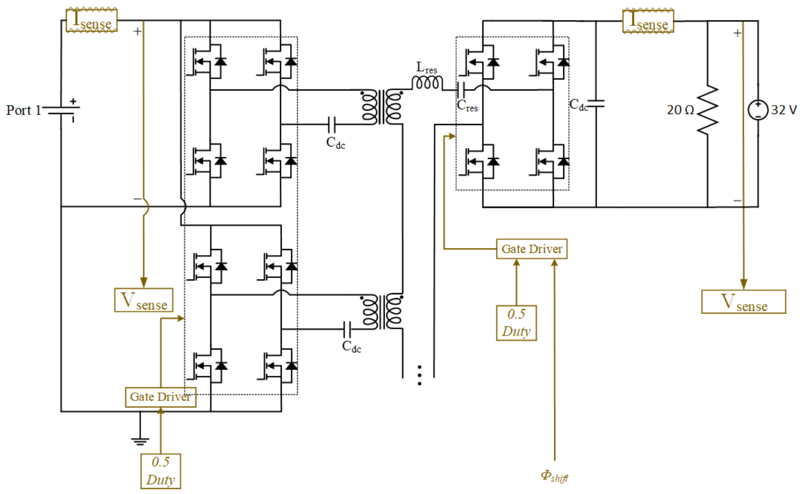


Fig. 5.7: Software Diagram of Open Loop Control Testing (Phase Shift only)

As shown below in Figures 5.8 the input current of an Open Loop system ramped up to 0.5 Duty Cycle in this instance was around 4.83 A. In this boost mode, the positive current is discharging the battery and the tank current can be seen rising with the voltage going through the switches. The negative current, on the other hand, is shown in Figure 5.9 which is charging the cell in buck converter mode, and the tank current of the Series Resonant Converter is falling when the voltage going through the switches is rising. This behavior permits the buck converter to charge the batteries with the opposite polarity of tank current providing negative input current to the input terminals.

5.2.2 Closed Loop Control

Closed loop regulation was primarily tested with a controlled parameter for reference current. This feedback loop can be seen in the following Figure 5.10. As shown in the Figure 5.10, the feedback controller is designed to regulate the current to a specified ”refer-

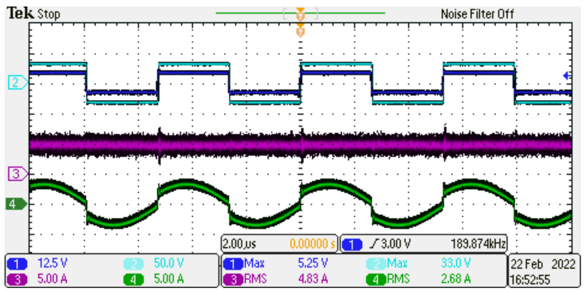


Fig. 5.8: Open Loop Control Waveform of 0 degree phase shift

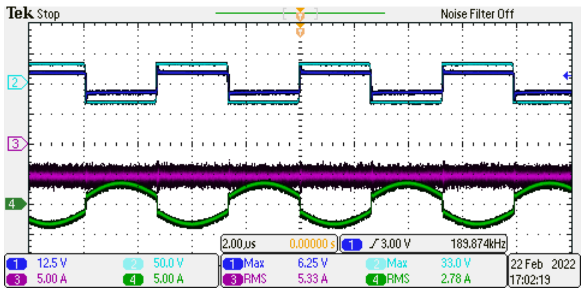


Fig. 5.9: Open Loop Control Waveform of -4 degree phase shift

ence current" I_{ref} by using ADC readings of the primary currents and a proportional and integral gain to change the current by shifting phase automatically until the reference value has been met.

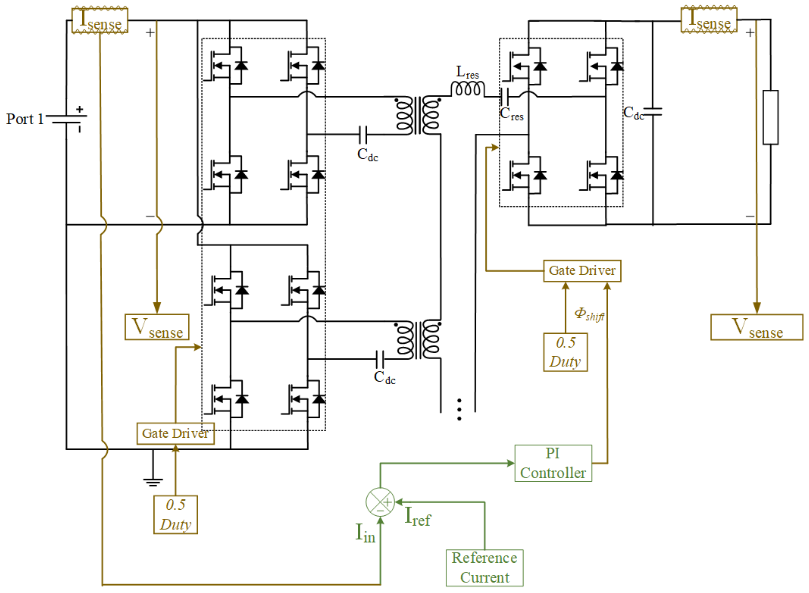


Fig. 5.10: Software Diagram of Closed Loop Control Testing (Reference Current command only)

These tests were performed using the Gustav-Klein (GK) as a bidirectional voltage source in parallel with a resistive load on the secondary. The Duty sent to the primaries remained 0.5 as well as the secondary, while the only parameter being changed was the phase shift of the secondary controlled by the current regulation loop. With the feedback controller automatically shifting the phase between the primary and the secondary, the system was tested for an output power between 612 W output and -509 W input as shown in the oscilloscope waveforms in Figures 5.11- 5.12 below.

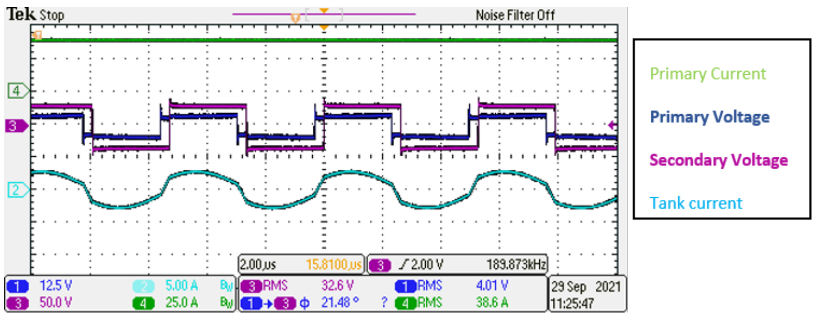


Fig. 5.11: Current Regulated Control Waveform. I_{ref}=40A, P_{out}=612W

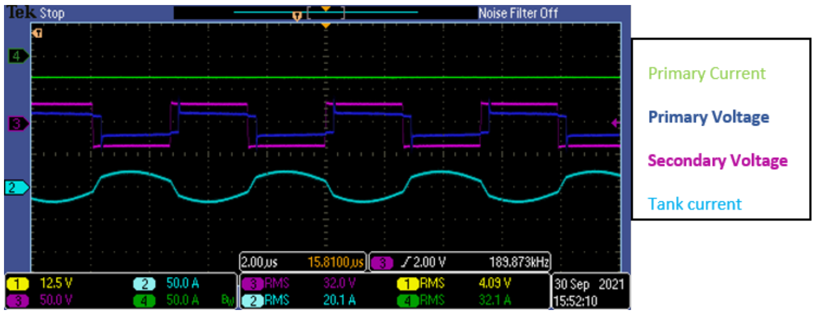


Fig. 5.12: Current Regulated Control Waveform. I_{ref}=-32A, P_{out}=-509W

Power Converter efficiency is also necessary to maximize the power being transferred and to minimize the losses in the system. The scaled-down prototype of the system operated at about 99.4% at peak efficiency and maintained an efficiency above 90% in the range of power the converter will be tested at. Figure 5.13 shows the efficiency of the new converter to be at 99.2% at maximum power and maintains efficiency above 90% for the entire range of operation needed. The values of positive and negative power will be a little off due to the margin of error inherent to Analog to Digital conversion, but the curve of the system and values will remain relatively the same. These values can be compared with the old version and the goal desired for efficiency which was provided by a Yukogawa Efficiency measurement device that is highly accurate in the Figure 5.14. With the similarity in values and efficiency curve, the closed loop operation of the converter units was verified to be fully functional and validated the control loop and circuitry of the new power converter.

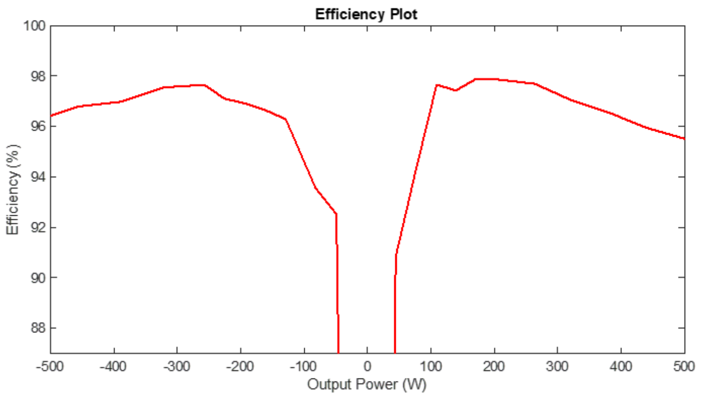


Fig. 5.13: Closed-loop operation efficiency curve of 100 kW system

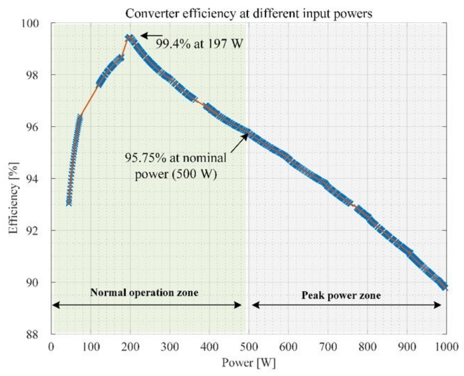


Fig. 5.14: Closed-loop operation efficiency curve of POC system

5.2.3 Differential Current Control Loop

The closed-loop feedback controller was then tested in “differential current control” mode where differential currents could be tested between each of the power ports to validate the ability to perform active balancing under the objective mapping. This feedback loop can be seen in the following Figure 5.15. The test was performed similar to the previous version of testing with the GK, only now each of the four ports would have different reference currents according to the differential current established between each of the ports. The results below show the test of current regulation between three different scenarios: reference current being equal for each of the ports, differential current being applied to each of the ports, and reference current being equalized once again. The test validated the differential mode with little to no transients in the primary ports for each of the steps as shown below in Figures 5.16- 5.19.

There was also software in the system to protect from faulty ADC readings in marginal δ_i values resulting in unstable activity. This software validated by ramping from -20 A

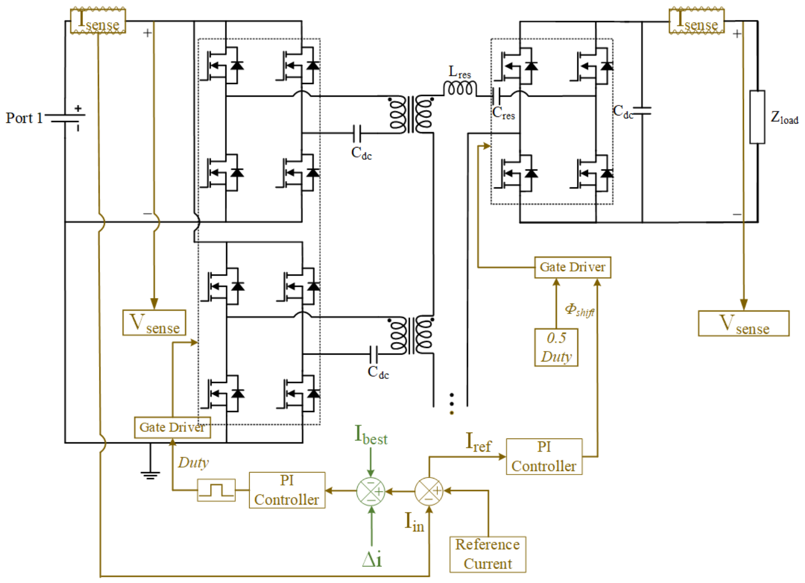


Fig. 5.15: Software Diagram of Differential Current Testing

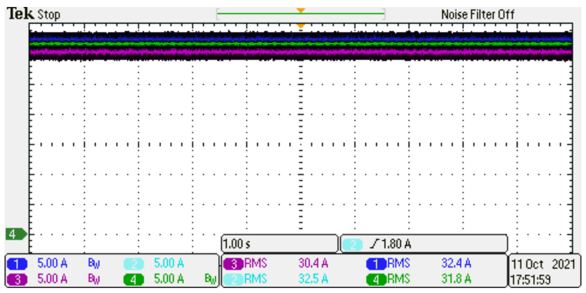


Fig. 5.16: Differential Current Operation using same Iref

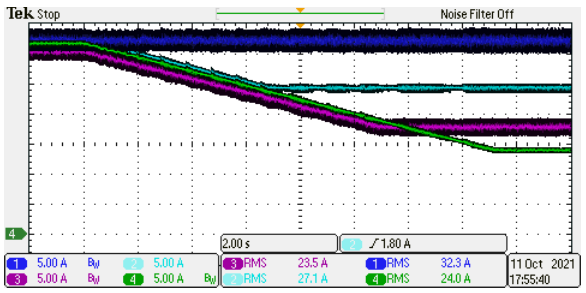


Fig. 5.17: Differential Current ramping using differential current

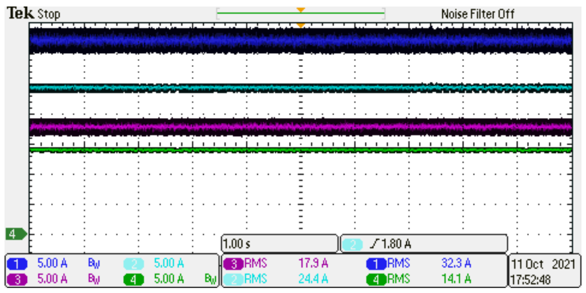


Fig. 5.18: Differential Current Operation using differential current

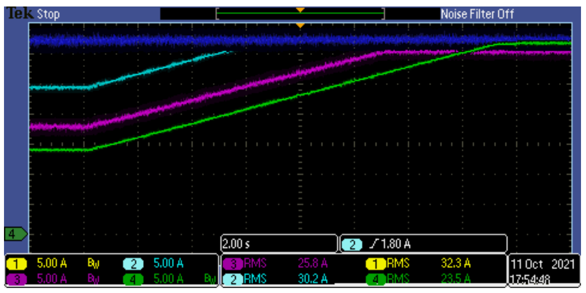


Fig. 5.19: Differential Current ramping to same Iref

(peak current) differential current to +20 A differential current and viewing the switching on and off of differential current control. This can be seen in Figure 5.20. There were slight transients when settling back to negative current, but nothing that the system can't handle. This performance also only seemed to appear when entering negative differential current after going through this off/on process of differential current so it was not something that would mess with the system. It is important to note that ADC precision was necessary for δ_i to stabilize so various ADCs needed to be replaced with proper gains and offsets similar to others in the system. This took time, but resulted in stable differential control in both the positive and negative current realm. This concluded validation for the integration of the power and control board, preparing the way to test a full module before populating all boards for final system operation.

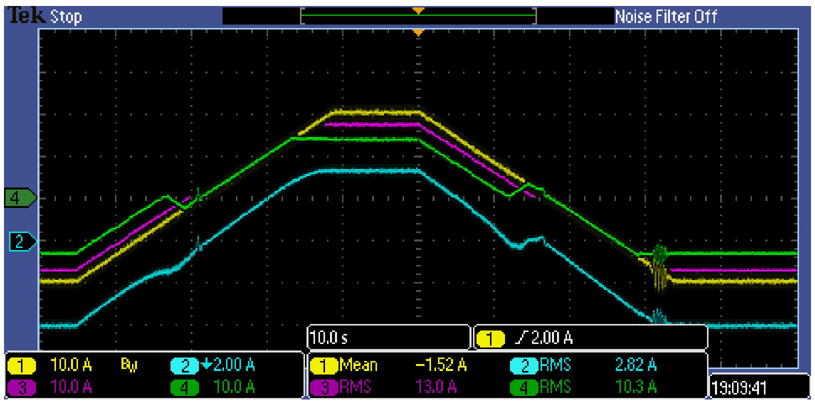


Fig. 5.20: Differential Control ramping from peak currents of -20 A to 20 A to -20 A

5.3 Module Testing

Modular Testing was important to synchronously send signals from the Microcontroller in the module controller to each of the converter units for regulation and ramp-up for active cell balancing. The setup used can be seen below in Figure 5.21. The initial testing found problems with the Pull-down resistors and the signals being sent from the four converters were at too low of a voltage to reach the threshold voltage needed to communicate between the two boards, so an OR-gate was implemented to the module controller which would enable communication from four boards at a time, which was needed for a fully functional module. The testing of the new format of module controller board indicated synchronous startup of two of the modules (1 and 3) in the 4 board setup, as shown in Figure 5.22 which validated the simultaneous ramping of each of the boards.

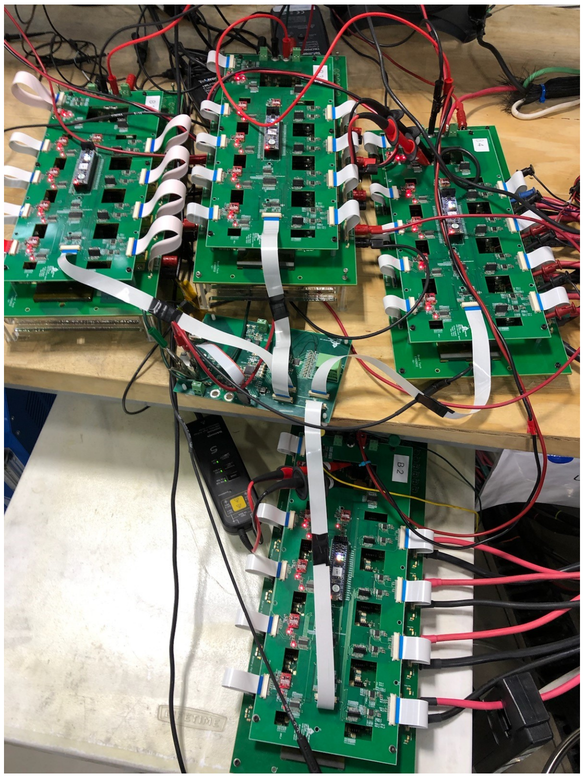


Fig. 5.21: Initial Module Testing setup with four modules in series

The stability of the initialization of each of these boards required a resistive load connected in parallel to the output of each of the converters. This would prevent any transients that could go over the voltage limit or ratings of the secondaries when the full string voltage may be applied to one converter. Figure 5.23 indicates this transient performance with no resistive load on the secondary while the waveform shown in Figure 5.24 shows the resulting ramping of the system with a resistive load. These tests were only done with 2 boards with a 50 V bus voltage to prevent any high voltage on the secondaries, as discussed. The Figure 5.24 also shows the performance of the converters in open loop with a slight variance in

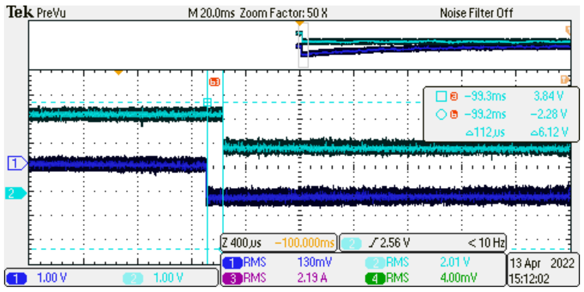


Fig. 5.22: LED waveform indicating synchronous startup of two of the converters

output voltage, about 23.5 and 26.5 V respectively. This is due to inevitable variations in the circuitry and components of each of the secondaries and will require the droop control as discussed previously in this thesis.

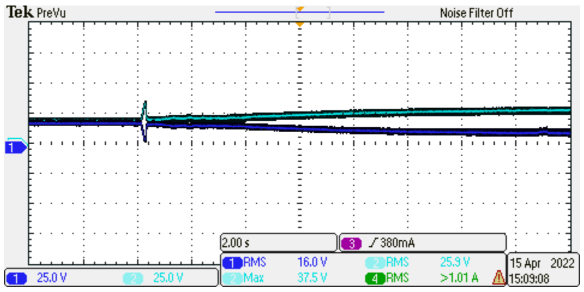


Fig. 5.23: 2 SRC Converters starting up with no resistive load

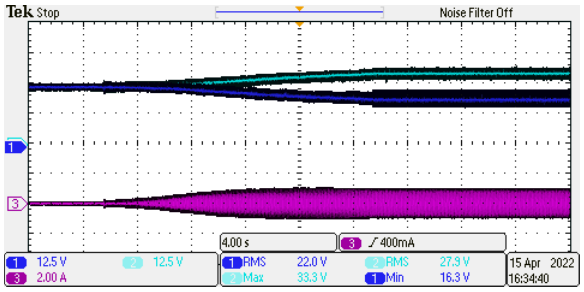


Fig. 5.24: 2 SRC Converters in series with a 50 V resistive load ramping to open loop (no current feedback) performance

The current regulation of this system will also require some droop current to stabilize the output voltage of the system to equally share among each converter as shown in

Figure 5.25. This was enabled with a proportional game that acted as a resistor to alter the current to compensate for the different voltages. The discharging portion of the system was able to basically voltage share as shown previously in Figure 5.24, but for the charging portion of the system the converters would not be able to operate in series and share the voltage at all. Rather, one converter would take all the voltage which would be deemed unstable and not fit for operation of active cell balancing due to a negative resistance inherent to the buck converter series connected operation. The droop constant was then put into a variable equation to act as the droop control discussed previously in Figure 2.4 [31]. The positive to negative current validation is shown in Figures 5.26- 5.27 where the droop is seen changing when surpassing that 0 A threshold and entering charging mode.

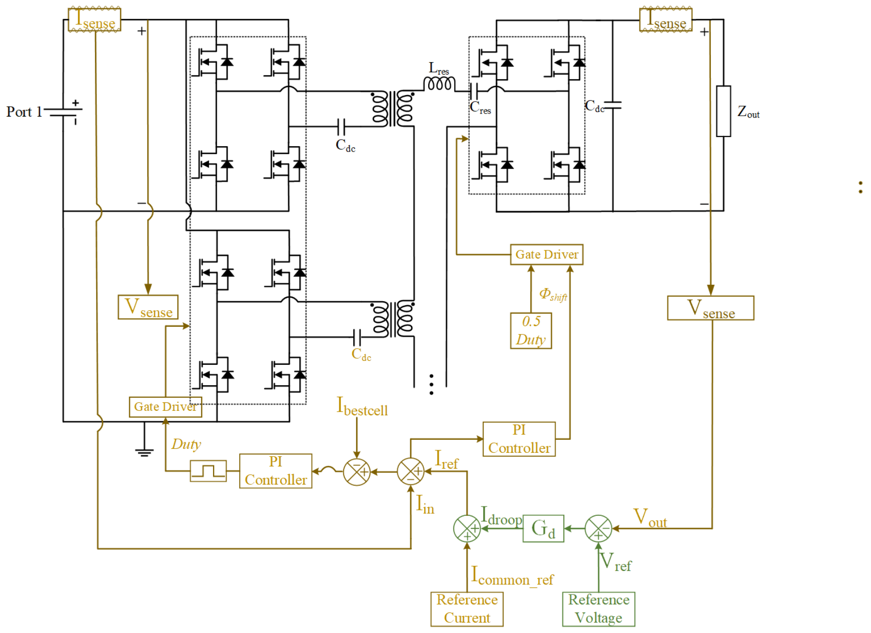


Fig. 5.25: Reference Current regulated with Droop Control for Voltage Sharing

This behavior needs to be validated for all four boards as shown in Figure 5.28 while connected to the Nissan Cells. This connection is shown below in Figure 5.29

5.4 String Testing

Series connection of 6 modules also needs to be tested as shown in Figure 5.30

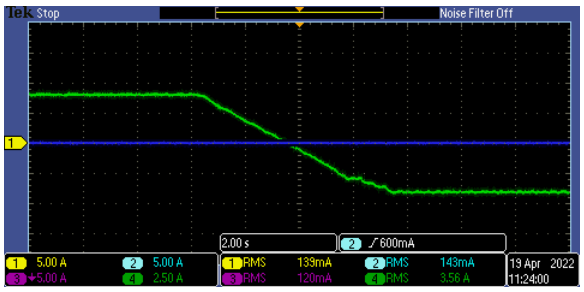


Fig. 5.26: Droop Control from positive to negative current

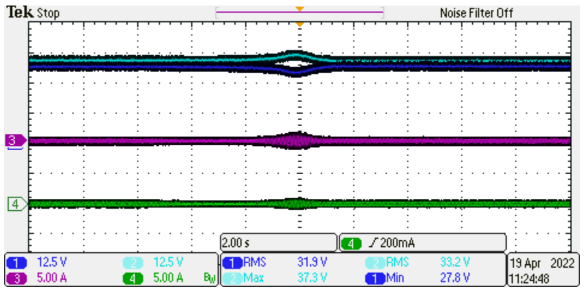


Fig. 5.27: Droop control stability transitioning from positive to negative current

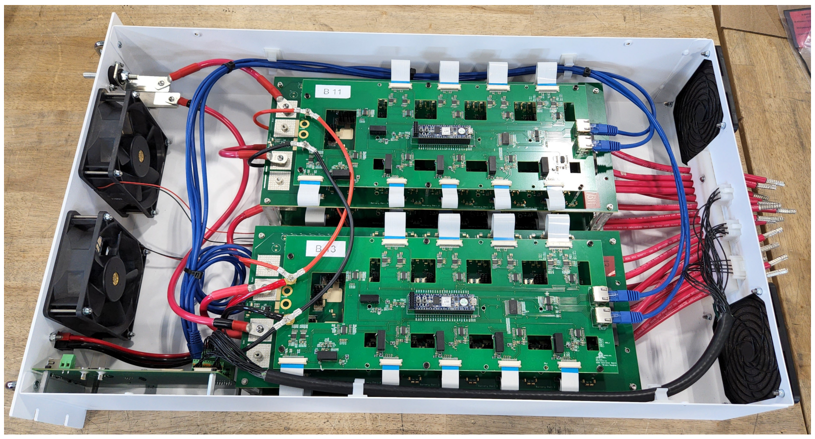


Fig. 5.28: Module Enclosure for testing of 4 SRC converters in series

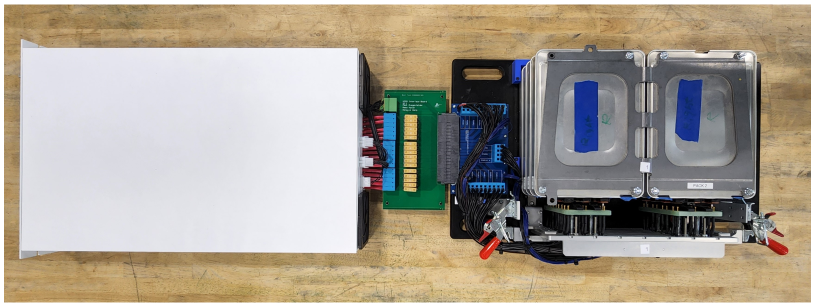


Fig. 5.29: Module Connection to Nissan Cells

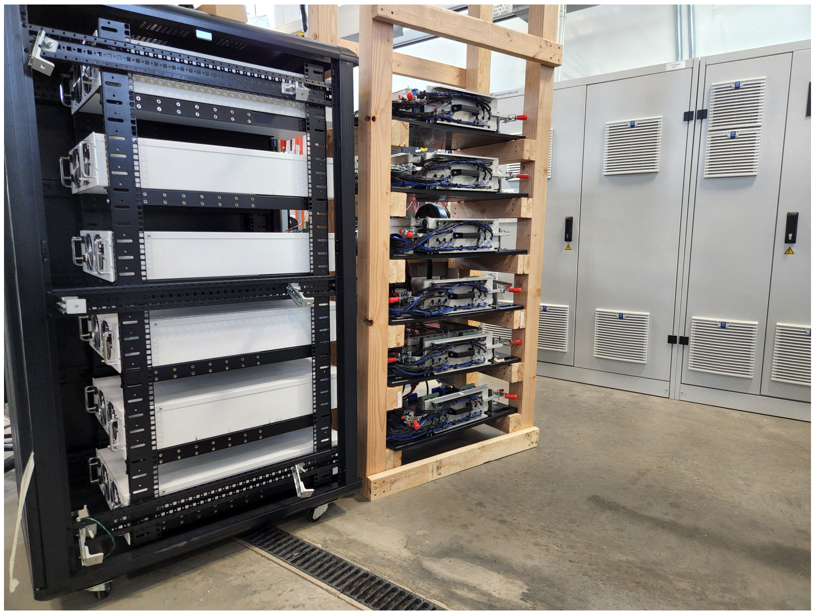


Fig. 5.30: Full string consisting of 6 modules and 6 Nissan Cell tubs

CHAPTER 6

HOT SWAPPING PROCESS AND ANALYSIS

The following is a Digest submitted to COMPEL pending approval:

Hot Swapping Analysis and System Operation of Series Connected Converters in Active Battery Reconditioning Systems for Second Life Battery Application

Abstract: This paper proposes analysis, control strategy, and procedure to achieve hot swapping operation of series connected power converters in an active battery reconditioning system for second life battery application. The proposed approach, control implementation, and transient analysis during the hot swapping are presented. The proposed approach allows the whole system to operate seamlessly and without any interruption when battery packs need to be disconnected for maintenance or replaced. A 100 V, 2 kW hardware system with 4 series-connected 25 V, 500 W converters is tested to validate the proposed analysis and operation.

6.1 Introduction

Series connected converters are commonly used in Lithium-Ion battery energy storage applications such as grid supplements, Battery Management Systems (BMS), Battery Energy Storage Systems (BESS), and other 2nd life battery applications. Hot swapping, or removing and replacing individually, of battery cells connected to these converters is vital for many systems to enable continuous operation while a converter's cells may concurrently be replaced for a variety of reasons [32] that could include but are not limited to: replacing battery cells that have failed or require repairs, fixing connections between converter and a battery cell, or to simply replace cells that are no longer needed in the series connected architecture with cells that are more desirable. There are various challenges and problems with hot swapping so it must be done with care, otherwise the battery may be damaged due to the peak current exceeding the nominal current of the system.

This paper develops the procedure and control needed for these types of systems to allow seamless hot swapping. This series connected architecture requires monitoring and analysis of the inrush of output voltage as well as the tank current of the resonant tank as it is distributed across the rest of the cells in the system to avoid surpassing current and voltage ratings while disconnecting the cells and re-enabling full capability of the system with the newly connected cells.

This digest presents the theoretical approach of the hot swapping and the performance needed for stable operation of the series connected architecture. A scaled-down prototype will then be tested to validate the approach and ensure safe, stable performance of the hot swapping procedure. More detailed analysis and expanded experimental results will be provided in the full paper.

6.2 Theoretical Approach, Simulation, and Hardware Results

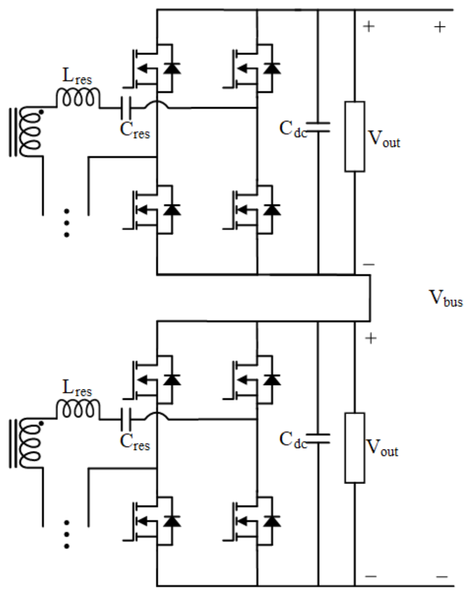


Fig. 6.1: Series-connected output in a module

The hot swapping analysis and implementation work in this paper is for a novel Heterogeneous Unifying Battery (HUB) reconditioning system that cycles Battery Powered Modules (BPM) to unify cells' state of health (SOH) to improve their second life battery performance after retired from electric vehicles (EV) such as Nissan and Tesla [33].

The HUB reconditioning system consists of series and parallel connected DC/DC power converters to actively balance SOH of the cells in the retired battery modules while providing non-interrupted power to the grid. After a period of reconditioning in the HUB system, the battery module will achieve homogeneity in capacity and will be removed and replaced by a heterogeneous retired battery module [28]. The process of replacing the homogeneous BPM with a heterogeneous BPM to be reconditioned should not interrupt the system operation or impact any grid services.

Figure 6.2 shows the architecture and circuitry topology between each of the battery cells being connected in series through a secondary winding transformer that allows the input voltage to meet the needs of the high voltage bus connected to the output. This high-frequency series connected secondary winding is a novel idea working to minimize component count and the contact resistance required to connect each output in series to connect to the output bus [7].

The proposed architecture of this hot swapping is to hot swap one of the n series connected DC/DC converter modules to enable the active cell reconditioning for a large number of battery cells while continuing operation of other cells that have reached their second life. The architecture is shown in Figure 6.1 connecting the secondary outputs in series together to account for the voltage on the output bus. The architecture for the full system includes four DC/DC Converter boards per module and has 6 modules connected in series for a full string, rated up to 800 V, that can have hot swapping during operation of all 6 modules in active cell balancing. These 6 modules with four converter boards per module will enable reconditioning of 96 cells from Nissan Leaf. This architecture will be shown and discussed in the full paper.

Voltage sharing across series connected power converter's is an emphasis of study for BPM. Perfectly equivalent circuits regulate the voltage equally, but component tolerances are inevitable and current regulation slightly differs due to the current sensor hardware differences between the two boards, inherent to current sensors, thus a solution is required to share the voltage equally, namely a droop control loop [31]. A PI controller has been implemented in the system to regulate the output voltage equally between each converter according to the total voltage of the system operating in series connection. This PI controller is modelled as the gain G_d in Figure 6.3 and is used to regulate the input reference current of the system. The reference current is used to control the input current of each port and will effect the output current of the system in a manner that will allow equal voltage sharing among the series connected modules.

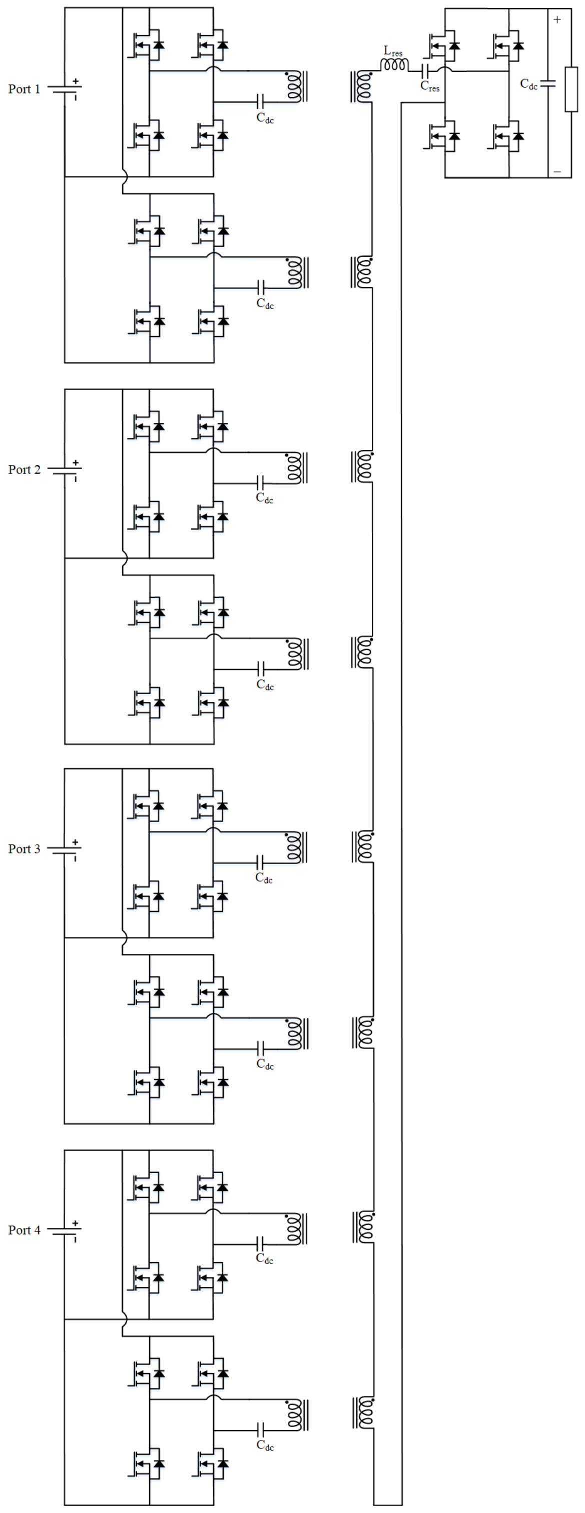


Fig. 6.2: Power converter architecture for 4-port Series Resonant Converter

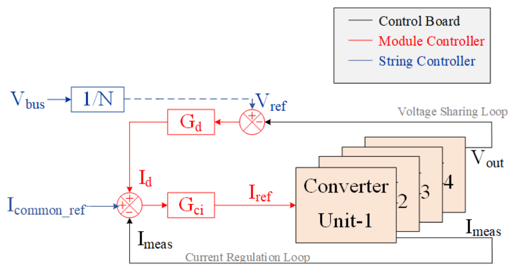


Fig. 6.3: Power converter architecture for 4-port Series Resonant Converter

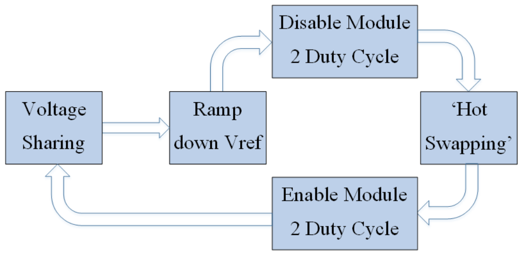


Fig. 6.4: Block Diagram modelling the process needed to perform Hot Swapping during full operation successfully

This process requires a regulated rampdown so controls were designed allowing hot swapping to be performed in a reasonable amount of time. This process was designed and simulated, before programming the control of the system, to allow the series connected system to operate as prescribed. Figure 6.4 shows a block chart describing the process needed to enable hot swapping of a specified module and the to re-enable full operation upon successful insertion of the new module.

As shown in Figure 6.4, when the system is in full operation, sharing voltage among the series connected converters, the reference voltage is slowly ramped down such that the n-1 modules begin to take the full DC bus voltage according to the droop control loop discussed earlier. The pulse-width modulation (PWM) control of the system enabling duty cycle control of each module is then changed so the module(s) still in operation continue with full duty cycle for the entire port (differential current will alter the duty cycle according to the SOC and SOH of the cells connected to each port), while the module being hot swapped discontinues duty cycle operation. Primary and tank current are then regulated to zero, denoting the instance needed for safe removal of the cells.

A transient analysis of a system with two modules connected in series was done of the preceding test procedure using PLECS simulation shown in Figure 6.5- 6.6 with the module 1 being the module continuing operation and the module 2 being hot swapped. The ramping of the voltages is shown as prescribed in the block diagram, with the slope of the rampdown getting larger as the Duty Cycle is disabled for Module 2. The transients seen in the output and bus current occur when PWM is disabled out of precaution to ensure

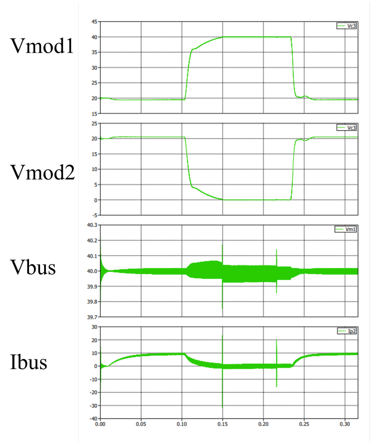


Fig. 6.5: Simulation waveform of Module and Bus voltage during hot-swapping

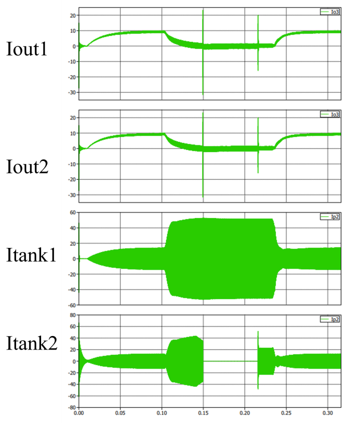


Fig. 6.6: Simulation waveform of Output and Tank current during hot-swapping

0% duty cycle, but the small transients of the system are well within the current ratings of the system. Also the LC resonant tanks of both modules do not exceed their ratings of 80 A and don't have any transients when disabling the PWM. As seen from the results of the simulation, the currents of both primaries and the secondary go to 0 A, due to the stress on the tank from one module's output bearing all of the voltage. This confirms the theory stated at the beginning of this section regarding the stress on the system when trying to send all of the shared voltage to just one module resulting in 0% efficiency. The figure also shows that both converters can share voltage after hot swapping, validating the process proposed.

The results of the hot swapping validation tests using a scaled-down hardware prototype are shown in Figure 6.8. The prototype's testing setup is shown in Figure 6.7, where the system has two converter modules connected in series to a voltage bus of 40 V which will be equally shared until hot swapping requires one module to bear the full voltage. The process of hot swapping described in the previous section can be seen as the system stabilizes to 20 V at each converter output after performing the hot swapping. Due to the limited number of series connected modules in the hardware setup, the series resonant tank experienced more stress, which will not occur when there are more converters in series, in the full system, to share the voltage of the high voltage bus.

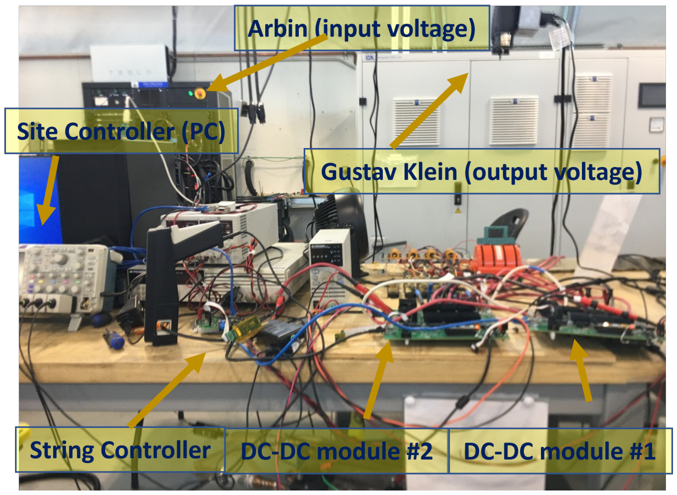


Fig. 6.7: Test Setup used for Hot Swapping of Scaled-down Prototype

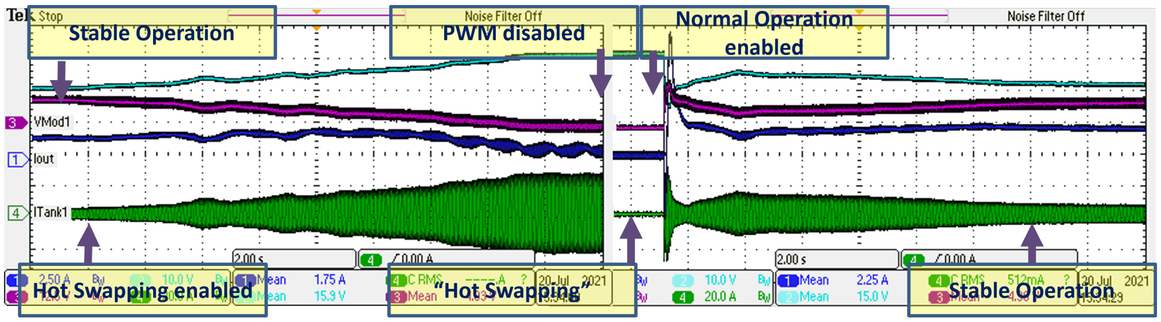


Fig. 6.8: Hardware Test of Scaled-down Prototype validating Hot Swapping Strategy

6.3 Conclusion

A hot-swapping approach for series connected converters is proposed in this work using voltage droop control. The control approach is able to regulate the voltage of the n series-connected converters while one module is hot-swapped. Performance is validated through simulations and experimental results from a scaled-down 50 V, 1 kW hardware system with 2 series-connected 25 V, 500 W converter modules. More details on the analysis, control, and implementation will be provided in the full paper along with experimental results of more series-connected modules.

CHAPTER 7
CONCLUSION

7.1 Completed Work

This thesis has entailed the design, control, and results of output series connected Series Resonant Converter boards. These boards have shown the ability to perform differential current feedback control loops necessary to perform the validated Active Cell Balancing algorithms. These series connections are capable of up to 600 V for the entire voltage bus on the secondary. Communication was established at a 1kHz frequency using SPI between 1 module controller board and 4 converter units, and CAN communication between 1 string controller and 6 module controller boards. Hot Swapping was also validated for the system to enable continuous balancing of an infinite number of cells, as modules are replaced.

Control Loops were designed for regulating the input current of the system for the Active Cell balancing algorithm, with a reference current, for the best cell, inputted by the user through the Site Controller (PC) to String Controller Board. The control loop for droop control was also implemented with a PI Compensator balancing the voltage for equal voltage sharing among each module and converter unit. The magnetics and component design and production for the SRC as a whole were also validated for efficient power transfer through each of the boards, and minimal losses through the system.

This enables the balancing of Nissan Cells that have reached their EOL stage of First-Life use in Electric Vehicles for second life uses. This homogeneity in battery cell's capacity enables efficient and longer lasting second-life use in Battery Energy Storage Services, grid services, and other applications consisting of the lower capacity, higher internal resistance in low power density applications. This will create more efficient uses for Lithium-Ion batteries and provide less detrimental environmental impacts from when batteries are taken out of first-life use.

REFERENCES

[1] R. Zhang, B. Xia, B. Li, Y. Lai, W. Zheng, H. Wang, W. Wang, and M. Wang, "Study on the characteristics of a high capacity nickel manganese cobalt oxide (nmc) lithium-ion battery—an experimental investigation," *Energies*, vol. 11, p. 2275, 08 2018.

[2] J. Rivera-Barrera, N. Munoz, and H. Sarmiento, "Soc estimation for lithium-ion batteries: Review and future challenges," *Electronics*, vol. 6, p. 102, 11 2017.

[3] M. A. Hannan, M. M. Hoque, A. Hussain, Y. Yusof, and P. J. Ker, "State-of-the-art and energy management system of lithium-ion batteries in electric vehicle applications: Issues and recommendations," *IEEE Access*, vol. 6, pp. 19 362–19 378, 2018.

[4] L. A. Perişoară, I. C. Guran, and D. C. Costache, "A passive battery management system for fast balancing of four lifepo4 cells," in *2018 IEEE 24th International Symposium for Design and Technology in Electronic Packaging (SIITME)*, 2018, pp. 390–393.

[5] S. Bal, D. B. Yelaverthi, A. K. Rathore, and D. Srinivasan, "Novel active rectification for extended zvs operation of bidirectional full bridge dc/dc converter for energy storage application," in *IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society*, Oct. 2018, pp. 2103–2109.

[6] F. Krismer and J. W. Kolar, "Efficiency-optimized high-current dual active bridge converter for automotive applications," *IEEE Transactions on Industrial Electronics*, vol. 59, pp. 2745–2760, 2012.

[7] R. H. D. B. Yelaverthi, M. Kamel and R. Zane, "High frequency link isolated multi-port converter for active cell balancing applications,"," in *2019 20th Workshop on Control and Modeling for Power Electronics (COMPEL)*, 2019, pp. 1–7.

[8] N. J. M. Koseoglou, E. Tsioumas and C. Mademlis, "Highly effective cell equalization in a lithium-ion battery management system," *IEEE Trans. Power Electronics*, vol. 35, pp. 2088–2099, Jul. 2020.

[9] J. Wooten, "Control of series connected battery powered modules," Ph.D. dissertation, Utah State University, Logan, UT, 2020.

[10] I. V. S. D.-I. S. M. S. A. W. J.-M. T. S. G. N. O. Martinez-Laserna, E. Sarasketa-Zabala and P. Rodriguez, "Technical viability of battery second life: A study from the ageing perspective," *IEEE Transactions on Industry Applications)*, vol. 54, pp. 2703–2713, Jan. 2018.

[11] K. H. R. Z. D. C. M. Evzelman, M. M. Ur Rehman and D. Maksimovic, "Active balancing system for electric vehicles with incorporated low-voltage bus," *IEEE Trans. Power Electronics*, vol. 31, pp. 7887–7895, Nov. 2016.

[12] W. Huang and J. A. A. Qahouq, "Energy sharing control scheme for state-of-charge balancing of distributed battery energy storage system," *IEEE Trans. Power Electronics*, vol. 62, pp. 2764–2776, May 2015.

[13] F. Z. R. Z. M. Kamel, M. M. Ur Rehman and D. Maksimovic, "Control of independent-input, parallel-output dc/dc converters for modular battery building blocks," *2019 IEEE Applied Power Electronics Conference and Exposition (APEC),*, pp. 234–240, Jan. 2019.

[14] C. T. Rim and G. H. Cho, "Phasor transformation and its application to the dc/ac analyses of frequency phase-controlled series resonant converters (src)," *IEEE Trans. Power Electronics*, vol. 5, pp. 201–211, Apr. 1990.

[15] D. Seltzer, "Modeling and control of the dual active bridge series resonant converter," Ph.D. dissertation, University of Colorado, Boulder, CO, 2014.

[16] M. H. S. Lee, J. Kim and H. Song, "Inrush current estimation for hot swap of the parallel connected large capacity battery pack," *2018 IEEE Energy Conversion Congress and Exposition (ECCE)*, pp. 2489–2492, Jan. 2018.

[17] E. Martinez-Laserna, E. Sarasketa-Zabala, D.-I. Stroe, M. Swierczynski, A. Warnecke, J. Timmermans, S. Goutam, and P. Rodriguez, "Evaluation of lithium-ion battery second life performance and degradation," in *2016 IEEE Energy Conversion Congress and Exposition (ECCE)*, 2016, pp. 1–7.

[18] M. Swierczynski, D.-I. Stroe, and S. K. Kær, "Calendar ageing of lifepo4/c batteries in the second life applications," in *2017 19th European Conference on Power Electronics and Applications (EPE'17 ECCE Europe)*, 2017, pp. P.1–P.8.

[19] E. Hossain, D. Murtaugh, J. Mody, H. M. R. Faruque, M. S. Haque Sunny, and N. Mohammad, "A comprehensive review on second-life batteries: Current state, manufacturing considerations, applications, impacts, barriers potential solutions, business strategies, and policies," *IEEE Access*, vol. 7, pp. 73 215–73 252, 2019.

[20] E. Locorotondo, V. Cultrera, L. Pugi, L. Berzi, M. Pasquali, N. Andrenacci, G. Lutzemberger, and M. Pierini, "Electrical lithium battery performance model for second life applications," in *2020 IEEE International Conference on Environment and Electrical Engineering and 2020 IEEE Industrial and Commercial Power Systems Europe (EEEIC / ICPS Europe)*, 2020, pp. 1–6.

[21] I. Sanz-Gorrachategui, P. Pastor-Flores, M. Pajovic, Y. Wang, P. V. Orlik, C. Bernal-Ruiz, A. Bono-Nuez, and J. S. Artal-Sevil, "Remaining useful life estimation for lfp cells in second-life applications," *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1–10, 2021.

[22] E. Martinez-Laserna, E. Sarasketa-Zabala, I. Villarreal Sarria, D.-I. Stroe, M. Swierczynski, A. Warnecke, J.-M. Timmermans, S. Goutam, N. Omar, and P. Rodriguez, "Technical viability of battery second life: A study from the ageing perspective," *IEEE Transactions on Industry Applications*, vol. 54, no. 3, pp. 2703–2713, 2018.

[23] R. Di Rienzo, M. Zeni, F. Baronti, R. Roncella, and R. Saletti, "Passive balancing algorithm for charge equalization of series connected battery cells," in *2020 2nd IEEE International Conference on Industrial Electronics for Sustainable Energy Systems (IESES)*, vol. 1, 2020, pp. 73–79.

[24] A. Fazeli, M. Stadie, M. Kerner, A. Burger, H. Nagaoka, M. Kramis, J. Ortloff, and F. Jomrich, "A proof of concept for the application of second-life electric vehicle batteries as a stationary energy storage system," in *2021 IEEE Electrical Power and Energy Conference (EPEC)*, 2021, pp. 14–19.

[25] Z. Cano, D. Banham, S. Ye, A. Hintennach, J. Lu, M. Fowler, and Z. Chen, "Batteries and fuel cells for emerging electric vehicle markets," *Nature Energy*, vol. 3, pp. 279–289, 04 2018.

[26] M. M. U. Rehman, "Modular, scalable battery systems with integrated cell balancing and dc bus power processing," 2018.

[27] F. B. R. R. R. S. Roberto Di Rienzo, Marco Zeni, "Passive balancing algorithm for charge equalization of series connected battery cells," *2020 2nd IEEE International Conference on Industrial Electronics for Sustainable Energy Systems (IESES)*, 2020.

[28] M. Rasheed, M. Kamel, H. Wang, R. Zane, and K. Smith, "Investigation of active life balancing to recondition li-ion battery packs for 2nd life," in *2020 IEEE 21st Workshop on Control and Modeling for Power Electronics (COMPEL)*, 2020, pp. 1–7.

[29] D. Costinett, K. Hathaway, M. U. Rehman, M. Evzelman, R. Zane, Y. Levron, and D. Maksimovic, "Active balancing system for electric vehicles with incorporated low voltage bus," in *2014 IEEE Applied Power Electronics Conference and Exposition - APEC 2014*, 2014, pp. 3230–3236.

[30] F. Zhang, M. M. Ur Rehman, H. Wang, Y. Levron, G. Plett, R. Zane, and D. Maksimović, "State-of-charge estimation based on microcontroller-implemented sigma-point kalman filter in a modular cell balancing system for lithium-ion battery packs," in *2015 IEEE 16th Workshop on Control and Modeling for Power Electronics (COMPEL)*, 2015, pp. 1–7.

[31] M. Kamel, "Analysis and control of parallel and series connected modular battery systems," Ph.D. dissertation, Utah State University, 2021.

[32] P. S. S. R. C. N. P.-P. Enver Candan, Derek Heeger, "Hot-swapping analysis and implementation of series-stacked server power delivery architectures," *IEEE Transactions on Power Electronics*, vol. 32, pp. 8071–808 813, Oct. 2017.

[33] "Driving to the future of energy storage: Techno-economic analysis of a novel method to recondition second life electric vehicle batteries," *Applied Energy*, vol. 295, p. 117007, 2021.

APPENDICES

APPENDIX A
MicroController Module Control Code

```
// ###################################################################
// *  main.c
// *  Author: Brooks Maughan, Mohamed Kamel, and Rohail Hassan
// *  Created on: May, 2019
// ###################################################################
#include "global.h"
#include "CONV_ControlLaw_shared_data.h"
#pragma CODE_SECTION(CONV_timeline_isr, ".TI.ramfunc");
extern uint16_t RamfuncsLoadStart, RamfuncsLoadSize, RamfuncsRunStart;
// Place buffers in GSRAM
//#pragma DATA_SECTION(sData, "ramgs0");
//#pragma DATA_SECTION(rData, "ramgs1");

// Define some local functions

interrupt void CONV_timeline_isr(void);
interrupt void cpuTimer1ISR(void);
interrupt void cpuTimer2ISR(void);
interrupt void canaISR(void);
interrupt void canbISR(void);
void ATREX_CAN_Communication(void);
void OneSecondHandle(void);
void OneMilliSecondHandle(void);
void ONeSecondSOCEstimator(void);
void SOCInitialization(void);

uint16_t EnableSense = 0;
int ProtectionCurrent = 0;
float CurrentInputInFloat = 0;

_iq  VuMinusVariable = 0;
_iq  VLVariable = 0;
_iq  VbusVariable = 0;

uint16_t BadCommunicationCounter = 0;
uint16_t SOC_Initialize = 0;
uint16_t Timer0Flag = 0;
uint16_t Timer1Flag = 0;
uint16_t SOC_Estimate = 0;
uint16_t count0Timer = 0;
uint16_t count1Timer = 0;
uint16_t DebugCount = 0;
uint16_t secondCounter = 0;
float CapacitiesHW[4] = {0.00000654378f, 0.00000695214f,
0.0000067421f, 0.0000063496f}; // 1/Q/3600
//
// From Rohail's file
// Defines
//
#define FIFO_LVL     8                // FIFO interrupt level
#define BURST        FIFO_LVL         // Each burst will empty the FIFO
#define TRANSFER     1               // It will take 1 burst of 8 to transfer
```

```
                                        // all data in rData


//
// Globals
//
int16_t sData[8] = {0};                    // Send data buffer
int16_t rData[8] = {0};                    // Receive data buffer
int16_t canCounter = 0;
uint16_t writenow = 0;
int16_t DiffEn = 0;
int16_t RST = 0;
int16_t CTLEn = 0;
int16_t SWEn = 0;
int16_t rampDuty = 0;
int16_t startupbits = 0;
volatile uint16_t done = 0;        // Flag set when data transferred

//Josh's variables for CAN
int16_t stringMessage[8];
uint32_t c = 0;



// Current Command generation variables
uint16_t MaxCurrentIndex = 0;
float MaxCapacity = 0.0f;
float MinCapacity = 100.0f;
float AvgCapacity = 0.0f;
float MaxDeltaSOC_Life = 0.2f; // 20%
float BetaValue = 1.0f;


float IcommonRef = 0.0f;
float MaxCurrentRef       = 0.0f;
float AvgSOC              = 0.0f;
float MaxDeltaSOC         = 0.01f; // initialize to minimum value
                 for this = 0.01 which corresponds to 1% SOC
float DeltaSOC = 0.0f;
float MaxSOC = 0.0f;
float MinSOC = 0.0f;
float MidSOC = 0.0f;
float TempAbsSOCRef = 0.0f;
float TempAbsCurrentRef = 0.0f;


float SOCDiff_Desired[bmsCELL_COUNT]    = {0.0f};
float SOCRef[bmsCELL_COUNT]             = {0.0f};
float ImaxSOC[bmsCELL_COUNT]            = {0.0f};
float IminSOC[bmsCELL_COUNT]            = {0.0f};
float DeltaCurrent[bmsCELL_COUNT]       = {0.0f};
float Iref[bmsCELL_COUNT]               = {0.0f};
float IrefScaled[bmsCELL_COUNT]         = {0.0f};
float Kdelta = 0.5f;
unsigned long timer0Value0 = 0;
unsigned long timer0Value1 = 0;
unsigned long timer1Value0 = 0;
unsigned long timer1Value1 = 0;
float total0Time = 0.0f;
float total1Time = 0.0f;
float randomCurrent = 1.0f;
float Vsec = 0.0f;
float VdcRef0 = 0.0f;
float Gdroop = 0.0f;
//
float IALL = 0.0f;
float Vdcref = 0.0f;
// Function Prototypes
//
void initDMA(void);
void initSPIAMaster(void);
__interrupt void dmaCh5ISR(void);
__interrupt void dmaCh6ISR(void);
void configGPIOs(void);

//#############################################################################
// ------ main function --------------------------------------------------------
// * system initialization
// ----------------------------------------------------------------------------
void main(void)
{
    //=========================================================================
    //--------- Step 1. Initialize System Control ------------------------------
    memcpy((uint16_t *)&RamfuncsRunStart,(uint16_t *)
    &RamfuncsLoadStart, (unsigned long)&RamfuncsLoadSize);
    InitSysCtrl();      // PLL, Flash, WatchDog, enable Peripheral
    Clocks- External oscillator, 100MHz system clock
    // Initialize GPIO
```

```
InitGpio();
//=====================================================================
//———————— Step 2. Initialize GPIO ————————————————————————
OpenLoopGPIOsNewPack();
DELAY_US(500000);   // This Delay is critical with the LTC2955
Pushbutton Control. The Blanking time ignores and seems to
oppose/invert ;any changes to PB or Kill during that half
second.
//===================================================================== //————
// Disable CPU interrupts and clear all CPU interrupt flags:
DINT;
IER = 0x0000;
IFR = 0x0000;
                   // IER and IFR are special registers and
                   are defined by "cregister".
                   // The compiler handles cregisters
                   differently because they require special
                   assembly output.
                   // They are not defined similarly as other
                   registers in the header files.

InitPieCtrl();       // Initialize PIE control registers to
their default state
InitPieVectTable(); // Initialize the PIE vector table with
pointers to the shell Interrupt
// * Re-mapped interrupts to ISR functions found within this
file:
EALLOW;
```

```
PieVectTable.TIMER0_INT = &CONV_timeline_isr;
PieVectTable.TIMER1_INT = &cpuTimer1ISR;
//PieVectTable.TIMER2_INT = &cpuTimer2ISR;
PieVectTable.CANA0_INT = &canaISR;
EDIS;
//==========================================================================
//------------ Step 4.0. Initialize CLA ----------------------------------
//InitializeCLA();
//==========================================================================
//------------ Step 5. Initialize all the Device Peripherals ---
// * Configure CPU-Timer 0 to interrupt at 10kHz (with 100Mhz
                    CPU):
InitializeTimers();
// * Initialize EPWM module:
CONV_InitEPwm();
// * Initialize ADC module:
// Initialize the state structure SOC ESTIMATION
vBmsInit( xCell );
// * Initialize CAN bus:
CONV_InitECana();
//CONV_InitECanb();
// * Initialize global variables:
InitGlobalVariable();



    // Rohail's SPI interrupt
    // Interrupts that are used in this example are re-mapped to
    ISR functions
```

```
// found within this file.
//
Interrupt_register(INT_DMA_CH5, &dmaCh5ISR);
Interrupt_register(INT_DMA_CH6, &dmaCh6ISR);
    //
// Set up DMA for SPI use, initialize the SPI for FIFO mode
//
configGPIOs();
initDMA();
initSPIAMaster();

//==================================================================
//------------ Step 6. Enable interrupts ----------------------
// * Enable specific interrupts in the PIE table:
PieCtrlRegs.PIECTRL.bit.ENPIE = 1;  //*0. Enable the PIE block
//PieCtrlRegs.PIEIER1.bit.INTx1 = 1;  //*1. Enable ADCINT1 in
the PIE: Group 0 interrupt 1
PieCtrlRegs.PIEIER1.bit.INTx7 = 1;  //*1. Enable TINT0 in the
PIE: Group 1 interrupt 7
PieCtrlRegs.PIEIER9.bit.INTx5 = 1;  //*1. Enable CANA0 in the
PIE: Group 9 interrupt 5

// * Enable interrupt:
IER |= M_INT1;  // Enable group 1 interrupts - ADC interrupts -
External interrupt 1 and 2 - and Timer 0
IER |= M_INT13;  // Enable group 13 interrupts - Timer1
//IER |= M_INT14;  // Enable group 14 interrupts - Timer2
IER |= M_INT9;  // Enable group 9 interrupts - CANA0 and CANB0
```

```
interrupt


    // Enable interrupt for SPI
// Enable interrupts required for this example
//
Interrupt_enable(INT_DMA_CH5);
Interrupt_enable(INT_DMA_CH6);

// * Enable global Interrupts and higher priority real-time debug events:
EINT;   // Enable Global interrupt INTM
ERTM;   // Enable Global realtime interrupt DBGM
//==========================================================================
//------------- Step 7. User specified code ----------------------
StartTimer(0);
StartTimer(1);
SOC_Estimate = 0;
DebugCount = 0;

    // From Rohail's file
        uint16_t i;
     for(i = 0; i < 8; i++)
{
    sData[i] = 0;
    rData[i] = 0;
    // rData[0] = i1
    // rData[1] = i2
    // rData[2] = i3
```

```
        // rData[3] = i4
        // rData[4] = v1
        // rData[5] = v2
        // rData[6] = v3
        // rData[7] = v4
    }
        writenow = 0;


    while(1)
    {
//                  // StartUp LEDs
//                // SOC Functions/1 Second interrupt Enabled
        if (Timer0Flag == 1)
        {
            OneMilliSecondHandle();
            // Clear the timer interrupt here!
            timer0Value1 = timer0Value0 - CpuTimer0Regs.TIM.all;
            total0Time = timer0Value1*0.01f; // time in uS
            Timer0Flag = 0;
        }

        if (Timer1Flag == 1)
        {
            OneSecondHandle();
            // Clear the timer interrupt here!
            timer1Value1 = timer1Value0 - CpuTimer1Regs.TIM.all;
            total1Time = timer1Value1*0.01f; // time in uS
            Timer1Flag = 0;
            if (count1Timer > 1){count1Timer = 2;/*SOC_Estimate =
            1;*/DebugCount++;}
        }

    }
}
//###########################################################################
// ———— timeline interrupt function ——————————————————
// * interrupt generated every 1000us
// ————————————————————————————————————————————
interrupt void CONV_timeline_isr(void)
{
    // The flag is cleared in OneSecondHandle function
    timer0Value0 = CpuTimer0Regs.TIM.all;
    Timer0Flag = 1;
    count0Timer++;
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
//      ////debugx.timer0[1] = 5999 - CpuTimer0.RegsAddr->TIM.all;
//      //=====================================================================
//      // Acknowledge this interrupt to receive more interrupts
from group 1
//      PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
//      return;
}

//###########################################################################
// ———— cpuTimer1ISR interrupt function ——————————————————
// * interrupt generated every 1 s
// ————————————————————————————————————————————————————————————————
interrupt void cpuTimer1ISR(void)
{
    timer1Value0 = CpuTimer1Regs.TIM.all;
    Timer1Flag = 1;
    count1Timer++;

// // Whatever!
//     Atrexmsg.txMsgData[0]++;
//     Atrexmsg.txMsgData[3]++;
//     EnableSense++;
//         CAN_sendMessage(CANA_BASE, Atrex_Tx_SOC_SOH,
MSG_DATA_LENGTH, Atrexmsg.txMsgData);// Transmit arbitrary response
debugging
//
//         CAN_sendMessage(CANA_BASE, Atrex_Tx_Vout, MSG_DATA_LENGTH,
Atrexmsg.txMsgData);// Transmit arbitrary response debugging
//
//     CAN_sendMessage(CANA_BASE, Atrex_Tx_Capacity,
MSG_DATA_LENGTH, Atrexmsg.txMsgData);// Transmit arbitrary response
debugging
//
//         CAN_sendMessage(CANA_BASE, Atrex_Tx_Currents,
MSG_DATA_LENGTH, Atrexmsg.txMsgData);// Transmit arbitrary response
debugging
//         CAN_sendMessage(CANA_BASE, Atrex_Tx_Temperature,
MSG_DATA_LENGTH, Atrexmsg.txMsgData);// Transmit arbitrary response
```

```
debugging

//      uint16_t MOduleIndex = 0;
//      for(MOduleIndex=0; MOduleIndex<10; MOduleIndex++)
//      {
//          SOCMessages.txMsgData[0] = 1; // This is SOC information
//          SOCMessages.txMsgData[1] = 0;// This is the module
average
//          SOCMessages.txMsgData[2] =
(uint16_t)Pack_IQ.CellSOC_CAN[3*MOduleIndex]>>8; // This is the
first cell in this Module
//          SOCMessages.txMsgData[3] =
(uint16_t)Pack_IQ.CellSOC_CAN[3*MOduleIndex];// This is the first
cell in this Module
//          SOCMessages.txMsgData[4] =
(uint16_t)Pack_IQ.CellSOC_CAN[3*MOduleIndex + 1]>>8; // This is the
second cell in this Module
//          SOCMessages.txMsgData[5] =
(uint16_t)Pack_IQ.CellSOC_CAN[3*MOduleIndex + 1]; // This is the
second cell in this Module
//          SOCMessages.txMsgData[6] =
(uint16_t)Pack_IQ.CellSOC_CAN[3*MOduleIndex + 2]>>8; // This is the
third cell in this Module
//          SOCMessages.txMsgData[7] =
(uint16_t)Pack_IQ.CellSOC_CAN[3*MOduleIndex + 2]; // This is the
third cell in this Module
//          CAN_sendMessage(CANA_BASE, MOduleIndex+14,
MSG_DATA_LENGTH, SOCMessages.txMsgData);
```

```
//      }

}

//###########################################################################
// ——— cpuTimer2ISR interrupt function ——————————————
// * interrupt generated every 1 s
// ————————————————————————————————————
interrupt void cpuTimer2ISR(void)
{

}

interrupt void canbISR(void)
{

}

interrupt void canaISR(void)
{


    c++;
//      Atrexmsg.status = CAN_getInterruptCause(CANA_BASE);    // Read
the CAN-A interrupt status to find the cause of the interrupt
//
//    CAN_clearInterruptStatus(CANA_BASE, Atrexmsg.status);
//
//    CAN_clearGlobalInterruptStatus(CANA_BASE,
CAN_GLOBAL_INT_CANINT0);   // Clear the global interrupt flag for
the CAN interrupt line
//
//   Interrupt_clearACKGroup(INTERRUPT_ACK_GROUP9);        //
Acknowledge this interrupt located in group 9

    //BMSMEM_handle u = &mem;
     canmsg.status = CAN_getInterruptCause(CANA_BASE);   // Read
     the CAN-A interrupt status to find the cause of the interrupt
     if(canmsg.status >= Module_Mailbox_1 && canmsg.status <=
     Module_Mailbox_10) // This is a message from the module we are
     looking for
     {


             CAN_readMessage(CANA_BASE, canmsg.status,
             canmsg.rxMsgData);
             if(canmsg.rxMsgData[0] == 1)
             {
                  stringMessage[0] = canmsg.rxMsgData[0];
                  stringMessage[1] = canmsg.rxMsgData[1];
                  stringMessage[2] = canmsg.rxMsgData[2];
                  stringMessage[3] = canmsg.rxMsgData[3];
                  stringMessage[4] = canmsg.rxMsgData[4];
                  stringMessage[5] = canmsg.rxMsgData[5];
                  stringMessage[6] = canmsg.rxMsgData[6];
                  stringMessage[7] = canmsg.rxMsgData[7];
```

```
                }

                IALL = _IQ8toF(_IQ8((stringMessage[1]<<8)>>8) +
                (_IQ8(stringMessage[2])>>8));
                Vdcref = _IQ8toF(_IQ8((stringMessage[3]<<8)>>8) +
                (_IQ8(stringMessage[4])>>8));

                startupbits = stringMessage[7];
                SOC_Estimate = startupbits>>5;


//                canmsg.txMsgData[0] = stringMessage[0];
//                canmsg.txMsgData[1] = stringMessage[1];
//                canmsg.txMsgData[2] = stringMessage[2];
//                canmsg.txMsgData[3] = stringMessage[3];
//                canmsg.txMsgData[4] = stringMessage[4];
//                canmsg.txMsgData[5] = stringMessage[5];
//                canmsg.txMsgData[6] = stringMessage[6];
//                canmsg.txMsgData[7] = stringMessage[7];
//                CAN_sendMessage(CANA_BASE, 13, MSG_DATA_LENGTH,
canmsg.txMsgData); // We are sending from mailbox #13
//
//
//                switch(canmsg.rxMsgData[0])
//                 {
//                       case CAN_Vcell:   // Store Vcell into
corresponding Module
////                         xCell[canmsg.status*3-3].fCellVoltage =
ADC_K_Vi*(((canmsg.rxMsgData[2] <<8) + canmsg.rxMsgData[3])*1.0f);
////                         xCell[canmsg.status*3-2].fCellVoltage =
ADC_K_Vi*(((canmsg.rxMsgData[4] <<8) + canmsg.rxMsgData[5])*1.0f);
////                         xCell[canmsg.status*3-1].fCellVoltage =
ADC_K_Vi*(((canmsg.rxMsgData[6] <<8) + canmsg.rxMsgData[7])*1.0f);
//                           xCell[0].fCellVoltage =
ADC_K_Vi*(((canmsg.rxMsgData[2] <<8) + canmsg.rxMsgData[3])*2.0f);
//                           canmsg.Icell[0] = (((canmsg.rxMsgData[4]
<<8) + canmsg.rxMsgData[5]));
//                           CurrentInputInFloat =
ADC_K_Ix*canmsg.Icell[0] ;
//                           xCell[0].fCellCurrent =
(-ADC_K_Ix*(canmsg.Icell[0]) + canmsg.FloatOffset);
//                       break;
//                 }
//               // Now let's make sure we estimate the SOC
//               if (SOC_Estimate == 0)
//               {
//                   vBmsEstInit( xCell, 0 );
//                   SOC_Estimate++;
////                     Pack_IQ.CellSOC_CAN[0] = _IQ15(xCell[ 0
].fCellSoc);
////                     Pack_IQ.CellSOC_CAN_V2[0] = _IQ14(xCell[ 0
].fCellSoc);
////                     SOCMessages.txMsgData[0] = 1; // This is SOC
information
////                     SOCMessages.txMsgData[1] = 0;// This is the
module average
```

```
////                    SOCMessages.txMsgData[2] =
(uint16_t)Pack_IQ.CellSOC_CAN[0]>>8; // This is the first cell in
this Module
////                    SOCMessages.txMsgData[3] =
(uint16_t)Pack_IQ.CellSOC_CAN[0];// This is the first cell in this
Module
////                    SOCMessages.txMsgData[4] = 0; // This is the
second cell in this Module
////                    SOCMessages.txMsgData[5] = 0; // This is the
second cell in this Module
////                    SOCMessages.txMsgData[6] = 0; // This is the
third cell in this Module
////                    SOCMessages.txMsgData[7] = 0; // This is the
third cell in this Module
////                    CAN_sendMessage(CANA_BASE, 14,
MSG_DATA_LENGTH, SOCMessages.txMsgData); // We are sending from
mailbox #14
//                      uint16_t ModuleIndex;
//                      for( ModuleIndex=0; ModuleIndex<3;
ModuleIndex++)
//                          {
//                              Pack_IQ.CellSOC_CAN[ModuleIndex] =
_IQ15(xCell[ ModuleIndex ].fCellSoc);
//                              SOCMessages.txMsgData[2*ModuleIndex] =
0; // This is the ModuleIndex cell in this Module
//                              SOCMessages.txMsgData[2*ModuleIndex+1]
= 0;// This is the ModuleIndex cell in this Module
//                  //
```

```
SOCMessages.txMsgData[2*ModuleIndex+1] = 0;// This is the
ModuleIndex cell in this Module
//                  //
SOCMessages.txMsgData[2*ModuleIndex+8] =
(uint16_t)Pack_IQ.CellVoltage_CAN[ModuleIndex]>>8; // This is the
ModuleIndex cell in this Module
//                  //
SOCMessages.txMsgData[2*ModuleIndex+9] =
(uint16_t)Pack_IQ.CellVoltage_CAN[ModuleIndex];// This is the
ModuleIndex cell in this Module
//                  //
SOCMessages.txMsgData[2*ModuleIndex+16] =
(uint16_t)Pack_IQ.CellCurrent_CAN[ModuleIndex]>>8; // This is the
ModuleIndex cell in this Module
//                  //
SOCMessages.txMsgData[2*ModuleIndex+17] =
(uint16_t)Pack_IQ.CellCurrent_CAN[ModuleIndex];// This is the
ModuleIndex cell in this Module
//                          }
//                          CAN_sendMessage(CANA_BASE, CAN_Mailbox_SOC,
MSG_DATA_LENGTH, SOCMessages.txMsgData); // We are sending from
mailbox #14
//
//                          for( ModuleIndex=0; ModuleIndex<3;
ModuleIndex++)
//                          {
//                                Pack_IQ.CellVoltage_CAN[ModuleIndex] =
_IQ12(xCell[ ModuleIndex ].fCellVoltage);
```

```
//
VoltageMessages.txMsgData[2*ModuleIndex] = 0; // This is the
ModuleIndex cell in this Module
//
VoltageMessages.txMsgData[2*ModuleIndex+1] = 0;// This is the
ModuleIndex cell in this Module
//                          }
//                          CAN_sendMessage(CANA_BASE,
CAN_Mailbox_Voltage, MSG_DATA_LENGTH, VoltageMessages.txMsgData);
// We are sending from mailbox #14
//
//                          for( ModuleIndex=0; ModuleIndex<3;
ModuleIndex++)
//                          {
//                                Pack_IQ.CellCurrent_CAN[ModuleIndex] =
_IQ8(xCell[ ModuleIndex ].fCellCurrent);
//
CurrentMessages.txMsgData[2*ModuleIndex] = 0; // This is the
ModuleIndex cell in this Module
//
CurrentMessages.txMsgData[2*ModuleIndex+1] = 0;// This is the
ModuleIndex cell in this Module
//                          }
//                          CAN_sendMessage(CANA_BASE,
CAN_Mailbox_Current, MSG_DATA_LENGTH, CurrentMessages.txMsgData);
// We are sending from mailbox #14
//                 }
//                 else
//              {
//                  // Step in time
//                  vBmsEstStep( xCell, 0);
//                  // Transmit the estimated SOC
//              }

////       else if(canmsg.status == ComputerDebug_Mailbox_13)      //
Computer Debug
////       {
////            CAN_readMessage(CANA_BASE, ComputerDebug_Mailbox_13,
canmsg.rxMsgData);
////       }
//      else     // Unexpected message received
//      {
//      }

     }
//        Pack_IQ.CellSOC_CAN[0] = _IQ15(xCell[ 0 ].fCellSoc);
//        Pack_IQ.CellSOC_CAN_V2[0] = _IQ14(xCell[ 0 ].fCellSoc);
//        SOCMessages.txMsgData[0] = 1; // This is SOC information
//        SOCMessages.txMsgData[1] = 0;// This is the module average
//        SOCMessages.txMsgData[2] =
(uint16_t)Pack_IQ.CellSOC_CAN[0]>>8; // This is the first cell in
this Module
//        SOCMessages.txMsgData[3] =
(uint16_t)Pack_IQ.CellSOC_CAN[0];// This is the first cell in this
Module
//        SOCMessages.txMsgData[4] = 0; // This is the second cell in
```

```
this Module
//        SOCMessages.txMsgData[5] = 0; // This is the second cell in
this Module
//        SOCMessages.txMsgData[6] = 0; // This is the third cell in
this Module
//        SOCMessages.txMsgData[7] = 0; // This is the third cell in
this Module
//        CAN_sendMessage(CANA_BASE, 14, MSG_DATA_LENGTH,
SOCMessages.txMsgData); // We are sending from mailbox #14

//       // Brooks added this to send SOC, Voltage, and Current to
MATLAB and commented the previous section
//      //According to the Technical Reference Manual, this
uController only allows 8 bits of data to be used in CAN
//              uint16_t ModuleIndex;
//              for( ModuleIndex=0; ModuleIndex<3; ModuleIndex++)
//              {
//                      Pack_IQ.CellSOC_CAN[ModuleIndex] = _IQ15(xCell[
ModuleIndex ].fCellSoc);
//                      SOCMessages.txMsgData[2*ModuleIndex] =
(uint16_t)Pack_IQ.CellSOC_CAN[ModuleIndex]>>8; // This is the
ModuleIndex cell in this Module
//                      SOCMessages.txMsgData[2*ModuleIndex+1] =
(uint16_t)Pack_IQ.CellSOC_CAN[ModuleIndex];// This is the
ModuleIndex cell in this Module
//      //              SOCMessages.txMsgData[2*ModuleIndex+1] = 0;//
This is the ModuleIndex cell in this Module
//      //              SOCMessages.txMsgData[2*ModuleIndex+8] =
(uint16_t)Pack_IQ.CellVoltage_CAN[ModuleIndex]>>8; // This is the
ModuleIndex cell in this Module
//      //              SOCMessages.txMsgData[2*ModuleIndex+9] =
(uint16_t)Pack_IQ.CellVoltage_CAN[ModuleIndex];// This is the
ModuleIndex cell in this Module
//      //              SOCMessages.txMsgData[2*ModuleIndex+16] =
(uint16_t)Pack_IQ.CellCurrent_CAN[ModuleIndex]>>8; // This is the
ModuleIndex cell in this Module
//      //              SOCMessages.txMsgData[2*ModuleIndex+17] =
(uint16_t)Pack_IQ.CellCurrent_CAN[ModuleIndex];// This is the
ModuleIndex cell in this Module
//              }
//              CAN_sendMessage(CANA_BASE, CAN_Mailbox_SOC,
MSG_DATA_LENGTH, SOCMessages.txMsgData); // We are sending from
mailbox #14
//
//              for( ModuleIndex=0; ModuleIndex<3; ModuleIndex++)
//              {
//                      Pack_IQ.CellVoltage_CAN[ModuleIndex] =
_IQ12(xCell[ ModuleIndex ].fCellVoltage);
//                      VoltageMessages.txMsgData[2*ModuleIndex] =
(uint16_t)Pack_IQ.CellVoltage_CAN[ModuleIndex]>>8; // This is the
ModuleIndex cell in this Module
//                      VoltageMessages.txMsgData[2*ModuleIndex+1] =
(uint16_t)Pack_IQ.CellVoltage_CAN[ModuleIndex];// This is the
ModuleIndex cell in this Module
//              }
//              CAN_sendMessage(CANA_BASE, CAN_Mailbox_Voltage,
MSG_DATA_LENGTH, VoltageMessages.txMsgData); // We are sending from
mailbox #14
//
//              for( ModuleIndex=0; ModuleIndex<3; ModuleIndex++)
//              {
//                   Pack_IQ.CellCurrent_CAN[ModuleIndex] =
_IQ8(xCell[ ModuleIndex ].fCellCurrent);
//                   CurrentMessages.txMsgData[2*ModuleIndex] =
(uint16_t)Pack_IQ.CellCurrent_CAN[ModuleIndex]>>8; // This is the
ModuleIndex cell in this Module
//                   CurrentMessages.txMsgData[2*ModuleIndex+1] =
(uint16_t)Pack_IQ.CellCurrent_CAN[ModuleIndex];// This is the
ModuleIndex cell in this Module
//              }
//              CAN_sendMessage(CANA_BASE, CAN_Mailbox_Current,
MSG_DATA_LENGTH, CurrentMessages.txMsgData); // We are sending from
mailbox #14
//
            DebugCount++;
        CAN_clearInterruptStatus(CANA_BASE, canmsg.status);       //
        Clear the interrupt from message object

        CAN_clearGlobalInterruptStatus(CANA_BASE,
        CAN_GLOBAL_INT_CANINT0);    // Clear the global interrupt
        flag for the CAN interrupt line

        Interrupt_clearACKGroup(INTERRUPT_ACK_GROUP9);          //
        Acknowledge this interrupt located in group 9
```

```
}
/***************************************************************
 *  FUNCTION         : __interrupt void canbISR(void)
 *       return      : void
 *       arg         : void
 *  Created by       : Mohamed Kamel
 *  Date created     : 05/28/2018
 *  Description      : This ISR is intende for responding to ATREX,
 it currently only sends an arbitrary response for testing purposes
 *                   :
 *                   :
 *  Notes            :
 ***************************************************************/
//interrupt void canaISR(void)
//{
//     canCounter++;
////     Atrexmsg.status = CAN_getInterruptCause(CANA_BASE);    //
Read the CAN-A interrupt status to find the cause of the interrupt
////
////    CAN_clearInterruptStatus(CANA_BASE, Atrexmsg.status);
////
////    CAN_clearGlobalInterruptStatus(CANA_BASE,
CAN_GLOBAL_INT_CANINT0);    // Clear the global interrupt flag for
the CAN interrupt line
////
////    Interrupt_clearACKGroup(INTERRUPT_ACK_GROUP9);         //
Acknowledge this interrupt located in group 9
//
```

```
//      //BMSMEM_handle u = &mem;
//       canmsg.status = CAN_getInterruptCause(CANA_BASE);    // Read
the CAN-A interrupt status to find the cause of the interrupt
//       if(canmsg.status >= Module_Mailbox_1 && canmsg.status <=
Module_Mailbox_10) // This is a message from the module we are
looking for
//       {
//                 CAN_readMessage(CANA_BASE, canmsg.status,
canmsg.rxMsgData);
//                 switch(canmsg.rxMsgData[0])
//                  {
////                          case CAN_Vcell:    // Store Vcell into
corresponding Module
//////                               xCell[canmsg.status*3-3].fCellVoltage
= ADC_K_Vi*(((canmsg.rxMsgData[2] <<8) +
canmsg.rxMsgData[3])*1.0f);
//////                               xCell[canmsg.status*3-2].fCellVoltage
= ADC_K_Vi*(((canmsg.rxMsgData[4] <<8) +
canmsg.rxMsgData[5])*1.0f);
//////                               xCell[canmsg.status*3-1].fCellVoltage
= ADC_K_Vi*(((canmsg.rxMsgData[6] <<8) +
canmsg.rxMsgData[7])*1.0f);
////                              xCell[0].fCellVoltage =
ADC_K_Vi*(((canmsg.rxMsgData[2] <<8) + canmsg.rxMsgData[3])*2.0f);
////                              canmsg.Icell[0] = (((canmsg.rxMsgData[4]
<<8) + canmsg.rxMsgData[5]));
////                              CurrentInputInFloat =
ADC_K_Ix*canmsg.Icell[0] ;
```

```
////                              xCell[0].fCellCurrent =
(-ADC_K_Ix*(canmsg.Icell[0]) + canmsg.FloatOffset);
////                         break;
//                  }
//                 // Now let's make sure we estimate the SOC
//                 if (SOC_Estimate == 0)
//                 {
//                      vBmsEstInit( xCell, 0 );
////                       SOC_Estimate++;
////                       Pack_IQ.CellSOC_CAN[0] = _IQ15(xCell[ 0
].fCellSoc);
////                       Pack_IQ.CellSOC_CAN_V2[0] = _IQ14(xCell[ 0
].fCellSoc);
////                       SOCMessages.txMsgData[0] = 1; // This is SOC
information
////                       SOCMessages.txMsgData[1] = 0;// This is the
module average
////                       SOCMessages.txMsgData[2] =
(uint16_t)Pack_IQ.CellSOC_CAN[0]>>8; // This is the first cell in
this Module
////                       SOCMessages.txMsgData[3] =
(uint16_t)Pack_IQ.CellSOC_CAN[0];// This is the first cell in this
Module
////                       SOCMessages.txMsgData[4] = 0; // This is the
second cell in this Module
////                       SOCMessages.txMsgData[5] = 0; // This is the
second cell in this Module
////                       SOCMessages.txMsgData[6] = 0; // This is the
```

```
third cell in this Module
////                    SOCMessages.txMsgData[7] = 0; // This is the
third cell in this Module
////                    CAN_sendMessage(CANA_BASE, 14,
MSG_DATA_LENGTH, SOCMessages.txMsgData); // We are sending from
mailbox #14
//                    uint16_t ModuleIndex;
//                    for( ModuleIndex=0; ModuleIndex<4;
ModuleIndex++)
//                        {
//                            Pack_IQ.CellSOC_CAN[ModuleIndex] =
_IQ15(xCell[ ModuleIndex ].fCellSoc);
//                            SOCMessages.txMsgData[2*ModuleIndex] =
0; // This is the ModuleIndex cell in this Module
//                            SOCMessages.txMsgData[2*ModuleIndex+1]
= 0;// This is the ModuleIndex cell in this Module
//                //
SOCMessages.txMsgData[2*ModuleIndex+1] = 0;// This is the
ModuleIndex cell in this Module
//                //
SOCMessages.txMsgData[2*ModuleIndex+8] =
(uint16_t)Pack_IQ.CellVoltage_CAN[ModuleIndex]>>8; // This is the
ModuleIndex cell in this Module
//                //
SOCMessages.txMsgData[2*ModuleIndex+9] =
(uint16_t)Pack_IQ.CellVoltage_CAN[ModuleIndex];// This is the
ModuleIndex cell in this Module
//                //
SOCMessages.txMsgData[2*ModuleIndex+16] =
(uint16_t)Pack_IQ.CellCurrent_CAN[ModuleIndex]>>8; // This is the
ModuleIndex cell in this Module
//                  //
SOCMessages.txMsgData[2*ModuleIndex+17] =
(uint16_t)Pack_IQ.CellCurrent_CAN[ModuleIndex];// This is the
ModuleIndex cell in this Module
//                          }
//                          CAN_sendMessage(CANA_BASE, CAN_Mailbox_SOC,
MSG_DATA_LENGTH, SOCMessages.txMsgData); // We are sending from
mailbox #14
//
//                          for( ModuleIndex=0; ModuleIndex<4;
ModuleIndex++)
//                          {
//                                Pack_IQ.CellVoltage_CAN[ModuleIndex] =
_IQ12(xCell[ ModuleIndex ].fCellVoltage);
//
VoltageMessages.txMsgData[2*ModuleIndex] = 0; // This is the
ModuleIndex cell in this Module
//
VoltageMessages.txMsgData[2*ModuleIndex+1] = 0;// This is the
ModuleIndex cell in this Module
//                          }
//                          CAN_sendMessage(CANA_BASE,
CAN_Mailbox_Voltage, MSG_DATA_LENGTH, VoltageMessages.txMsgData);
// We are sending from mailbox #14
//
```

```
//                          for( ModuleIndex=0; ModuleIndex<4;
ModuleIndex++)
//                          {
//                                  Pack_IQ.CellCurrent_CAN[ModuleIndex] =
_IQ8(xCell[ ModuleIndex ].fCellCurrent);
//
CurrentMessages.txMsgData[2*ModuleIndex] = 0; // This is the
ModuleIndex cell in this Module
//
CurrentMessages.txMsgData[2*ModuleIndex+1] = 0;// This is the
ModuleIndex cell in this Module
//                          }
//                          CAN_sendMessage(CANA_BASE,
CAN_Mailbox_Current, MSG_DATA_LENGTH, CurrentMessages.txMsgData);
// We are sending from mailbox #14
//              }
//              else
//              {
//                  // Step in time
//                  vBmsEstStep( xCell, 0);
//                  // Transmit the estimated SOC
//              }
//      }
////      else if(canmsg.status == ComputerDebug_Mailbox_13)     //
Computer Debug
////      {
////          CAN_readMessage(CANA_BASE, ComputerDebug_Mailbox_13,
canmsg.rxMsgData);
```

```
////      }
//      else    // Unexpected message received
//      {
//      }
////        Pack_IQ.CellSOC_CAN[0] = _IQ15(xCell[ 0 ].fCellSoc);
////        Pack_IQ.CellSOC_CAN_V2[0] = _IQ14(xCell[ 0 ].fCellSoc);
////        SOCMessages.txMsgData[0] = 1; // This is SOC information
////        SOCMessages.txMsgData[1] = 0;// This is the module
average
////        SOCMessages.txMsgData[2] =
(uint16_t)Pack_IQ.CellSOC_CAN[0]>>8; // This is the first cell in
this Module
////        SOCMessages.txMsgData[3] =
(uint16_t)Pack_IQ.CellSOC_CAN[0];// This is the first cell in this
Module
////        SOCMessages.txMsgData[4] = 0; // This is the second cell
in this Module
////        SOCMessages.txMsgData[5] = 0; // This is the second cell
in this Module
////        SOCMessages.txMsgData[6] = 0; // This is the third cell
in this Module
////        SOCMessages.txMsgData[7] = 0; // This is the third cell
in this Module
////        CAN_sendMessage(CANA_BASE, 14, MSG_DATA_LENGTH,
SOCMessages.txMsgData); // We are sending from mailbox #14
//
//       // Brooks added this to send SOC, Voltage, and Current to
MATLAB and commented the previous section
```

```
//      //According to the Technical Reference Manual, this
uController only allows 8 bits of data to be used in CAN
//              uint16_t ModuleIndex;
//              for( ModuleIndex=0; ModuleIndex<4; ModuleIndex++)
//              {
//                      Pack_IQ.CellSOC_CAN[ModuleIndex] = _IQ15(xCell[
ModuleIndex ].fCellSoc);
//                      SOCMessages.txMsgData[2*ModuleIndex] =
(uint16_t)Pack_IQ.CellSOC_CAN[ModuleIndex]>>8; // This is the
ModuleIndex cell in this Module
//                      SOCMessages.txMsgData[2*ModuleIndex+1] =
(uint16_t)Pack_IQ.CellSOC_CAN[ModuleIndex];// This is the
ModuleIndex cell in this Module
//      //              SOCMessages.txMsgData[2*ModuleIndex+1] = 0;//
This is the ModuleIndex cell in this Module
//      //              SOCMessages.txMsgData[2*ModuleIndex+8] =
(uint16_t)Pack_IQ.CellVoltage_CAN[ModuleIndex]>>8; // This is the
ModuleIndex cell in this Module
//      //              SOCMessages.txMsgData[2*ModuleIndex+9] =
(uint16_t)Pack_IQ.CellVoltage_CAN[ModuleIndex];// This is the
ModuleIndex cell in this Module
//      //              SOCMessages.txMsgData[2*ModuleIndex+16] =
(uint16_t)Pack_IQ.CellCurrent_CAN[ModuleIndex]>>8; // This is the
ModuleIndex cell in this Module
//      //              SOCMessages.txMsgData[2*ModuleIndex+17] =
(uint16_t)Pack_IQ.CellCurrent_CAN[ModuleIndex];// This is the
ModuleIndex cell in this Module
//              }
//              CAN_sendMessage(CANA_BASE, CAN_Mailbox_SOC,
MSG_DATA_LENGTH, SOCMessages.txMsgData); // We are sending from
mailbox #14
//
//              for( ModuleIndex=0; ModuleIndex<4; ModuleIndex++)
//              {
//                      Pack_IQ.CellVoltage_CAN[ModuleIndex] =
_IQ12(xCell[ ModuleIndex ].fCellVoltage);
//                      VoltageMessages.txMsgData[2*ModuleIndex] =
(uint16_t)Pack_IQ.CellVoltage_CAN[ModuleIndex]>>8; // This is the
ModuleIndex cell in this Module
//                      VoltageMessages.txMsgData[2*ModuleIndex+1] =
(uint16_t)Pack_IQ.CellVoltage_CAN[ModuleIndex];// This is the
ModuleIndex cell in this Module
//              }
//              CAN_sendMessage(CANA_BASE, CAN_Mailbox_Voltage,
MSG_DATA_LENGTH, VoltageMessages.txMsgData); // We are sending from
mailbox #14
//
//              for( ModuleIndex=0; ModuleIndex<4; ModuleIndex++)
//              {
//                      Pack_IQ.CellCurrent_CAN[ModuleIndex] =
_IQ8(xCell[ ModuleIndex ].fCellCurrent);
//                      CurrentMessages.txMsgData[2*ModuleIndex] =
(uint16_t)Pack_IQ.CellCurrent_CAN[ModuleIndex]>>8; // This is the
ModuleIndex cell in this Module
//                      CurrentMessages.txMsgData[2*ModuleIndex+1] =
(uint16_t)Pack_IQ.CellCurrent_CAN[ModuleIndex];// This is the
ModuleIndex cell in this Module
//              }
//              CAN_sendMessage(CANA_BASE, CAN_Mailbox_Current,
MSG_DATA_LENGTH, CurrentMessages.txMsgData); // We are sending from
mailbox #14
//
////                DebugCount++;
//          CAN_clearInterruptStatus(CANA_BASE, canmsg.status);
// Clear the interrupt from message object
//
//          CAN_clearGlobalInterruptStatus(CANA_BASE,
CAN_GLOBAL_INT_CANINT0);   // Clear the global interrupt flag for
the CAN interrupt line
//
//          Interrupt_clearACKGroup(INTERRUPT_ACK_GROUP9);         //
Acknowledge this interrupt located in group 9
//}

/*************************************************************
*   FUNCTION          : void ChangeDroopValuesOverCAN(void)
*       return        : void
*       arg           : void
*   Created by        : Mohamed Ahmed
*   Date created      : 07/14/2018
*   Description       : Detect Series or parallel connection
*   Notes             :
*************************************************************/
//void ChangeDroopValuesOverCAN(uint16_t DroopValue)
```

```
//{
//    if (ChangeDroopOnce)
//    {
//
//         TransmitToModules[0] = 3;
//         TransmitToModules[1] = DroopValue;
//         TransmitToModules[2] = 1; // Set to 0 to ignore the
calibrated offset!
//         TransmitToModules[3] = 0;
//         TransmitToModules[4] = 0;
//         TransmitToModules[5] = 0;
//         TransmitToModules[6] = 0;
//         TransmitToModules[7] = 0;
//
//         CAN_sendMessage(CANB_BASE, 15, MSG_DATA_LENGTH,
TransmitToModules);    // Send Delta Vref and Irefs to the
appropriate Module
//         CAN_sendMessage(CANB_BASE, 16, MSG_DATA_LENGTH,
TransmitToModules);    // Send Delta Vref and Irefs to the
appropriate Module
//         CAN_sendMessage(CANB_BASE, 17, MSG_DATA_LENGTH,
TransmitToModules);    // Send Delta Vref and Irefs to the
appropriate Module
//         CAN_sendMessage(CANB_BASE, 18, MSG_DATA_LENGTH,
TransmitToModules);    // Send Delta Vref and Irefs to the
appropriate Module
//         CAN_sendMessage(CANB_BASE, 19, MSG_DATA_LENGTH,
TransmitToModules);    // Send Delta Vref and Irefs to the
appropriate Module
//          CAN_sendMessage(CANB_BASE, 20, MSG_DATA_LENGTH,
TransmitToModules);    // Send Delta Vref and Irefs to the
appropriate Module
//          CAN_sendMessage(CANB_BASE, 21, MSG_DATA_LENGTH,
TransmitToModules);    // Send Delta Vref and Irefs to the
appropriate Module
//          CAN_sendMessage(CANB_BASE, 22, MSG_DATA_LENGTH,
TransmitToModules);    // Send Delta Vref and Irefs to the
appropriate Module
//          CAN_sendMessage(CANB_BASE, 23, MSG_DATA_LENGTH,
TransmitToModules);    // Send Delta Vref and Irefs to the
appropriate Module
//          CAN_sendMessage(CANB_BASE, 24, MSG_DATA_LENGTH,
TransmitToModules);    // Send Delta Vref and Irefs to the
appropriate Module
//          ChangeDroopOnce = 0;
//    }
//
//}

/*************************************************************
 *  FUNCTION          : ONeSecondSOCEstimator(void)
 *        return      : void
 *        arg         : void
 *  Created by        : MOhamed
 *  Date created      : 10/23/2018
 *  Description       : SOC Estimation every one second
```

```
  *  Notes            :
  ***************************************************************/
void ONeSecondSOCEstimator(void)
{
//    uint16_t CellIndex = 0;
//    uint16_t ModuleIndex = 0;
//
//    for(ModuleIndex = 0; ModuleIndex< 10; ModuleIndex++)
//    {
//        for(CellIndex = 3*ModuleIndex; CellIndex< (3*ModuleIndex
+ 3); CellIndex++)
//            {
//                vBmsEstStep( xCell, CellIndex);
//            }
//    }
 //   vBmsEstStep( xCell, CellIndex);
}
/****************************************************************
  *  FUNCTION         : SOCInitialization(void)
  *      return       : void
  *      arg          : void
  *  Created by       : MOhamed
  *  Date created     : 10/23/2018
  *  Description      : SOC Initialization Function
  *  Notes            :
  ***************************************************************/
void SOCInitialization(void)
{
    // Get initial SOC's
//    uint16_t i = 0;
//    uint16_t j = 0;
//    for(j = 0; j< 10; j++)  // Make sure that modules have sent
data to ensure that useful data is put into initial SOC estimation
//    {
//        for(i = 3*j; i< (3*j+3); i++)
//        {
//            vBmsEstInit( xCell, i );    // Initial Calculation of
SOC for each cell i
//        }
//    }
 //   vBmsEstInit( xCell, i );

}


/***********************************************************
 *  FUNCTION         : OneSecondTemperatureCalculator(void)
 *      return       : void
 *      arg          : void
 *  Created by       : MOhamed
 *  Date created     : 10/23/2018
 *  Description      : Temperature Interpolation every one second
 *  Notes            :
 **********************************************************/
void OneSecondTemperatureCalculator(void)
{
```

```
//    uint16_t CellIndex = 0;
//    uint16_t ModuleIndex = 0;
//
//    for(ModuleIndex = 0; ModuleIndex< 10; ModuleIndex++)
//    {
//        PowerBoardThermistor(&boardTemp, ModuleIndex);
//        // Update the temperature value for each cell in the
Handle
//        for(CellIndex = 3*ModuleIndex; CellIndex<
(3*ModuleIndex+3); CellIndex++)
//        {
//            xCell[CellIndex].fCellTemperature =
boardTemp.Result[ModuleIndex];;    // Initial Calculation of SOC
for each cell i
//        }
//    }
}
/*************************************************************
 * FUNCTION        : OneMilliSecondHandle(void)
 *     return      : void
 *     arg         : void
 * Created by      : Rohail
 * Date created    : 03/15/2021
 * Description     : One MilliSecond Function Call
 * Notes           :
 *************************************************************/
void OneMilliSecondHandle(void)
{
    // Every one millisecond, start communication channels to send
    current references and control commands to the FPGA and receive
    currents and voltages from the FPGA
    DMA_startChannel(DMA_CH6_BASE);
    DMA_startChannel(DMA_CH5_BASE);

    // Now let's unpack the data and feed them into the SOC
    estimation array
    int16_t CellIndex = rData[0];
    Vsec = ADC_K_Vsec*((rData[1])*1.0f);
    xCell[CellIndex].fCellCurrent = -ADC_K_Ix*((rData[2])*1.0f); //
    The SOC estimation code assumes that the current drawn from the
    cell is negative for coulumb counting.
    xCell[CellIndex].fCellVoltage = ADC_K_Vi*((rData[3])*1.0f);

    // This is the droop equation
    IcommonRef = -(IALL + Gdroop*(VdcRef0 - Vsec));



//     uint16_t CellIndex = 0;
//     for(CellIndex = 0; CellIndex< (bmsCELL_COUNT); CellIndex++)
// double check this bmsCELL_COUNT variable if you are going to
increase the number of cells to anything beyond rData structure
//     {
//         xCell[CellIndex].fCellCurrent =
-ADC_K_Ix*((rData[CellIndex])*1.0f); // The SOC estimation code
assumes that the current drawn from the cell is negative for
coulumb counting.
//
////          xCell[CellIndex].fCellVoltage = (SOC_Estimate == 0) ?
ADC_K_Vi*((rData[CellIndex + 4])*1.0f) : ADC_K_Vi*((rData[CellIndex
+ 4])*1.0f) - ((xCell[CellIndex].fCellCurrent)*ADC_K_Rcable); //
accounts for cable resistance
//          xCell[CellIndex].fCellVoltage =
ADC_K_Vi*((rData[CellIndex + 4])*1.0f);
//
//     }

//===================== The next part is for Objective map
initialization ========================================

          MaxCapacity          = 0.0f;
          MinCapacity          = 100.0f;
          MaxCurrentIndex      = 0;
          MaxCurrentRef        = 0.0f;
          AvgCapacity          = 0.0f;
          AvgSOC               = 0.0f;
          MaxDeltaSOC          = 0.005f; // initialize MaxDeltaSOC to
          1% so that it is never less than 1%

          // initialize capacities of cells for coulomb counting and
          objective map generation
          xCell[0].fCellQ = 42.4491f;
          xCell[1].fCellQ = 39.9557f;
          xCell[2].fCellQ = 41.2005f;
```

```
xCell[3].fCellQ = 43.7473f;

// find maximum, minimum and average capacities and SOCs
for(CellIndex = 0; CellIndex < (bmsCELL_COUNT);
CellIndex++) // double check this bmsCELL_COUNT variable if
you are going to increase the number of cells to anything
beyond rData structure
{
    if (xCell[CellIndex].fCellQ > MaxCapacity) MaxCapacity
    = xCell[CellIndex].fCellQ;

    if (xCell[CellIndex].fCellQ < MinCapacity) MinCapacity
    = xCell[CellIndex].fCellQ;

    AvgCapacity = AvgCapacity + xCell[CellIndex].fCellQ;
    AvgSOC = AvgSOC + xCell[CellIndex].fCellSoc;
}
AvgCapacity = AvgCapacity*one_over_bmsCELL_COUNT;
AvgSOC = AvgSOC*one_over_bmsCELL_COUNT;

if (MaxCapacity > MinCapacity) // compute SOCDiff_Desired
only if capacities are different, else skip and keep
SOCDiff_Desired = 0
{
    DeltaSOC = MaxDeltaSOC_Life*BetaValue*((MaxCapacity -
    AvgCapacity) / (MaxCapacity - MinCapacity));
    MaxSOC = 0.85f - DeltaSOC;
    MinSOC = 0.15f + DeltaSOC;
```

```
        MidSOC = (MaxSOC + MinSOC)/2.0f;

        for (CellIndex = 0; CellIndex < (bmsCELL_COUNT);
        CellIndex++)
            SOCDiff_Desired[CellIndex] =
            2.0f*BetaValue*MaxDeltaSOC_Life*((xCell[CellIndex].
            fCellQ - AvgCapacity)/(MaxCapacity -
            MinCapacity))*((AvgSOC - MidSOC)/(MaxSOC -
            MinSOC));
    }
    else
    {
        for (CellIndex = 0; CellIndex < (bmsCELL_COUNT);
        CellIndex++)
            SOCDiff_Desired[CellIndex] = 0.0f;
    }


//========================== if SOC_Estimate == 0 we do nothing, else
we calculate current references====================================

    if (SOC_Estimate == 0)
     {
     }
     else // calculate current references based on the
     objective map
     {
```

```
for (CellIndex = 0; CellIndex < (bmsCELL_COUNT);
CellIndex++)
{
    if (IcommonRef < 0.0f) SOCRef[CellIndex] =
    xCell[CellIndex].fCellSoc − (AvgSOC +
    SOCDiff_Desired[CellIndex]);
    else                   SOCRef[CellIndex] =
    −(xCell[CellIndex].fCellSoc − (AvgSOC +
    SOCDiff_Desired[CellIndex]));

    TempAbsSOCRef = fabsf(SOCRef[CellIndex]); // get
    the absolute value of SOCRef
    if (TempAbsSOCRef > MaxDeltaSOC) MaxDeltaSOC =
    TempAbsSOCRef; // find the maximum absolute value
    of SOCRef, initialize MaxDeltaSOC to 1 so that it
    is never less than 1
}

for (CellIndex = 0; CellIndex < (bmsCELL_COUNT);
CellIndex++) // compute SOC based current limits
{
    ImaxSOC[CellIndex] = fminf( 45.0f + 900.0f*(0.8f −
    xCell[CellIndex].fCellSoc),45.0f); // SOC based
    maximum current limit
    ImaxSOC[CellIndex] = fmaxf(  0.0f,
    ImaxSOC[CellIndex]);

    IminSOC[CellIndex] = fminf(−45.0f + 900.0f*(0.2f −
    xCell[CellIndex].fCellSoc),00.0f);// SOC based
    minimum current limit
    IminSOC[CellIndex] = fmaxf(−45.0f,
    IminSOC[CellIndex]);
}

for (CellIndex = 0; CellIndex < (bmsCELL_COUNT);
CellIndex++) // compute current references to bring
SOCRef to 0
{
    DeltaCurrent[CellIndex] = SOCRef[CellIndex] /
    MaxDeltaSOC;
    if  ((IcommonRef < 3.0f) && (IcommonRef > −3.0f))
    Iref[CellIndex] = IcommonRef; // No differential
    current below common current of 3A
    else Iref[CellIndex] = IcommonRef*(1.0f +
    (Kdelta*DeltaCurrent[CellIndex])); // multiplied
    by 0.5 to limit current reference variation around
    Icommon.. with 0.5, Imax = 3*Imin

    if      (Iref[CellIndex] > ImaxSOC[CellIndex])
    Iref[CellIndex] = ImaxSOC[CellIndex]; // apply SOC
    based current limits
    else if (Iref[CellIndex] < IminSOC[CellIndex])
    Iref[CellIndex] = IminSOC[CellIndex];

    TempAbsCurrentRef = fabsf(Iref[CellIndex]); // get
    the absolute value of SOCRef
```

```
                if (TempAbsCurrentRef > MaxCurrentRef) // extract
                Imax index for best cell selection
                {
                    MaxCurrentRef = TempAbsCurrentRef;
                    MaxCurrentIndex = CellIndex;
                }
            }
          // Transmit the estimated SOC
        }

// Now that we have current references and best cell, lets compute
values to send to the FPGA
// sData[4] is the CONTROLuC register in FPGA, with each bit
corresponding to something
// sData[4][11:0] = [bestCell[3]  bestCell[2]  bestCell[1]
bestCell[0]   x   x   SOC_Est_En  rampDuty  DiffEn  RST  CTLEn
SWEn];

    for (CellIndex = 0; CellIndex < (bmsCELL_COUNT); CellIndex++)
    // compute current references to bring SOCRef to 0
    {
        sData[CellIndex] =
        (int16_t)_IQ10int(_IQ10((-Iref[CellIndex])*IPri_Gain));
    }

//      if      (MaxCurrentIndex==0) sData[4] =  256 + (rampDuty<<4)
+ (DiffEn<<3) + (RST<<2) + (CTLEn<<1) + SWEn;
//      else if (MaxCurrentIndex==1) sData[4] =  512 + (rampDuty<<4)
```

```
+ (DiffEn<<3) + (RST<<2) + (CTLEn<<1) + SWEn;
//     else if (MaxCurrentIndex==2) sData[4] = 1024 + (rampDuty<<4)
+ (DiffEn<<3) + (RST<<2) + (CTLEn<<1) + SWEn;
//     else if (MaxCurrentIndex==3) sData[4] = 2048 + (rampDuty<<4)
+ (DiffEn<<3) + (RST<<2) + (CTLEn<<1) + SWEn;
     if      (MaxCurrentIndex==0) sData[4] =  256 + startupbits;
     else if (MaxCurrentIndex==1) sData[4] =  512 + startupbits;
     else if (MaxCurrentIndex==2) sData[4] = 1024 + startupbits;
     else if (MaxCurrentIndex==3) sData[4] = 2048 + startupbits;

}



/****************************************************************
  * FUNCTION          : OneSecondHandle(void)
  *     return        : void
  *     arg           : void
  * Created by        : MOhamed
  * Date created      : 10/23/2018
  * Description       : One Second Function Call
  * Notes             :
  ***************************************************************/
void OneSecondHandle(void)
{
          secondCounter++;
          // If SOC Estimation is enabled let's run all of that.
          However, we assume that we will enable regulation after we
          estimate the SOC from the open circuit voltage.
```

```
    // So, when "SOC_Estimate = 0", we will just look at the
    V-SOC curve then we will update the flag so that we never
    implement this function
    uint16_t CellIndex = 0;
    if(secondCounter % 4 == 0)
    {
        if (SOC_Estimate == 0)
         {
              for(CellIndex = 0; CellIndex< (bmsCELL_COUNT);
              CellIndex++) // double check this bmsCELL_COUNT
              variable if you are going to increase the number
              of cells to anything beyond rData structure
              {
                  vBmsEstInit( xCell, CellIndex );

              }

//                  SOC_Estimate++;
//                      Pack_IQ.CellSOC_CAN[0] = _IQ15(xCell[ 0
].fCellSoc);
//                      Pack_IQ.CellSOC_CAN_V2[0] = _IQ14(xCell[ 0
].fCellSoc);
//                      SOCMessages.txMsgData[0] = 1; // This is SOC
information
//                      SOCMessages.txMsgData[1] = 0;// This is the
module average
//                      SOCMessages.txMsgData[2] =
(uint16_t)Pack_IQ.CellSOC_CAN[0]>>8; // This is the first cell
```

```
in this Module

//                    SOCMessages.txMsgData[3] =
(uint16_t)Pack_IQ.CellSOC_CAN[0];// This is the first cell in
this Module
//                    SOCMessages.txMsgData[4] = 0; // This is the
second cell in this Module
//                    SOCMessages.txMsgData[5] = 0; // This is the
second cell in this Module
//                    SOCMessages.txMsgData[6] = 0; // This is the
third cell in this Module
//                    SOCMessages.txMsgData[7] = 0; // This is the
third cell in this Module
//                    CAN_sendMessage(CANA_BASE, 14,
MSG_DATA_LENGTH, SOCMessages.txMsgData); // We are sending from
mailbox #14
          }
          else
          {
               // Step in time, estimate SOC using coulomb
               counting



               for(CellIndex = 0; CellIndex< (bmsCELL_COUNT);
               CellIndex++) // double check this bmsCELL_COUNT
               variable if you are going to increase the number
               of cells to anything beyond rData structure
               {
```

```
//                      vBmsEstStep( xCell, CellIndex );
                        xCell[CellIndex].fCellCurrent = randomCurrent;
                        xCell[CellIndex].fCellSoc =
                        xCell[CellIndex].fCellSoc +
                        xCell[CellIndex].fCellCurrent*CapacitiesHW[Cel
                        lIndex]*1.0f; // 0.001 for 1 kHz sampling rate
                    }
                }

//      if (SOC_Estimate == 0)
//      {
//  //          SOCInitialization();
//          //AllowRetrieve = 1;
//          //CanQ0.InitializeI2CData = 2; // Here retreive all
the data from I2C EEPROM
//      }
//      else
//      {
//          //if (AllowRetrieve == 2)
//          //{
//              // Estimate SOC
//  //              ONeSecondSOCEstimator();
//          //}
//      }


    // Calculate Deltas and SOCs and then transmit them over
    CAN
    //BMSAveraging();



//      uint16_t MOduleIndex = 0;
//       for(MOduleIndex=0; MOduleIndex<10; MOduleIndex++)
//       {
//           SOCMessages.txMsgData[0] = 1; // This is SOC
information
//           SOCMessages.txMsgData[1] = 0;// This is the module
average
//           SOCMessages.txMsgData[2] =
(uint16_t)Pack_IQ.CellSOC_CAN[3*MOduleIndex]>>8; // This is the
first cell in this Module
//           SOCMessages.txMsgData[3] =
(uint16_t)Pack_IQ.CellSOC_CAN[3*MOduleIndex];// This is the
first cell in this Module
//           SOCMessages.txMsgData[4] =
(uint16_t)Pack_IQ.CellSOC_CAN[3*MOduleIndex + 1]>>8; // This is
the second cell in this Module
//           SOCMessages.txMsgData[5] =
(uint16_t)Pack_IQ.CellSOC_CAN[3*MOduleIndex + 1]; // This is
the second cell in this Module
//           SOCMessages.txMsgData[6] =
(uint16_t)Pack_IQ.CellSOC_CAN[3*MOduleIndex + 2]>>8; // This is
the third cell in this Module
//           SOCMessages.txMsgData[7] =
(uint16_t)Pack_IQ.CellSOC_CAN[3*MOduleIndex + 2]; // This is
```

```
the third cell in this Module
//          CAN_sendMessage(CANA_BASE, MOduleIndex+14,
MSG_DATA_LENGTH, SOCMessages.txMsgData);
//      }

        // Transmit values to MATLAB over CAN
        uint16_t ModuleIndex;

        for( ModuleIndex=0; ModuleIndex<4; ModuleIndex++)
        {
            Pack_IQ.CellSOC_CAN[ModuleIndex] = _IQ15(xCell[
            ModuleIndex ].fCellSoc);
            SOCMessages.txMsgData[2*ModuleIndex] =
            (uint16_t)Pack_IQ.CellSOC_CAN[ModuleIndex]>>8; //
            This is the ModuleIndex cell in this Module
            SOCMessages.txMsgData[2*ModuleIndex+1] =
            (uint16_t)Pack_IQ.CellSOC_CAN[ModuleIndex];// This
            is the ModuleIndex cell in this Module
        }
        CAN_sendMessage(CANA_BASE, CAN_Mailbox_SOC,
        MSG_DATA_LENGTH, SOCMessages.txMsgData); // We are
        sending from mailbox #14

        for( ModuleIndex=0; ModuleIndex<4; ModuleIndex++)
        {
            Pack_IQ.CellVoltage_CAN[ModuleIndex] =
            _IQ12(xCell[ ModuleIndex ].fCellVoltage);
            VoltageMessages.txMsgData[2*ModuleIndex] =
                (uint16_t)Pack_IQ.CellVoltage_CAN[ModuleIndex]>>8;
                // This is the ModuleIndex cell in this Module
                VoltageMessages.txMsgData[2*ModuleIndex+1] =
                (uint16_t)Pack_IQ.CellVoltage_CAN[ModuleIndex];//
                This is the ModuleIndex cell in this Module
            }
            CAN_sendMessage(CANA_BASE, CAN_Mailbox_Voltage,
            MSG_DATA_LENGTH, VoltageMessages.txMsgData); // We are
            sending from mailbox #14

            for( ModuleIndex=0; ModuleIndex<4; ModuleIndex++)
            {
                Pack_IQ.CellCurrent_CAN[ModuleIndex] = _IQ8(xCell[
                ModuleIndex ].fCellCurrent);
                CurrentMessages.txMsgData[2*ModuleIndex] =
                (uint16_t)Pack_IQ.CellCurrent_CAN[ModuleIndex]>>8;
                // This is the ModuleIndex cell in this Module
                CurrentMessages.txMsgData[2*ModuleIndex+1] =
                (uint16_t)Pack_IQ.CellCurrent_CAN[ModuleIndex];//
                This is the ModuleIndex cell in this Module
            }
            CAN_sendMessage(CANA_BASE, CAN_Mailbox_Current,
            MSG_DATA_LENGTH, CurrentMessages.txMsgData); // We are
            sending from mailbox #14
        }
    // Transmit values to String Controller over CAN
    // send data for cell 1
    if(secondCounter % 4 == 1)
```

```
{
    canmsg_cell0.txMsgData[0] = 0;                  //module 0
    canmsg_cell0.txMsgData[1] =
    (uint16_t)Pack_IQ.CellVoltage_CAN[0]>>8; //Voltage byte 1
    canmsg_cell0.txMsgData[2] =
    (uint16_t)Pack_IQ.CellVoltage_CAN[0];     //Voltage byte 2
    canmsg_cell0.txMsgData[3] =
    (uint16_t)Pack_IQ.CellCurrent_CAN[0]>>8; //Current byte 1
    canmsg_cell0.txMsgData[4] =
    (uint16_t)Pack_IQ.CellCurrent_CAN[0];     //Current byte 2
    canmsg_cell0.txMsgData[5] =
    (uint16_t)Pack_IQ.CellSOC_CAN[0]>>8;      //Soc byte 1
    canmsg_cell0.txMsgData[6] =
    (uint16_t)Pack_IQ.CellSOC_CAN[0];         //Soc byte 2
    canmsg_cell0.txMsgData[7] = 0;
    //cell number
    CAN_sendMessage(CANA_BASE, 13, MSG_DATA_LENGTH,
    canmsg_cell0.txMsgData); // We are sending from mailbox #13
}

// send data for cell 2
if(secondCounter % 4 == 2)
{
    canmsg_cell1.txMsgData[0] = 0;
    //module 0
    canmsg_cell1.txMsgData[1] =
    (uint16_t)Pack_IQ.CellVoltage_CAN[1]>>8; //Voltage byte 1
    canmsg_cell1.txMsgData[2] =
    (uint16_t)Pack_IQ.CellVoltage_CAN[1];     //Voltage byte 2

    canmsg_cell1.txMsgData[3] =
    (uint16_t)Pack_IQ.CellCurrent_CAN[1]>>8; //Current byte 1
    canmsg_cell1.txMsgData[4] =
    (uint16_t)Pack_IQ.CellCurrent_CAN[1];     //Current byte 2
    canmsg_cell1.txMsgData[5] =
    (uint16_t)Pack_IQ.CellSOC_CAN[1]>>8;      //Soc byte 1
    canmsg_cell1.txMsgData[6] =
    (uint16_t)Pack_IQ.CellSOC_CAN[1];         //Soc byte 2
    canmsg_cell1.txMsgData[7] = 1;
    //cell number
    CAN_sendMessage(CANA_BASE, 13, MSG_DATA_LENGTH,
    canmsg_cell1.txMsgData); // We are sending from mailbox #13
}

// send data for cell 3
if(secondCounter % 4 == 3)
{
    canmsg_cell2.txMsgData[0] = 0;              //module 0
    canmsg_cell2.txMsgData[1] =
    (uint16_t)Pack_IQ.CellVoltage_CAN[2]>>8; //Voltage byte 1
    canmsg_cell2.txMsgData[2] =
    (uint16_t)Pack_IQ.CellVoltage_CAN[2];     //Voltage byte 2
    canmsg_cell2.txMsgData[3] =
    (uint16_t)Pack_IQ.CellCurrent_CAN[2]>>8; //Current byte 1
    canmsg_cell2.txMsgData[4] =
    (uint16_t)Pack_IQ.CellCurrent_CAN[2];     //Current byte 2
    canmsg_cell2.txMsgData[5] =
    (uint16_t)Pack_IQ.CellSOC_CAN[2]>>8;        //Soc byte 1
    canmsg_cell2.txMsgData[6] =
    (uint16_t)Pack_IQ.CellSOC_CAN[2];           //Soc byte 2
    canmsg_cell2.txMsgData[7] = 2;
    //cell number
    CAN_sendMessage(CANA_BASE, 13, MSG_DATA_LENGTH,
    canmsg_cell2.txMsgData); // We are sending from mailbox #13
}

// send data for cell 4
if(secondCounter % 4 == 0)
{
    canmsg_cell3.txMsgData[0] = 0;
    //module 0
    canmsg_cell3.txMsgData[1] =
    (uint16_t)Pack_IQ.CellVoltage_CAN[3]>>8; //Voltage byte 1
    canmsg_cell3.txMsgData[2] =
    (uint16_t)Pack_IQ.CellVoltage_CAN[3];     //Voltage byte 2
    canmsg_cell3.txMsgData[3] =
    (uint16_t)Pack_IQ.CellCurrent_CAN[3]>>8; //Current byte 1
    canmsg_cell3.txMsgData[4] =
    (uint16_t)Pack_IQ.CellCurrent_CAN[3];     //Current byte 2
    canmsg_cell3.txMsgData[5] =
    (uint16_t)Pack_IQ.CellSOC_CAN[3]>>8;        //Soc byte 1
    canmsg_cell3.txMsgData[6] =
    (uint16_t)Pack_IQ.CellSOC_CAN[3];           //Soc byte 2
    canmsg_cell3.txMsgData[7] = 3;
    //cell number
    CAN_sendMessage(CANA_BASE, 13, MSG_DATA_LENGTH,
    canmsg_cell3.txMsgData); // We are sending from mailbox #13
}
}


void initSPIAMaster()
{
//
// Must put SPI into reset before configuring it
//
SPI_disableModule(SPIA_BASE);

//
// FIFO configuration
//
SPI_enableFIFO(SPIA_BASE);
SPI_clearInterruptStatus(SPIA_BASE, SPI_INT_RXFF |
SPI_INT_TXFF);
SPI_setFIFOInterruptLevel(SPIA_BASE, (SPI_TxFIFOLevel)FIFO_LVL,
                          (SPI_RxFIFOLevel)FIFO_LVL);
//
// SPI configuration. Use a 500kHz SPICLK and 16-bit word size.
//
SPI_setConfig(SPIA_BASE, DEVICE_LSPCLK_FREQ, SPI_PROT_POL0PHA0,
              SPI_MODE_MASTER, 2500000, 16);
SPI_disableLoopback(SPIA_BASE);
SPI_enableModule(SPIA_BASE);
}

//
// DMA setup for both TX and RX channels.
//
void initDMA()
{
//
// Initialize DMA
//
DMA_initController();

//
// Configure DMA Ch5 for TX. When there is enough space in the
FIFO, data
// will be transferred from the sData buffer to the SPI
module's transmit
// buffer register.
//
DMA_configAddresses(DMA_CH5_BASE, (uint16_t *)(SPIA_BASE +
SPI_O_TXBUF),
                    sData);
DMA_configBurst(DMA_CH5_BASE, BURST, 1, 0);
DMA_configTransfer(DMA_CH5_BASE, TRANSFER, 1, 0);
DMA_configMode(DMA_CH5_BASE, DMA_TRIGGER_SPIATX,
DMA_CFG_ONESHOT_DISABLE |
               DMA_CFG_CONTINUOUS_DISABLE | DMA_CFG_SIZE_16BIT);
```

```
//
// Configure DMA Ch5 interrupts
//
DMA_setInterruptMode(DMA_CH5_BASE, DMA_INT_AT_END);
DMA_enableInterrupt(DMA_CH5_BASE);
DMA_enableTrigger(DMA_CH5_BASE);

//
// Configure DMA Ch6 for RX. When the FIFO contains at least 8
words to
// read, data will be transferred from the SPI module's receive
buffer
// register to the rData buffer.
//
DMA_configAddresses(DMA_CH6_BASE, rData,
                    (uint16_t *)(SPIA_BASE + SPI_O_RXBUF));
DMA_configBurst(DMA_CH6_BASE, BURST, 0, 1);
DMA_configTransfer(DMA_CH6_BASE, TRANSFER, 0, 1);
DMA_configMode(DMA_CH6_BASE, DMA_TRIGGER_SPIARX,
DMA_CFG_ONESHOT_DISABLE |
               DMA_CFG_CONTINUOUS_DISABLE | DMA_CFG_SIZE_16BIT);

//
// Configure DMA Ch6 interrupts
//
DMA_setInterruptMode(DMA_CH6_BASE, DMA_INT_AT_END);
DMA_enableInterrupt(DMA_CH6_BASE);
```

```
    DMA_enableTrigger(DMA_CH6_BASE);
}

//
// DMA Channel 5 ISR
//
__interrupt void dmaCh5ISR(void)
{
    DMA_stopChannel(DMA_CH5_BASE);
    Interrupt_clearACKGroup(INTERRUPT_ACK_GROUP7);
//      DMA_clearTriggerFlag(DMA_CH5_BASE);
//      DMA_forceTrigger(DMA_CH5_BASE);
    return;
}

//
// DMA Channel 6 ISR
//
 __interrupt void dmaCh6ISR(void)
{
//      uint16_t i;

    DMA_stopChannel(DMA_CH6_BASE);
    Interrupt_clearACKGroup(INTERRUPT_ACK_GROUP7);
//      DMA_clearTriggerFlag(DMA_CH6_BASE);

    //
    // Check for data integrity
    //
//      for(i = 0; i < 8; i++)
//      {
//          SPI_readDataBlockingFIFO(SPIA_BASE);
////            if (rData[i] != i)
////            {
////                // Something went wrong. rData doesn't contain
expected data.
////                ESTOP0;
////            }
//      }

//      SPI_resetRxFIFO(SPIA_BASE);
//      done = 1;
    return;
}
 //
 // Configure GPIOs for external loopback.
 //
 void configGPIOs(void)
 {
        EALLOW;

        // GPIO9: SPICLKA
        GpioCtrlRegs.GPAPUD.bit.GPIO9 = 1;    // Disable pullup on
        GPIO3
        GpioCtrlRegs.GPAMUX1.bit.GPIO9 = 0;   // GPIO3 pin set as
        GPIO3
```

```
GpioCtrlRegs.GPADIR.bit.GPIO9 = 1;    // GPIO3 = output
GpioDataRegs.GPASET.bit.GPIO9 = 0;    // Load output latch
GpioDataRegs.GPACLEAR.bit.GPIO9 = 1; // set low

// GPIO5: SPICSA
GpioCtrlRegs.GPAPUD.bit.GPIO5 = 1;    // Disable pullup on
GPIO5
GpioCtrlRegs.GPAMUX1.bit.GPIO5 = 0;  // GPIO5 pin set as
GPIO5
GpioCtrlRegs.GPADIR.bit.GPIO5 = 1;    // GPIO5 = output
GpioDataRegs.GPASET.bit.GPIO5 = 0;    // Load output latch
GpioDataRegs.GPACLEAR.bit.GPIO5 = 1; // set low

// GPIO8: SPISIMOA
GpioCtrlRegs.GPAPUD.bit.GPIO8 = 1;    // Disable pullup on
GPIO8
GpioCtrlRegs.GPAMUX1.bit.GPIO8 = 0;  // GPIO8 pin set as
GPIO8
GpioCtrlRegs.GPADIR.bit.GPIO8 = 1;    // GPIO8 = output
GpioDataRegs.GPASET.bit.GPIO8 = 0;    // Load output latch
GpioDataRegs.GPACLEAR.bit.GPIO8 = 1; // set low

// GPIO10: SPISOMIA
GpioCtrlRegs.GPAPUD.bit.GPIO10 = 1;    // Disable pullup on
GPIO10
GpioCtrlRegs.GPAMUX1.bit.GPIO10 = 0;  // GPIO10 pin set as
GPIO10
GpioCtrlRegs.GPADIR.bit.GPIO10 = 1;    // GPIO10 = output
```

```
GpioDataRegs.GPASET.bit.GPIO10 = 0;   // Load output latch
GpioDataRegs.GPACLEAR.bit.GPIO10 = 1; // set low

EDIS;

//
 // GPIO10 is the SPISOMIA.
 //
 GPIO_setMasterCore(10, GPIO_CORE_CPU1);
 GPIO_setPinConfig(GPIO_10_SPISOMIA);
 GPIO_setPadConfig(10, GPIO_PIN_TYPE_PULLUP);
 GPIO_setQualificationMode(10, GPIO_QUAL_ASYNC);

 //
 // GPIO8 is the SPISIMOA clock pin.
 //
 GPIO_setMasterCore(8, GPIO_CORE_CPU1);
 GPIO_setPinConfig(GPIO_8_SPISIMOA);
 GPIO_setPadConfig(8, GPIO_PIN_TYPE_PULLUP);
 GPIO_setQualificationMode(8, GPIO_QUAL_ASYNC);

 //
 // GPIO5 is the SPISTEA.
 //
 GPIO_setMasterCore(5, GPIO_CORE_CPU1);
 GPIO_setPinConfig(GPIO_5_SPISTEA);
 GPIO_setPadConfig(5, GPIO_PIN_TYPE_PULLUP);
 GPIO_setQualificationMode(5, GPIO_QUAL_ASYNC);
```

```
        //
        // GPIO9 is the SPICLKA.
        //
        GPIO_setMasterCore(9, GPIO_CORE_CPU1);
        GPIO_setPinConfig(GPIO_9_SPICLKA);
        GPIO_setPadConfig(9, GPIO_PIN_TYPE_PULLUP);
        GPIO_setQualificationMode(9, GPIO_QUAL_ASYNC);
}
```