2014

DC MICROGRID STABILIZATION THROUGH FUZZY CONTROL OF INTERLEAVED, HETEROGENEOUS STORAGE ELEMENTS

Robert David Smith

Michigan Technological University

Follow this and additional works at: https://digitalcommons.mtu.edu/etds

Part of the Power and Energy Commons

DC MICROGRID STABILIZATION THROUGH FUZZY CONTROL OF INTERLEAVED, HETEROGENEOUS STORAGE ELEMENTS

By

Robert David Smith

A THESIS

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

In Electrical Engineering

MICHIGAN TECHNOLOGICAL UNIVERSITY

2014

This thesis has been approved in partial fulfillment of the requirements for the Degree of MASTER OF SCIENCE in Electrical Engineering.

Department of Electrical and Computer Engineering

Thesis Advisor:	*Dr. Wayne W. Weaver*
Committee Member:	*Dr. Roger M. Kieckhafer*
Committee Member:	*Dr. Gordon G. Parker*
Committee Member:	*Dr. Timothy C. Havens*
Department Chair:	*Dr. Daniel R. Fuhrmann*

Dedication

For my grandfathers, Robert and David

Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank Dr. Weaver for providing many opportunities to get involved in the design and development of power converter hardware. Likewise, I appreciate his insight and guidance as an advisor. It was a privilege to work with him these last few years.

I want to also thank the other professors in the Electrical and Mechanical Engineering departments. Dr. Kieckhafer, Dr. Havens, and more have all helped me discover fun and intriguing aspects of engineering. I also would like to thank the gentlemen in the Lab Services group for sharing their wealth of practical knowledge.

I also want to express my appreciation to the music community at Michigan Tech, especially the Brothers of Mu Beta Psi. The delights of playing music and the fellowship of the musicians is something I could never and would never replace.

Finally, I would like to thank my parents, grandparents, and sister for their unfailing love and support throughout my entire education. Without them all, it would have been nigh impossible to make it through this journey.

Abstract

As microgrid power systems gain prevalence and renewable energy comprises greater and greater portions of distributed generation, energy storage becomes important to offset the higher variance of renewable energy sources and maximize their usefulness. One of the emerging techniques is to utilize a combination of lead-acid batteries and ultracapacitors to provide both short and long-term stabilization to microgrid systems.

The different energy and power characteristics of batteries and ultracapacitors imply that they ought to be utilized in different ways. Traditional linear controls can use these energy storage systems to stabilize a power grid, but cannot effect more complex interactions. This research explores a fuzzy logic approach to microgrid stabilization. The ability of a fuzzy logic controller to regulate a dc bus in the presence of source and load fluctuations, in a manner comparable to traditional linear control systems, is explored and demonstrated. Furthermore, the expanded capabilities (such as storage balancing, self-protection, and battery optimization) of a fuzzy logic system over a traditional linear control system are shown. System simulation results are presented and validated through hardware-based experiments. These experiments confirm the capabilities of the fuzzy logic control system to regulate bus voltage, balance storage elements, optimize battery usage, and effect self-protection.

Chapter 1

Introduction

One of the main challenges with integrating renewable energy into a power grid is the variable nature of many types of sources. Cloud cover over a solar panel cannot be controlled and wind speeds can change with no notice. This is usually not a problem with large-scale power grids ("macrogrids"), as there is enough overhead and spare capacity amongst the large power plants to absorb such fluctuations. On a small-scale grid (a "microgrid" as described by Hatziargyriou et al. [1]: low and medium-voltage generation & distribution systems with close geographic proximity between generation and consumption), these variations can still be absorbed by generation, but at a higher cost.

For a macrogrid power system, the amount of lost power from inefficient use of renewable sources is very small compared to the total power generated, and is usually ignored. In microgrid systems, especially isolated ones, energy can have a significantly higher cost than a macrogrid system. (One example is that of military forward operating bases, as analyzed by Prado et al. [2].) In those situations, it is useful to apply maximum power point tracking ("MPPT") algorithms to obtain as much energy as possible from renewable sources. This requires a trade-off between efficiency and stability; a fixed-generation unit like a diesel generator can run with spare capacity to improve stability, but this typically comes at the expense of efficiency. The inverse can also be true; the same diesel generator, running at capacity, will have a diminished ability to regulate the power level if power input from renewable sources suddenly changes.

This thesis explored an alternative technique to low-voltage dc grid stabilization. The linear Proportional-Integral (PI) controller is a very common method of controlling a system's output based on a reference input. The alternative control scheme examined is based on fuzzy logic. Also, the control of energy storage elements has the additional aspect of having different types of storage elements interleaved in a monolithic converter. Having immediate knowledge of the states of the different storage systems present on the grid allows for additional considerations to be taken when managing the power flow between all of the elements.

The overall outline of the remainder of this thesis is as follows:

Chapter 2: Background on the topics of power electronics, energy storage, fuzzy logic, and grid stabilization are presented, giving the reader a summary of the critical points of those fields used in this research and a review of recent and related technological developments in the same. Additionally, the goals of this research effort are detailed.

Chapter 3: Two key topics are presented in this chapter: the development of hardware to support experimental research into microgrid control algorithms, and the theories and methods used as the basis for the research presented herein.

Chapter 4: The development of a computer simulation to predict the behaviors of the PI and fuzzy logic control systems is detailed here, as well as the results of the same.

Chapter 5: An experiment (based on hardware) to show the implementation and effectiveness of the fuzzy logic stabilizer is described here. Equipment setup, control algorithm development and implementation, scenario setup, and the results of the same are given here.

Chapter 6: This chapter discusses the results of both the simulations and the physical experiments, comparing and contrasting the two control systems. Concluding remarks are given on the effectiveness of the fuzzy logic control system for select applications.

Chapter 7: Possibilities for future work are suggested here.

Chapter 2

Overview

2.1 Background

2.1.1 Power Electronics

In the 1800s and early 1900s, power conversion was largely accomplished through transformers and/or rotating machinery. Both methods occupy a lot of physical space, and the rotating-machine method suffers from the inefficiency of converting from electrical energy to mechanical energy and back again. In the mid-1900s, great advances started to be made in the area of solid-state electronics, specifically transistors. Reductions in size and cost were achieved, as well as increased flexibility in controlling the flow of electricity. Power electronics, solid-state electronics meant to handle sizeable quantities of electrical energy, became the preferred method of converting and managing power in the sub-kilovolt range [3]. Power electronics, as well as electronics in general, continued to shrink in size and increase in efficiency and capability through the late 1900s and into modern times. They are practically required wherever portable electronics may be found.

Not only are power electronics useful for small-scale electronics, but the ability to effectively handle hundreds volts and/or amperes puts them in the realm of power grid interfacing. Richmond et al. [4] outline recent developments (particularly those based on silicon carbide) which have produced power switches with sufficiently-high ratings as to be utilized on the power distribution scale. Of particular note are their usefulness in relation to renewable energy. One instance of this effect is seen in solar panels; the innate dc output of solar panels needs conversion to properly work with the greater ac power grid, and power electronics are well-suited towards fulfilling this requirement.

One additional benefit of power electronics is the ability to closely integrate them with programmable controls. Controlling a steam turbine requires a complex interplay of electrical and mechanical components; exerting control on a power transistor can be as simple as changing a digital signal. Developments in microprocessing units has allowed power converters and their control systems to scale downwards in size together, so that a system that converts several kilowatts of power can easily fit in a regular shoebox 3.1.2.

2.1.2 Energy Storage

Mechanical systems were the first ways mankind stored energy. Despite rapid advances in other fields, mechanical energy storage technology has continued to develop as well. Common modern storage systems [5] include flywheels, pumped-storage hydroelectrics, compressed-air storage, and gravity potential storage. Many of these types are best suited towards bulk energy storage rather than grid stabilization[1], as the responsiveness of such systems is rather slow despite the ability to store large quantities of energy [6].

Another form of energy storage is that of thermal storage, one that also has some overlap with solar renewable energy. A large imbalance of thermal energy is maintained as a managed potential. Energy is stored by increasing the imbalance; harnessing the flow of energy towards equilibrium makes use of that stored energy. With the climate control requirements imposed by humans, it is sometimes more efficient to simply not involve thermal energy storage with grid-scale electrical energy requirements. For example, a solar-based pre-heater for a building's hot water supply might as well not bother with a multi-step thermal-electrical-thermal energy conversion. Another

[1]Flywheels are one notable exception to this.

example would be that of a re-purposed coal mine in Nova Scotia, Canada, where the latent thermal energy present in an old coal mine was processed through a heat pump and used to provide year-round temperature regulation on an industrial scale [7].

Some chemical energy storage processes involve converting electrical energy to fuel, including both conventional electrolysis and methane production. Such technologies are, however, better suited towards bulk energy storage due to their slow responsiveness [8].

Electrochemical storage is by far the most prevalent form of storing energy on the microgrid scale and lower (e.g. device scale). The majority of electrochemical storage devices take the form of batteries and capacitors[2]. With regards to storing energy on the microgrid scale, batteries and ultracapacitors are the two types of electrochemical devices that have sufficient energy and power capacity for use in this research.

In this research, lead-acid batteries are one of the two forms of energy storage utilized. Their ubiquitousness and established recycling infrastructure make them well-suited for small scale power storage in addition to their more common use in starting internal-combustion engines. Jenkins, Fletcher, and Kane demonstrate the usefulness of lead-acid batteries in micro-generation scenarios [9], such as a single household with a

[2]More accurately speaking, some capacitors types are better described *not* as electrochemical devices, such as ceramic or tantalum or film capacitors. They are more akin to purely electrical storage. Still, for the sake of simplicity, they are included here alongside electrolytic capacitors due to their similar behaviors.

photovoltaic (PV) panel, small wind turbine, or other kilowatt-level sources of energy. While said batteries are very useful, they note that irregular charging and discharging of the batteries places significant stress upon them; this shows a potential usefulness for integration with other forms of energy storage to mitigate that irregularity.

The other form of energy storage used in this research is that of ultracapacitors. More correctly called electrolytic double-layer capacitors, they were first created in the late 1950's and commercialized in the late 1970's by Nippon Electric Company [10]. Lai, Levy, and Rose describe [11] characteristics of ultracapacitors: key features relevant to this research are their very high power density, lower energy density, low voltage ratings, and high cycle lifespan. In simplified terms, ultracapacitors cannot store much total energy, but they have the ability to source or sink very high levels of power for brief periods of time. Their low voltage requires several units to be connected in series in order to be utilized in a microgrid, and the long lifespan of the units makes them suitable for dealing with frequent variations in loads. They form a good complement to batteries, which can store vastly more energy but are limited in their ability to source or sink it rapidly.

2.1.3 Fuzzy Logic

One of the first mentions of fuzzy logic came in the form of fuzzy sets, proposed by Klaua and Zadeh in the mid 1900s. In this work [12], Zadeh proposes an extension on the classical definition of a set: a degree of membership in a classification. This associates numerical data with linguistic adjectives. For example, if Robert Smith is 179cm tall, he would be *mostly normal* and *slightly tall*[3]. This lends a great deal of intuition to humans, who naturally think in such terms. Zadeh and Klaua's work lay the foundation to impart this type of behavior and analysis to a computational system. With these mathematical transformations, a computer can consider values in such terms as *very low* or *a little large* or many others.

This process works both ways for a system. Both inputs and outputs can be described in terms of fuzzy sets, and the interface in the middle is a set of rules. Fuzzy rules translate input memberships to output memberships. An example of a rule might be "if *bus voltage* is *very low* and *battery voltage* is *slightly high*, then *battery output* is *somewhat positive*." This rule shows how linguistic definitions can be applied to real-world values. One of the key processes in fuzzy logic is combining several rules with different weightings. The phrase *mostly neutral, slightly positive* has meaning to

[3]At least, in North America he would likely be considered such. In eastern Asia he might be considered *slightly normal* and *somewhat tall*. By contrast, the terms *mostly normal* and *slightly short* would better describe him if in northern Europe. The context in which a number relates to an aspect of a system requires significant consideration when constructing membership functions.

a human, but is useless to a computer unless translated into an actual value. Because input values often have non-zero membership in multiple functions, it is usually the case that multiple fuzzy rules apply for a given set of inputs.

The process of defuzzification takes a set of membership levels created by the fuzzy rules and generates real-world output values from that collection. There are a variety of algorithms for defuzzification, and evaluating different functions for optimal performance can be a significant undertaking in and of itself [13]. The general process of defuzzification is to take the aggregated membership levels and form an output that relates to the magnitudes of the collective. One of the simplest forms of defuzzification functions is to take just the largest membership function to form the output. For example, if there are three rules triggered by the input membership functions [*about zero, somewhat positive, very positive*] with triggering weights $[0.2, 0.8, 0.35]$, then the output corresponding to *somewhat positive* will be triggered and the others will be discounted. This is incredibly simple, but it does not represent some situations well. What if the triggering weights were $[0.4, 0.6, 0.1]$? In that case, it is more intuitive to have a blend of the two larger rules. This is often represented in another common form of defuzzification: Center of Area. In this case, the logical union of the output membership functions is constructed, and the center of the resulting area is computed to produce an explicit output value. Higher rule triggering weights will have a more pronounced effect in shifting the aggregate center towards the rule's individual center, while lower triggering weights will have a more modest impact.

2.2 Related Research

Thounthong et al. [14] performed very similar research to that being proposed here. In their system, a photovoltaic generator was interleaved with a fuel cell and an ultracapacitor bank. The fuel cell was used as a direct supplement to the PV generator, and a (rather straightforward) fuzzy control system exerted stabilizing influence on the bus through the ultracapacitor bank. Their experimental results show good regulation of the bus voltage in the presence of supplementary power generation. Sathishkumar, Kollimalla, and Mishra also performed a study [15] regarding the stabilization of PV source with a combined battery and ultracapacitor system. Their control system separated the load variation into long-term and short-term variation, using the battery system to counteract the long-term variation and the ultracapacitor bank to handle the short-term variation. Their simulations predict reduced stress on the battery system when integrated with an ultracapacitor system to handle the short-term, high-power requirements.

The concept of combining lead-acid batteries with ultracapacitors is not a new concept. Stienecker, Stuart, and Ashtiani proposed such a system [16] as a way of leveraging the lower cost of lead-acid batteries (as opposed to nickel-metal hydride batteries) in hybrid electric vehicles. Their approach treats the ultracapacitor system as a selectable alternative to the battery. While this limits the capabilities somewhat, it shows considerable cost-effectiveness under the appropriate conditions. Haifeng and Xueyu performed a similar study[17], comparing the vehicle-based alternate-source strategy examined by Stienecker et al. with one based on a dc converter via power electronics. They showed that both methods result in better preservation of battery life, though utilizing a dc converter can provide more flexibility given adequate control algorithms.

Ultracapacitor energy storage has not been limited to just dc microgrid applications. Bilbao et al. performed a study [18] that utilized an ultracapacitor bank to stabilize an ac microgrid (using a dc link). They correctly noted that battery systems not only have a higher weight-to-power ratio, but also tend to have uneven charge and discharge characteristics (whereas an ultracapacitor system avoids these deficiencies at the expense of a low weight-to-energy ratio). An additional aspect they investigated was the functionality of the stabilization system when transitioning from grid-connected mode to islanded mode. One assumption made in that study was how the storage system would not have to account for a long-term surplus or deficit of energy; the primary focus was towards examining the ability of such a system to provide short-term stabilization, which it properly accomplished.

An unconventional method of interleaving a battery bank with an ultracapacitor bank was proposed by Onar and Khaligh [19]. Their proposed system magnetically couples the inductors of the two storage banks by placing both on a single core, with the benefit of reducing the size and cost of the inductors required by the system. An additional benefit is how the ultracapacitor bank is maintained at a minimum state of charge by partial absorption of transients in the system. Their experimental results show that the system can effectively counteract both short and long-term transient behavior using the coupled-inductor structure.

Papadimitriou and Vovos have performed several studies into the application of fuzzy control to grid stabilization. In one case [20] they examined how fuzzy control of a fuel cell / battery hybrid stabilizer can provide useful operation in both islanded mode and in grid-connected mode, despite a historical tendency to limit the flexibility of distributed systems operating in grid-connected mode. In addition to supply regulation in islanded mode, the system operates to correct power factor deficiencies when connected to a main power grid. A key feature of having both a fuel cell system and a battery bank is to accommodate different response times: the fuel cell is slower, but can provide more energy in the long-term, while the battery bank has the opposite characteristics. In a subsequent study [21], they extend this concept to coordinate the fuzzy-controlled fuel cell and battery system with a doubly-fed induction generator (whose input is modeled to be that of a wind turbine) for operation in both islanded and grid-connected modes. In this scenario they again demonstrate the ability of the fuzzy control system to provide both active and reactive power regulation in the presence of external grid influence and local generation.

2.3 Objectives

To support this research, a dc microgrid is modeled and constructed. Four main elements comprise the system: a stochastic power source, a stochastic impedance load, a fixed impedance ballast, and a stabilizer unit based on energy storage elements. Figure 2.1 gives a simplified representation of this system.

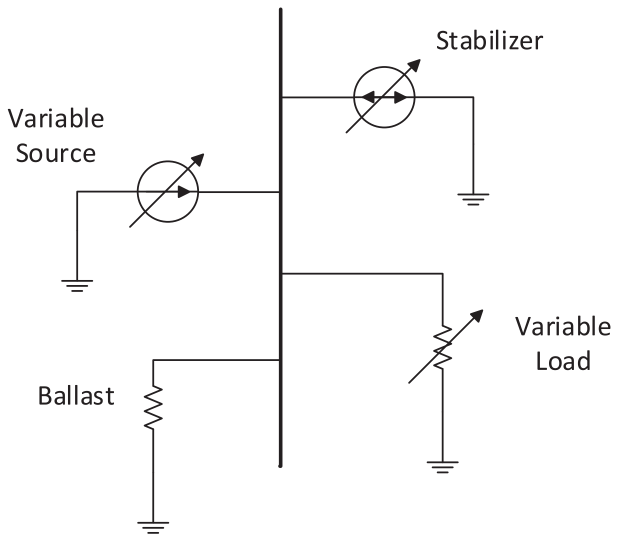


Figure 2.1: Simplified Microgrid System

This research effort is focused around the specific comparison of a fuzzy-logic (FL) control algorithm to a traditional PI control algorithm. The main point under consideration is to show that a fuzzy logic controller can stabilize a dc microgrid to within a given tolerance range in a manner comparable to that of a PI controller, while having an improved treatment of the battery system when compared to the aforementioned

PI controller. Two secondary points will also be considered. The first is showing how the performance of a battery/ultracapacitor system under FL control can execute a balancing effect between storage elements, transferring excess energy in one element to another that is depleted. The second is showing how FL control can exert self-limiting functionality to protect the battery and ultracapacitor systems. A PI control system is typically not able to perform this, requiring additional control algorithms for such a feature.

Chapter 3

Methodology

With the goals of this research stated, the process by which they are to be accomplished must be specified. This chapter details the procedures and methods used to effect bus regulation, reduced battery usage, inter-element energy transfer, and system self-protection.

3.1 Hardware Design

3.1.1 Design Goals

The design process of power converter hardware was rooted in the requirements of the system. The overarching goal was to have a power conversion and control platform that could be scaled upwards and downwards to facilitate experiments in any of a multitude of microgrid research topics. Previous design work at Michigan Technological University was primarily oriented around creating an arbitrary multi-phase power converter that could be controlled through MATLAB/dSPACE. With the desired expansion of microgrid research capabilities at Michigan Tech, the direction of design work had to expand beyond just the design of an arbitrary power converter.

To conduct a wider variety of experimental research, the systems that make up the microgrid test system are designed around three layers, each with its own functionality and device-scale implementation. The actual power switching and measurement devices make up the lowest level, which takes the form of a multi-phase arbitrary power inverter. The middle contains the servo-level control of the power inverter and data reporting to the upper layer; this is implemented in a microcontroller-based control unit. The uppermost layer consists of high level power management algorithms and inter-grid communication; this functionality is found in the form of a single-board computer. To support this research, the first two levels were implemented through hardware designed and built for this purpose.

3.1.2 Power Inverter

The central component of the power inverter is an integrated power module (IPM). This unit combines power semiconductors (IGBTs, in this instance) with basic control and protection functions on a single chip. Many different types of modules exist, with different features[1] and power ratings. For the purposes of experimental microgrid research, the ability to handle a few kilowatts of power per unit is sufficient, maintaining a balance of capability and small size. The specific unit chosen for this design is the PS21765 from Powerex Inc and Mitsubishi Electric [22]. Figure 3.1 shows the configuration of the power semiconductors.

In addition to the IPM, the power inverter has circuitry for taking voltage and current measurements on all of the phases as well as the high bus. For voltage measurements, a resistor divider and an active low-pass filter condition the signal to a range suitable for microcontroller analog-to-digital converters (ADCs). For current measurements, hall-effect transducers are used instead of the more common (and noticeably cheaper)

[1]Depending on the model, such functions may include shoot-through protection, deadband insertion, overcurrent shutoff, undervoltage lockout, and others.

low-side resistor technique. Despite the extra cost, the advantages to using high-side hall-effect transducers include a small reduction in measurement losses and a reduced chance of not recognizing an external short-to-ground fault condition.

As the inverter must handle both analog measurement and power switching, various techniques are used in the PCB design to reduce electromagnetic interference (EMI). The most common technique is implementing a multi-layer printed circuit board (PCB). Dedicated ground and power planes provide low-level decoupling across the entire circuit board. Creative usage of ground plane segments can also act as a shield for EMI. A second technique used in the PCB design is that of star-topology grounding. While the grounds for both the measurement circuitry and power-switching devices are electrically connected, the portions are physically segregated and then tied together at a single point elsewhere. This greatly reduces the propagation of EMI from the power switching area to the measurement area.

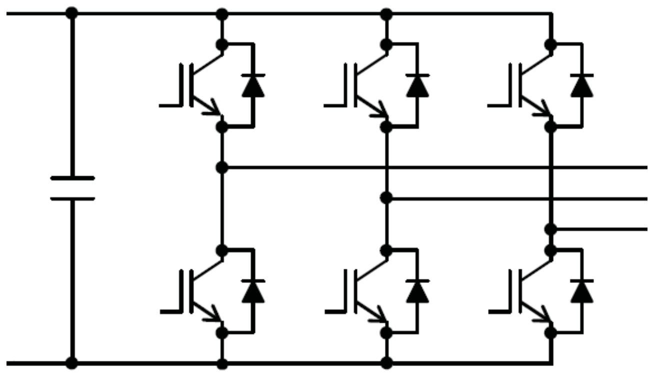


Figure 3.1: IPM Switch Configuration

With the inverter specified to handle power in the kilowatt range, thermal issues had to be resolved. The two main considerations were board trace sizing for thermal rise and heat sinking for the IPM. Solving the former issue is very straightforward, for data tables relating PCB trace size, current, and temperature rise are readily found in technical standards [23]. The latter issue is slightly more complex, as a series of thermal interfaces must be considered when calculating power dissipation. It was determined that ambient convection alone would not be sufficient to cool the unit, and forced airflow would be required.

Figure 3.2 shows a completed power inverter unit. At lower voltages (on the order of 100 V on the phase side), each individual phase can convert approximately 1 kW of power, with the entire unit being able to convert 1.5 kW as a whole. At higher voltages (over 250 V), those capabilities roughly double. While fully functional, the unit requires an external control system for operation.

3.1.3 Servo Controller

The control unit must exert tight, rapid command over the power converter. At the same time, it should gather data regarding the converter's operation for reporting to the upper layer. Finally, it must receive and implement commands from an upper layer supervisor. These requirements make a microcontroller-based solution a clear

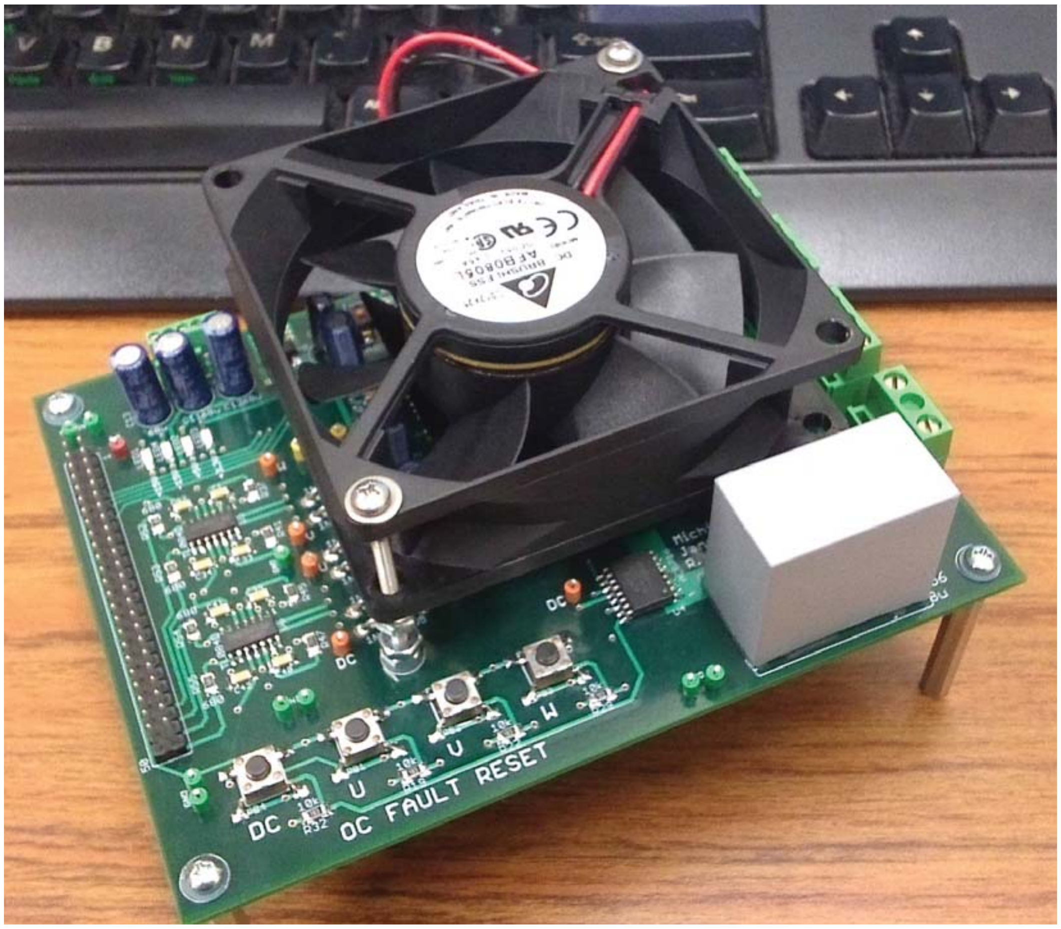


Figure 3.2: Completed Power Inverter

choice.

Given the ultimate goal of conducting microgrid control research, it would be beneficial to have a system that facilitates rapid control development and deployment. Ideally, such a system would be programmed by a high-level language such as MATLAB/Simulink or LabView. This would allow for reduced development times when implementing control schemes.

The C2000 series of microcontrollers (MCUs) from Texas Instruments is well-suited

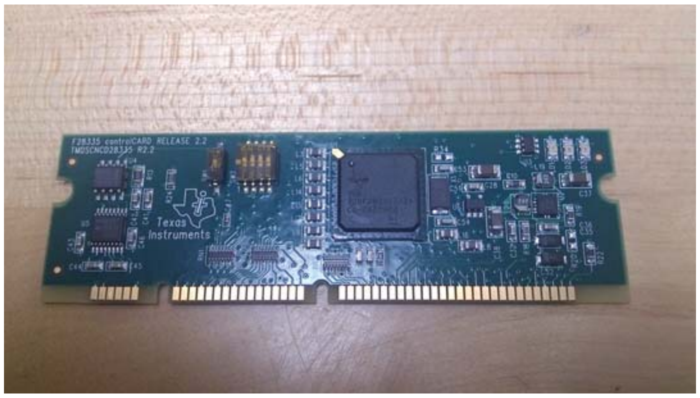


Figure 3.3: C2000 Delfino controlCARD

toward fulfilling these requirements. In addition to common MCU peripheral units such as communication modules and timers and ADCs, C2000 series MCUs contain standalone PWM modules. Having these capabilities reduces the load on the CPU and timers for generating the appropriate PWM waveforms for the power inverter, as well as providing fault signals directly to the modules for increased responsiveness (instead of requiring a CPU interrupt routine to react to a fault condition).

TI makes a series of evaluation modules [24] for C2000-series MCUs called controlCARDs (see Figure 3.3). Units are mounted on an interface PCB with supporting circuitry and connectors. The same units can be programmed through a simple dock with a USB interface. Standard IDE software can load programs, created either manually or automatically, onto the flash memory in the units for independent operation.

3.2 Supporting Software

Historically, microcontrollers have been programmed in mid-level languages (most commonly C and assembly). Modern software has advanced to the point where a great deal of work can be accomplished using high-level languages and automatic code generators (ACGs). These ACGs can produce code in a mid-level language from original high-level programs. Many compilers contain program interfaces which allow the ACGs to automatically compile the mid-level code into a low-level executable through the compiler. MCUs cover an interesting area where the benefits of using one language level might not dominate all others. In this research, some of the mid-level outputs of the high-level ACG present in MATLAB/Simulink are manually modified with additional code and recompiled into low-level executable code.

One of the leading commercial software packages for rapid control development is a Mathworks product suite called MATLAB/Simulink [25]. The MATLAB system forms a base engine for numerical processing, and the Simulink portion allows for intuitive, graphical modeling and programming of control systems. Software expansions have been created for MATLAB/Simulink for a variety of purposes, including ones specifically oriented towards programming embedded systems from high-level Simulink graphical code (including C2000 series MCUs and others, such as the Arduino, BeagleBoard, and Raspberry Pi).

In addition to MATLAB/Simulink software suite, the software package Code Composer Suite (CCS) from Texas Instruments [26] is required to compile the code from C language into machine code that is loaded directly onto the MCU, and the controlSuite plug-in for CCS is used by MATLAB/Simulink's Embedded Coder software expansion to generate the appropriate C code.

3.3 Microgrid Configuration

3.3.1 Simplified Model

Figure 3.4 (first given as Figure 2.1 and repeated here again) gives the most basic representation of the microgrid under consideration. The four elements present are a variable source[2], a variable load, a bi-directional stabilizer unit, and a ballast load. All four elements are connected with a common bus and ground.

While this system is very easy to understand and model, a practical microgrid is made up of more complex devices. Sections 3.3.2 through 3.3.5 detail how the four elements in the simplified microgrid system are modeled and constructed in much greater detail.

[2]Though the source is shown as a current source, the current is specifically controlled to emulate a constant power source.

3.3.2 Stochastic Source

The specific choice of what *type* of stochastic source depends on the situation. The context of this research is that of a microgrid arrangement. To obtain the greatest benefit from a renewable source, control algorithms are often implemented as MPPT controls. With this in mind, the stochastic source is configured as a constant power source. A pseudo-random number generator provides a target power, and the associated converter tracks to this level. In both simulation and experimental work, this system takes the form of a boost converter. To obtain the given power, the control routine measures the input voltage and exerts control on the duty cycle to maintain the input current that corresponds to the given power level.

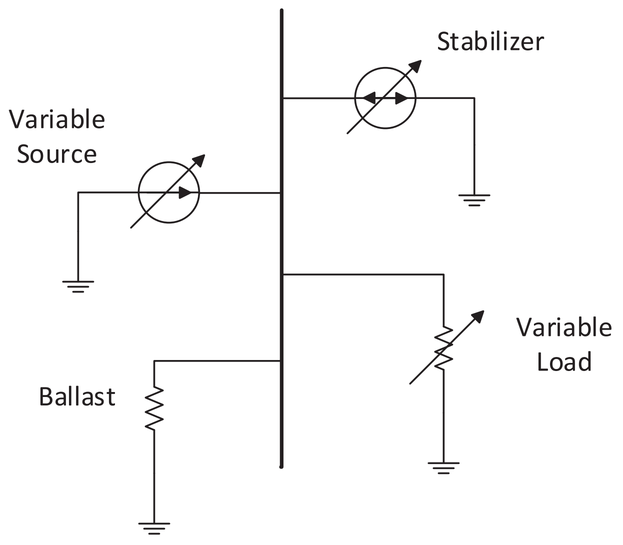


Figure 3.4: Simplified Microgrid System

Figure 3.5 shows the schematic used as the basis for implementing and simulating the stochastic power source. It include basic loss modeling in the form of added resistances on both the low side and high side of the switch. (The high-side resistor is especially useful for solving Kirchoff's Current Law to determine the bus voltage.) Equations

$$L\frac{di_{src}}{dt} = V_{src} - i_{src}R_l - (1-q)v_C \tag{3.1}$$

and

$$C\frac{dv_C}{dt} = (1-q)i_{src} - \frac{v_C - V_{bus}}{R_h} \tag{3.2}$$

give the differential equations that describe the converter's behavior. It should be noted that the inductors modeled in this research are assumed to be operating in continuous conductance mode. (Furthermore, in these and all other differential equations given in this research, all lower-case variables are time-varying signals, and have the (t) suffix removed for simplicity, while all upper-case variables are constant values.)

By measuring the low-side voltage and controlling the low-side current into system,

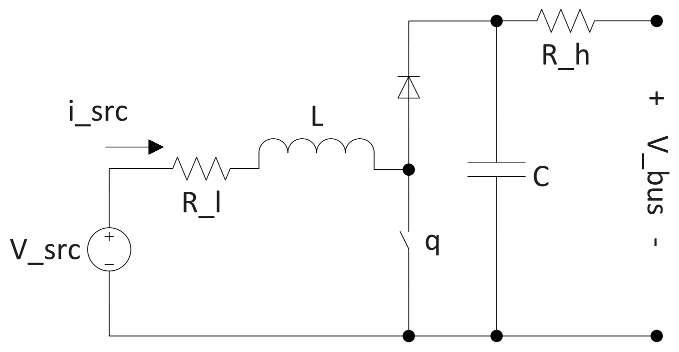


Figure 3.5: Schematic of Stochastic Source

the output of the converter into the bus is not a truly constant power source, as the losses inherent in the inverter are non-linear. However, the variation of those losses is small compared to the total power supplied to the bus, and thus the non-linearity can be ignored with a reasonable expectation of accuracy using a linear loss model.

3.3.3 Stochastic Load

The specific type of stochastic load is that of a constant impedance load. First, a pseudo-random number is generated to become the power draw at the nominal bus voltage for a period of time, then the equivalent nominal resistance is calculated and imposed upon the grid. This load type was chosen instead of a constant power load for simplicity, as constant power loads have an inherent destabilizing effect on power grids [27]. Despite the reduction in complexity, the variability is still sufficient to show the operation of both the PI and FL control systems.

The matching of a power source with an impedance load results in an innate form of stability in the interoperation between the two devices. Figure 3.6 shows the how the source and load lines interact. Assuming finite limits of operation regarding voltage, current, resistance, and power, an equilibrium will always be reached.

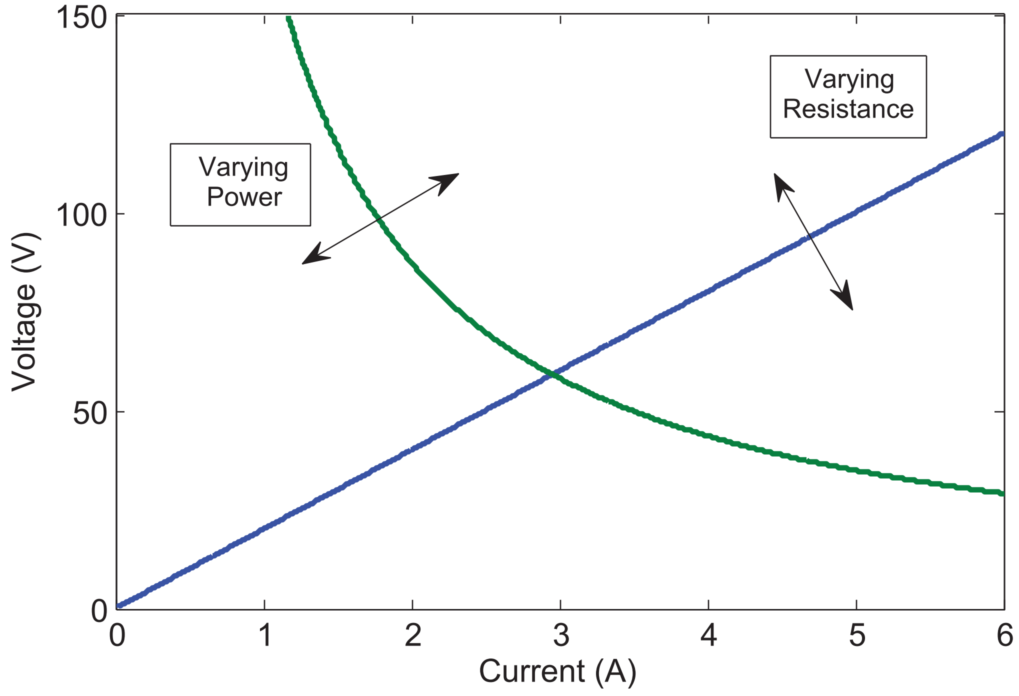


Figure 3.6: Load & Source Lines for Power Source and Impedance Load

The effect of a random impedance upon the bus is a straightforward implementation of Ohm's Law. However, a true, continuously-variable impedance is more difficult to implement in hardware. It is much easier to construct a unit that *emulates* a given impedance. Given a fixed resistance, a control unit can regulate the voltage imposed across the resistance. That load voltage fluctuates in relation to the bus voltage: higher bus voltages will result in higher load voltages, emulating the effect of a resistive load. For an ideal load with neither inductance nor losses,

$$R_{eff} = \frac{R_{conv}}{D_{conv}} \tag{3.3}$$

describes the effective resistance imposed upon the bus given the actual load resistance and the duty cycle of the converter. The problem the ideal situation (3.3) describes is that it produces very high peak currents in practice, leading to poor bus regulation. An improvement would be to add an inductor to reduce the peaks, but that would also have the effect of increasing the effective resistance due to its complex impedance, making the direct proportionality invalid.

An easier method of emulating a random resistance is to control the voltage across a known load resistance, scaling it with respect to the bus voltage. The relationship between effective bus resistance and implemented load is shown by

$$P_{nom} = \frac{V_{bus-n}^2}{R_{eff}} = \frac{V_{load-n}^2}{R_{load}} \quad \rightarrow \quad P_{act} = \frac{V_{bus-a}^2}{R_{eff}} = \frac{V_{load-a}^2}{R_{load}}, \tag{3.4}$$

with respect to actual and nominal bus voltages ("$-a$" and "$-n$" suffixes, respectively). Equation

$$P_{act} = \frac{P_{nom}}{P_{nom}} \frac{P_{act}}{1} = P_{nom} \frac{\frac{V_{bus-a}^2}{R_{eff}}}{\frac{V_{bus-n}^2}{R_{eff}}} = P_{nom} \frac{V_{bus-a}^2}{V_{bus-n}^2} \tag{3.5}$$

rearranges (3.4) for the effective resistance to relate to a nominal power and bus voltage levels. Equation

$$V_{load-a} = \sqrt{R_{load} P_{nom} \frac{V_{bus-a}^2}{V_{bus-n}^2}} \tag{3.6}$$

combines (3.4) and (3.5) to relate nominal power, nominal bus voltage, actual bus voltage, and load resistance (which are all fixed or measured values) to the actual voltage that must be imposed upon the load.

Figure 3.7 shows the schematic of the stochastic load's experimental implementation. (Current polarity i is into the load resistance R_{load} from the bus). Equations

$$0 = i(R_{load} + R_l) + L\frac{di}{dt} - v_C\, q \tag{3.7}$$

and

$$\frac{V_{bus} - v_C}{R_h} = C\frac{dv_C}{dt} + i\, q \tag{3.8}$$

describe the behavior of the unit. The values of the passive components used in the experimental hardware are $R_h = 0.2\ \Omega$, $C = 1.018$ mF, $L = 1$ mH, $R_l = 0.4\ \Omega$, and $R_{load} = 25\ \Omega$. For decreased ripple current, the load is expand to be a two-phase interleaved buck converter, with each phase having the same inductance and resistances as shown in Figure 3.7 and the power draw divided equally between the two phases.

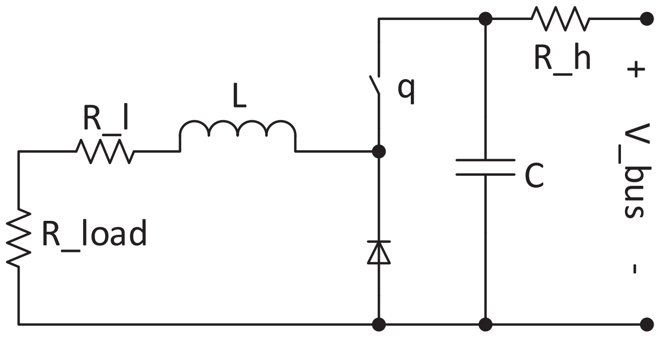


Figure 3.7: Schematic of Stochastic Load

3.3.4 Stabilizer

The grid stabilizer is implemented as a three-phase interleaved converter. Two of the phases are dedicated towards interfacing with energy storage elements, while the last phase is reserved for dissipating excess energy. Figure 3.8 shows the overall schematic of the stabilizer as modeled[3] and constructed in this research.

[3]Each phase's pair of IGBTs is operated in purely complementary fashion.

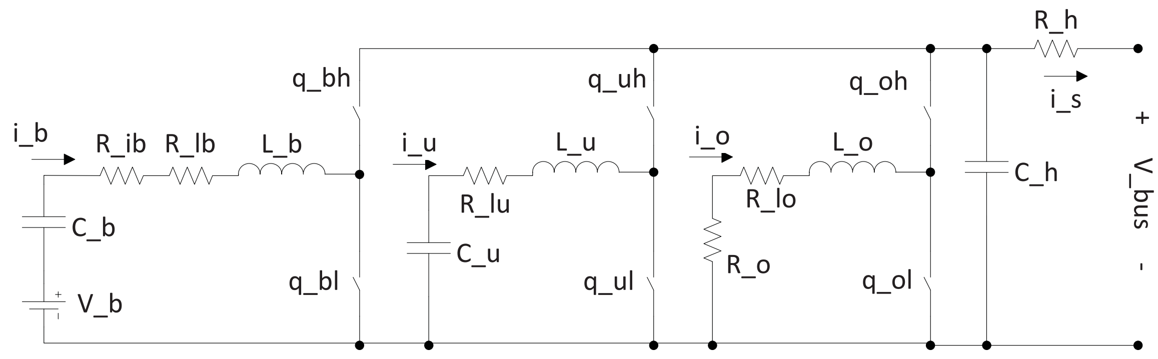


Figure 3.8: Grid Stabilizer Schematic

The high-side capacitor is used to suppress transient spikes in both the operation of the stabilizer and with respect to interfacing with the common bus. The capacitance show here is implemented in hardware as a combination of capacitor types: low-capacity film type for high surge current suppression and high-capacity aluminum electrolytic type for bulk storage. The interaction between the stabilizer and the bus system is determined by

$$\frac{v_{bus} - v_{C_h}}{R_h} = C_h \frac{dv_{C_h}}{dt} + q_{bh} i_b + q_{uh} i_u + q_{oh} i_o. \tag{3.9}$$

The specific values used in the simulation and closely matched in the hardware are $C_h = 1.018$ mF and $R_h = 0.2\ \Omega$.

3.3.4.1 Battery Bank Phase

One of the phases of the stabilizer inverter is dedicated to interfacing with a battery bank. There are several types of batteries that could have been used for this purpose. Lead-acid batteries were chosen for their ease of use and availability. Lithium-ion batteries are another popular type, and while they are superior to lead-acid batteries in energy density (by mass) [28], lead-acid batteries have similar power density and are cheaper to find in higher capacities. Four 12 V, 10 Ah units were connected in series to form this energy storage bank.

While the curves relating cell voltage to state of charge ("SoC") are rather non-linear as a whole, the middle portion of the operating range (typically between 20% and 80% SoC) *does* exhibit a near-linear relationship [29]. Thus if the battery operation and analysis thereof is modestly constrained in this manner, the models and control systems are greatly simplified. For use in this middle range, the battery is modeled as a base voltage with a large, additional capacitance added on. The SoC range in consideration is between 11.8 V and 12.7 V per battery, making the bank SoC range from 47.2 V to 50.8 V. The behavior of the battery bank phase as part of the stabilizer unit is described by

$$V_b + v_{C_b} = i_b(R_{ib} + R_{lb}) + L_b\frac{di_b}{dt} + q_{bh}v_{C_b} \tag{3.10}$$

and

$$C_b \frac{dv_{C_b}}{dt} = -i_b. \tag{3.11}$$

The specific passive component values used in simulation (and closely matched by hardware) are $V_b = 47.2$ V (the 0% SoC level), $C_b = 3$ kF, $L_b = 1.1$ mF, $R_{ib} = 0.2\ \Omega$, and $R_{lb} = 0.4\ \Omega$. The value of capacitance is undersized by a factor of 3 from the full battery capacity. This is for restriction of the usable capacity due to the linearization mentioned previously, to account for reduced capacity on account of ageing with the units used, and to be conservative in the estimate.

One additional complication is that lead-acid batteries have a non-negligible internal resistance, as noted by Ichimura et al. [30] and many others. This is accounted for by the addition of the resistance R_{ib} in the battery bank phase of the stabilizer (see Figure 3.8). The additional inaccuracy is included in the simulation to observe its effects on the FL system's performance.

The power conversion system is configured to be that of a bi-directional dc-dc converter, with the battery bank occupying the "lower-voltage" side and the bus connecting to the "higher-voltage" side. (See Krein [3] for a detailed analysis of this topology.) While all batteries exhibit self-discharge, the time scales considered in this research are so short by comparison that such can be safely ignored.

3.3.4.2 Ultracapacitor Bank Phase

Like the battery bank, the ultracapacitor bank is connected on the "lower-voltage" side of a bi-directional dc-dc converter. Because individual ultracapacitors have very low voltage ratings [31], units are connected in series to form a bank in order to bring the storage phase into useful voltage levels for interfacing with the bus. The specific configuration used in this research is that of a 150 F bank rated at 54 Vdc, made up of 20 BCAP3000 units by Maxwell Technologies [32]. Being a capacitor, modeling and control of the bank is very straightforward.

Because the bank needs to be within a certain voltage range for effective operation, the point at which SoC is considered zero is not at 0V, but at a higher level. This was chosen to be 20 V, which will keep the bank in a voltage range that has a reasonable conversion ratio with a nominal bus voltage of 100 V. Additionally, the point at which the bank is considered to be completely full is 50 V, which allows for a margin of safety before the bank exceeds its rated voltage and risks permanent damage. These restrictions modestly limit the usable energy storage capacity, and the lower limit is illustrated in Figure 3.9. One additional aspect of restricting "usable" energy to that between 20 V and 50 V is that the energy-voltage curve over this interval (in Figure 3.9) is close to linear with respect to voltage, with less than 10% error between the actual curve and the linearization. This linearization is made throughout this research for simplified understanding and calculation.

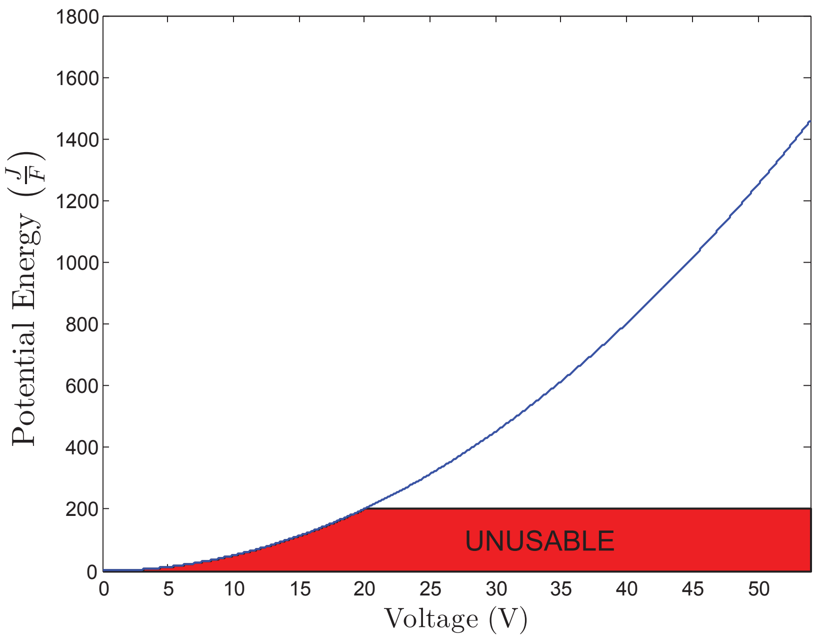


Figure 3.9: Ultracapacitor Bank - Usable Capacity vs Total Capacity

Equations

$$v_{C_u} = i_u R_{lu} + L_u \frac{di_u}{dt} + q_{uh} v_{C_h} \tag{3.12}$$

and

$$C_u \frac{dv_{C_u}}{dt} = -i_u \tag{3.13}$$

govern the behavior of the system. The values of the passive components used in simulation and hardware are $L_u = 1.1$ mF, $R_{lu} = 0.4\ \Omega$, and $C_u = 150$ F.

3.3.4.3 Overvoltage Discharge (OVD) Phase

While it is desirable to store as much energy as possible for future use, there are finite limits to the amount that can be contained in the storage elements. For such situations where the energy storage elements are at capacity, one of the inverter phases is devoted to dissipating excess energy. A set of power resistors provides the ability to dissipate a noticeable amount of power, while an inductor included in series prevents overly-high peak currents.

The particular resistance of the bank was chosen as a pair of 5.6 Ω resistors in series with a 600 μH inductor. The nominal power rating per resistor is 250 W, and the maximum permissible load power imposed upon the pair is 400 W. The operation of the OVD phase within the whole stabilizer system is described by

$$0 = i_o(R_o + R_{lo}) + L_u \frac{di_o}{dt} + q_{oh} v_{C_h}. \tag{3.14}$$

Note that, given how the current polarities indicated in Figure 3.8, the value of i_o will always be non-positive.

3.3.5 Bus Ballast

Given that some of the units attached to the microgrid are in the form of boost converters, it is useful to have a minimum load present on the grid at all times: a boost converter operating with no load is unstable and can destroy itself. A fixed RC load is imposed upon the bus. In order to better demonstrate the effects of the stabilizer unit, the random source's target output power is offset by the nominal power draw of the ballast. This maintains the interplay between the random source, random load, and stabilizer while providing a baseline power draw that the grid can work on top of. This also provides a measure of protection should the system happen upon a situation that causes the random load to drop out.

In addition to a 50 Ω resistance[4] that comprises the bulk of the load ballast, a modest capacitance (20 μF) is added for noise suppression and a small increase in inherent stability. Inter-device current flow and the bus voltage is determined by

$$C_{bus}\frac{dv_{C_{bus}}}{dt} = i_{source} + i_{stabilizer} - i_{load} - \frac{v_{C_{bus}}}{R_{bus}}. \tag{3.15}$$

[4]For a higher combined power rating, two 100 Ω, 250 W resistors are connected in parallel to form the bus resistance

3.4 Stabilization Algorithms

3.4.1 Traditional PI

The traditional PI control equations can be found in any of a multitude of textbooks on classical control theory (such as [33], for example), and are not repeated here.

The specific form implemented is that of two cascaded PI controllers, with one control pair for each phase. The two controllers are arranged so that the outer loop tracks the error between the nominal bus voltage and actual bus voltage. It outputs a current reference to the inner loop, which tracks the difference between that and the actual phase current and outputs a duty cycle command to the stabilizer. This also makes the higher-level (outer loop) functionality of voltage tracking more resistant to errors and disturbances in the lower-level implementation and control of the stabilizer unit (as demonstrated by Vilanova and Arrieta [34]).

Saturation non-linearities are implemented in all of the PI controls. Duty cycle outputs for the battery and ultracapacitor banks are limited to a maximum of 80% on the low-side switch. Current reference outputs are limited based on the specific device attached to the phase. With a nominal capacity of 10 Ah, the battery phase is limited to ± 5 A. The ultracapacitor current is limited[5] to ± 11 A. The OVD phase is limited to a 6 A maximum current sink.

The two storage phases are both controlled by PI systems, but having two completely duplicate controllers does a disservice to the stated goal of having improved treatment of the battery system. To reduce the strain on the battery system, the PI controls on the outer loop were adjusted so that the gains affecting the ultracapacitor phase are several times greater than those of the battery phase. This has the effect of causing the ultracapacitor bank to source and sink more current in response to variations in the load and source. The inner-loop gains are all identical, and this similarity persists between the PI and FL systems as well.

The OVD phase was configured to dissipate energy based on the SoC of both storage phases. The phase voltages are compared to nominal "high SoC" thresholds (80% SoC), scaled, saturated, and added together before being used as the input to a PI control block. The scaling and saturation is adjusted so that each storage phase can utilize up to half of the OVD phase's energy dissipation capacity (3 A current sink) at 100% SoC. The saturation limits for current are set to $[-1, 3]$ for each phase's contribution. (If the lower limit were at 0, the integral effect would persist even after the storage phase voltages decreased below the OVD thresholds.)

[5]This is predominantly a factor of the power inverter limitations, not those of the ultracapacitor bank.

3.4.2 Fuzzy Logic

3.4.2.1 Overall Operation

The design of the fuzzy logic system is based on expert human knowledge. The expert human defines the set of input and output membership functions that describe the system characteristics, as well as the rules that govern the desired system behavior. The primary goal of the FL control system functionality is similar to that of the PI system: sink or source energy to counteract variations in bus voltage. The secondary goals of the system are to reduce battery strain, provide self-protecting functionality, and to effect energy balancing. These are all accomplished through the fuzzy rules that drive the system.

One of the advantages a PI controller has over stateless fuzzy logic systems is the concept of memory. The integral term provides information to the control system about the past conditions of the system. This is a key component in eliminating steady-state error. Without this information, an FL control system's performance will be more akin to that of a non-linear proportional control. It is for that reason that an integral term is added to this FL system in addition to standard device measurements.

There are four inputs to the FL controller: bus voltage error, integrated bus voltage error, battery voltage, and ultracapacitor voltage. It utilizes those inputs in a set of fuzzy rules to generate three current references as outputs: battery current, ultracapacitor current, and overvoltage discharge current. Both input and output values are normalized into ranges of either $[-1, 1]$ or $[0, 1]$; input values are conditioned into those ranges, while output values are scaled out from those ranges. This allows for ease of adjustment if the ratings of the storage elements are altered (e.g. doubling the capacity of the battery bank can allow for almost double the current output).

The general operation of the system is to have the ultracapacitor bank source and sink the majority of the current needed to stabilize the bus. The battery bank exerts noticeably less effort, except for under two general conditions: (a) the ultracapacitor bank is nearing its SoC limits and cannot perform the required actions, and (b) the battery bank is nearing its SoC limits and needs to be brought back into a state near the middle of its range. Additionally, if the two banks are at opposite ends of their ranges, they will transfer energy between them in an attempt to balance out. The OVD phase comes into use only when both banks are near full capacity and need to be discharged towards the middle of their SoC ranges.

3.4.2.2 Membership Functions

As mentioned previously, membership functions describe how much a given value "belongs" to a given fuzzy set. The function is commonly expressed as $\mu_i(x)$. The degree of membership a value can have is in the range [0,1], where 0 denotes no "belonging" at all, middle values indicate partial "belonging," and 1 denotes a complete "belonging" to that set. A given value can also have membership in multiple fuzzy sets; if Robert D. Smith weighs 99.1 kg, he might have membership of 0.29 in the fuzzy set *normal*, 0.71 in the set *overweight*, and 0.04 in the set *obese*. (In a more numerical description, the three results would be represented as $\mu_N(99.1) = 0.29$ and $\mu_{OW}(99.1) = 0.71$ and $\mu_{OB}(99.1) = 0.04$.)

A variety of shapes can be used to specify membership functions. Marshall, Kazerani, and Shatshat [35] performed an investigation into varying the shapes of membership functions to achieve certain optimizations in a HVDC system controlled via fuzzy logic. Their conclusions were that it was exceedingly unlikely for a single type of membership function shape to work well for every situation, and that the choice of membership functions should be chosen based on the primary system requirements. For the simplicity of computation and understanding, the membership functions in this research are described using triangles and trapezoids.

The membership functions for bus voltage error and integrated bus voltage error are given in Figures 3.10 and 3.11 respectively. Each input has only two membership functions covering the entire range: *negative* and *positive*. There is a noticeable amount of overlap between the two functions, which reduces the abruptness of transitions between the rules that include the membership functions. The values of the membership functions correspond to the proportion of the full error ranges. For example, if the possible range of values for bus voltage error is ±5 V, then an input value of 0.25 would correspond to an actual error value of 1.25 V and result in membership values of $\mu_N(0.25) = 0.250$ and $\mu_P(0.25) = 0.607$.

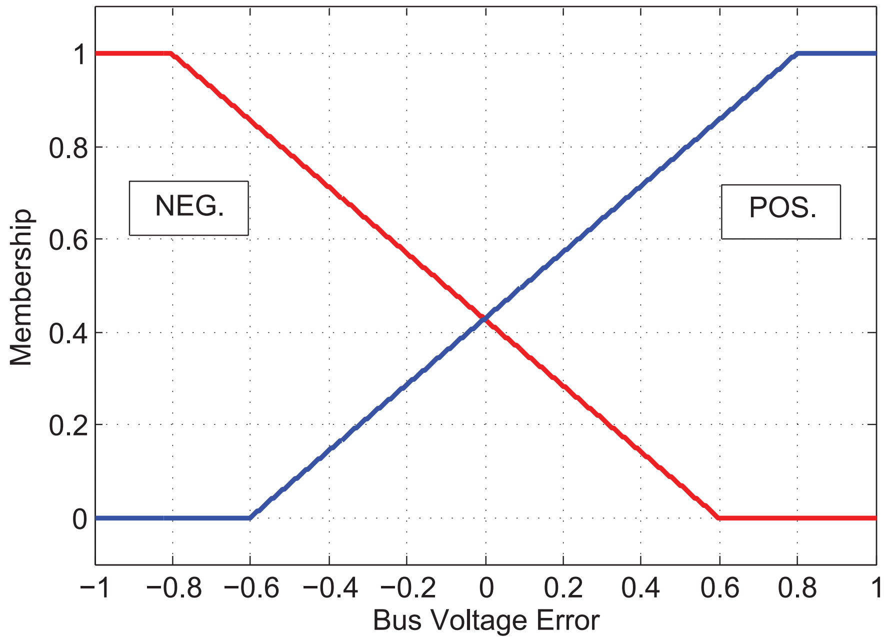


Figure 3.10: Membership Function for Bus Voltage Error

The membership functions for both the battery bank and ultracapacitor bank SoC are slightly unusual: there are only two functions, *low* and *high*. The system can still operate properly in the middle ranges not covered by the two functions by utilizing a transformation similar to a logical “not” operator. (This is explained later in 3.4.2.3.) The membership functions for the battery cover more of the SoC range than those of the ultracapacitors, as the ultracapacitors are more tolerant of large changes in SoC.

Figures 3.14 through 3.16 show the membership functions for the output currents on the stabilizer phases. For the storage phases, positive current denotes current into the converter. The inverse is true for the OVD current membership function. While there are five membership functions that cover the battery current, there are only four for the ultracapacitor bank. This is because of the addition of an extra fuzzy rule to provide an additional zero-current bias effect on the battery bank output under certain conditions. The units of all three membership functions are in terms of portion of rated current. For example, the center point of the battery current membership

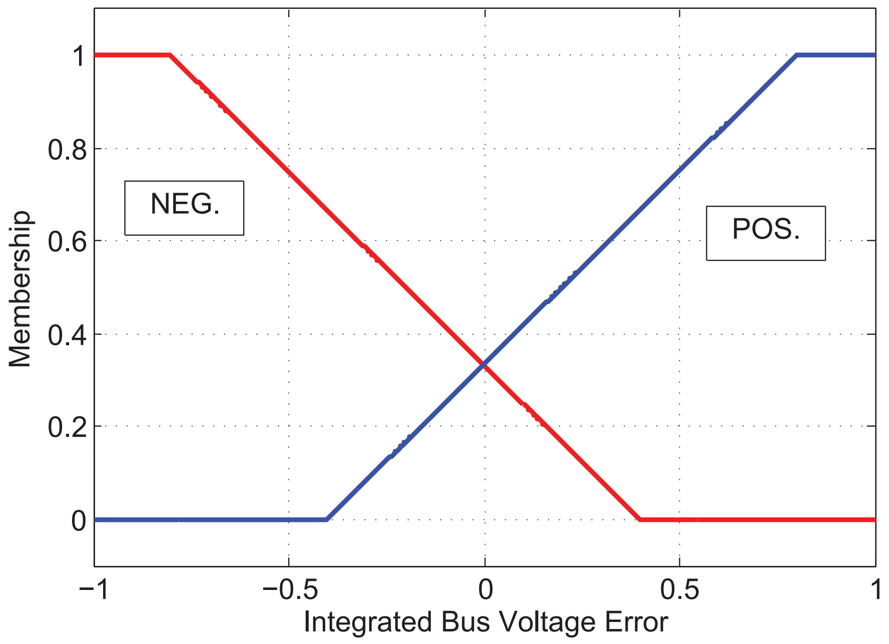


Figure 3.11: Membership Function for Integrated Bus Voltage Error

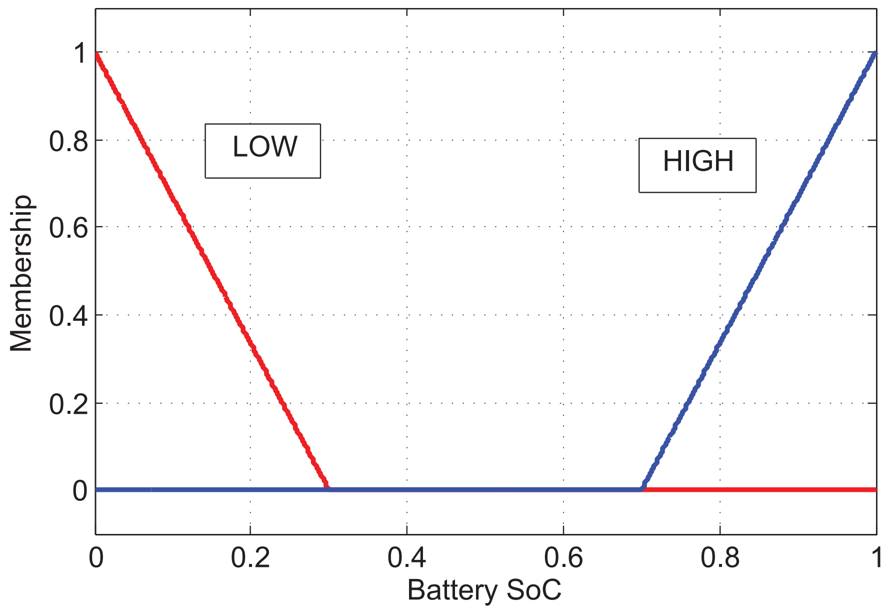


Figure 3.12: Membership Function for Battery Bank SoC

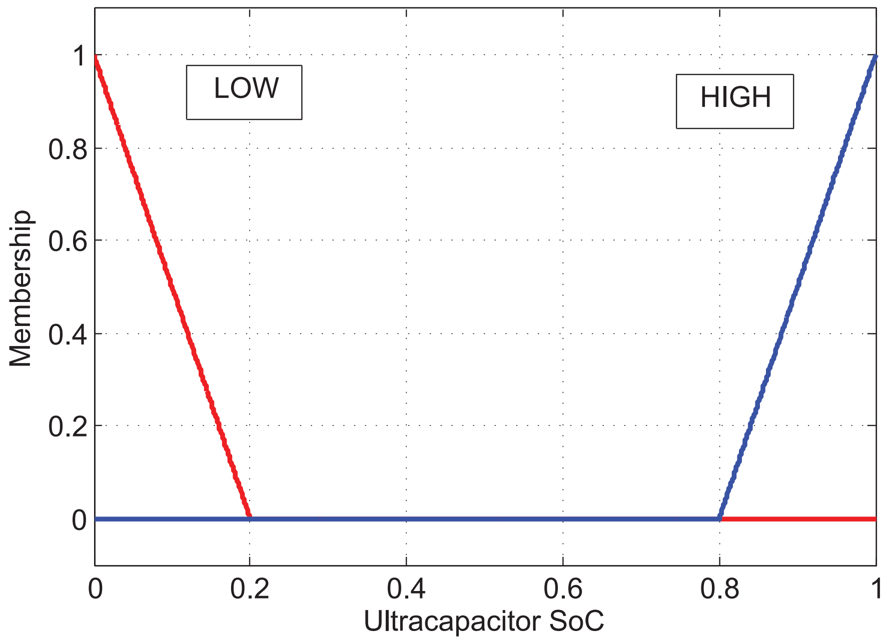


Figure 3.13: Membership Function for Ultracapacitor Bank SoC

function *negative* is the value -0.3. If the battery bank is rated for ±3.25 A, then the value corresponds to -0.975 A.

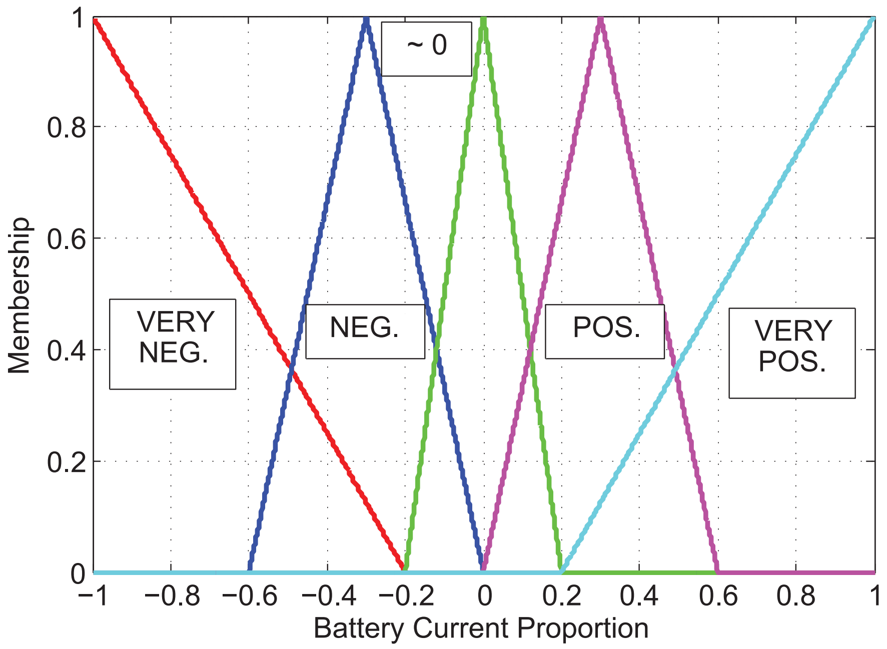


Figure 3.14: Membership Function for Battery Bank Current

The membership function *zero* is included along with the two other OVD current functions to provide MATLAB/Simulink with a bias towards zero current during times when the other membership functions are not in effect. If this were not the case, the default action would be to output the middle of the range of currents (meaning that the OVD phase would be drawing current when it wasn't supposed to). Additional fuzzy rules were added to the simulation to generate this functionality. The experimental implementation has neither the *zero* membership function nor the extra fuzzy rules to trigger it; a small amount of zero bias is added during the defuzzification process to cause this effect.

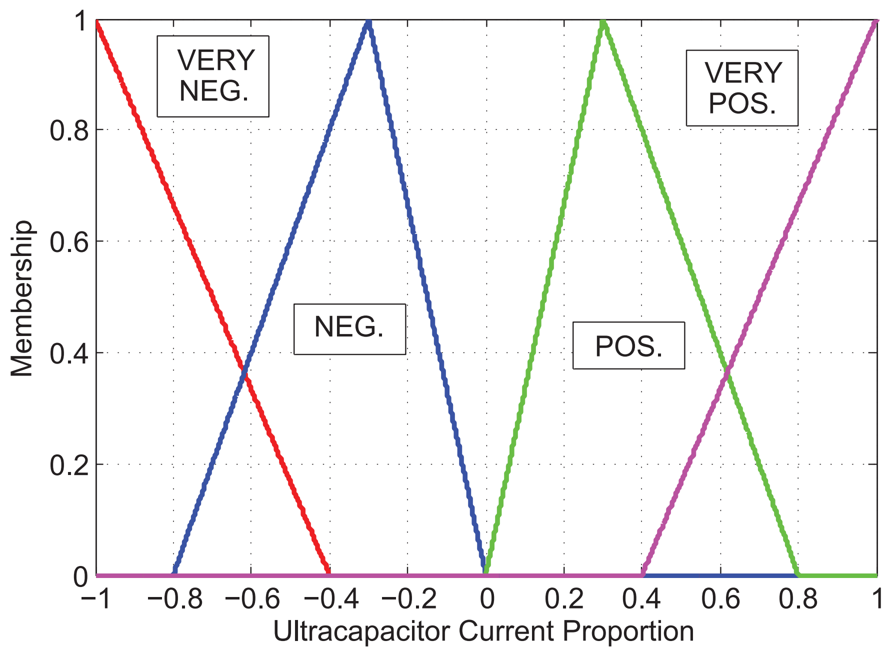


Figure 3.15: Membership Function for Ultracapacitor Bank Current

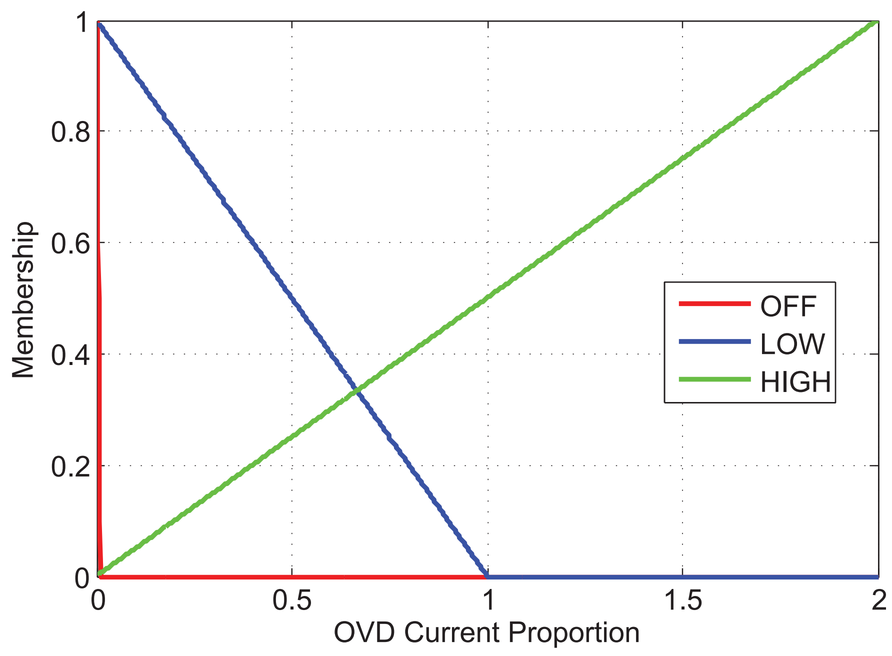


Figure 3.16: Membership Function for Overvoltage Discharge Current

3.4.2.3 Fuzzy Rules

For rules given here, the input and output values are notated as in Table 3.1. One piece of information to keep in mind is that positive values of bus voltage error mean that the actual voltage is lower than nominal, and negative values of bus voltage error mean that the actual voltage is above the nominal level.

Value Name	Term
Bus Voltage Error (V)	e
Integrated Bus Voltage Error (V sec)	$\int e$
Battery Voltage (V)	v_b
Ultracapacitor Voltage (V)	v_u
Battery Current (A)	i_b
Ultracapacitor Current (A)	i_u
Overvoltage Discharge Current (A)	i_o

Table 3.1
Input/Output Value Notation

The fuzzy rules themselves are formed similar to "if-then" statements. The logical operator "and" has a multiplicative effect to determine the net triggering weight of the rule. Additionally, for each membership function $\mu_i(x)$, the logical operator "not" has the effect of computing $1 - \mu_i(x)$. As an example, the rule "if X is *high* and Y is not *low*, then G is *very negative*" is a linguistic description of the equation $\mu_{VN}(G) = \mu_H(X) \cdot (1 - \mu_L(Y))$. (The process of solving for an explicit value of G is called defuzzification, and is described later in 3.4.2.4.)

Rules 1 through 6 describe the battery bank output with respect to bus voltage stabilization. The inclusion of criteria regarding the ultracapacitor bank SoC lends the tendency for the battery to not exert influence when the ultracapacitor bank can handle stabilization efforts by itself. Rules 7 through 10 are for the ultracapacitor bank current with respect to bus stabilization. While the rules include criteria for self-protection like the rules for the battery current, they do not include any terms for limiting the ultracapacitor current on the basis of the battery bank SoC. The rules for the OVD phase current are items 11 through 16. As mentioned earlier, the rules relating to the *off* condition are utilized in the simulation, but not in the experimental implementation. The last four rules (17 through 20) govern the transfer of energy between the storage banks if they are at opposite ends of their SoC ranges. These rules are implemented simultaneously alongside the rules for bus stabilization. The rules for energy transfer do not invoke the *very negative* or *very positive* output current membership functions, so that the action of bus stabilization will, in a way, have priority over that of SoC balancing.

1. if v_b is not *high* and v_u is *high* and e is *negative* and $\int e$ is *negative*, then i_b is *very negative*

2. if v_b is not *high* and v_u is not *low* and e is *negative*, then i_b is *negative*

3. if v_b is not *low* and v_u is not *high* and e is *positive*, then i_b is *positive*

4. if v_b is not *low* and v_u is *low* and e is *positive* and $\int e$ is *negative*, then i_b is *very positive*

5. if v_u is not *low* and e is *positive*, then i_b is *about zero*

6. if v_u is not *high* and e is *negative*, then i_b is *about zero*

7. if v_u is not *high* and e is *negative* and $\int e$ is *negative*, then i_u is *very negative*

8. if v_u is not *high* and e is *negative*, then i_u is *negative*

9. if v_u is not *low* and e is *positive*, then i_u is *positive*

10. if v_u is not *low* and e is *positive* and $\int e$ is *positive*, then i_u is *very positive*

11. if v_b is *high* and v_u is *high* and e is *negative*, then i_o is *low*

12. if v_b is *high* and v_u is *high* and e is *negative* and $\int e$ is *negative*, then i_o is *high*

13. if v_b is not *high*, then i_o is *off*

14. if v_u is not *high*, then i_o is *off*

15. if e is not *positive*, then i_o is *off*

16. if $\int e$ is not *positive*, then i_o is *off*

17. if v_b is *high* and v_u is *low*, then i_b is *positive*

18. if v_b is *high* and v_u is *low*, then i_u is *negative*

19. if v_b is *low* and v_u is *high*, then i_b is *negative*

20. if v_b is *low* and v_u is *high*, then i_u is *positive*

3.4.2.4 Defuzzification

The defuzzification process starts with the aggregation of the outputs of the fuzzy rules. In this implementation, the aggregation is performed by taking the sum of the output trigger weights for each membership function across all the rules that influence said function.

The exact implementations in the simulation and experimental systems are different, but are closely related and based on the Center of Area algorithm. In the simulation, each membership function is scaled by the sum of the triggering weights, then the union of the membership functions is taken, and the center of that area is computed (via numeric integration). The location of the center on the output current axis is the system reference output.

In the experimental system, a reduction on the Center of Area algorithm is used. First all the trigger weights for a given output membership function are added together. Next, the function is truncated at a height equal to the accumulated trigger weights. (This process imposes a maximum value of 1 for the trigger summation.) The center of area for the truncated function is computed, then all the centers are averaged together.

Because all the output membership functions are triangles, a compact algebraic equation exists to find the centroid of a trapezoid based on known characteristics of the shape. The vertical center of a truncated triangle, given its height h and a maximum height of 1 (full triangle), can be calculated through

$$\bar{y} = \frac{h}{3}\,\frac{3-2h}{2-h}. \tag{3.16}$$

Computing the horizontal center of a truncated triangle requires a more complex algebraic expression, but those expressions can be reduced by the inclusion of the vertical center term and the base b of the triangle. Equation

$$\bar{x}_l = \frac{b}{2}\,(1-\bar{y}) \tag{3.17}$$

gives the horizontal coordinate of the center of area for a truncated right triangle with the vertical leg on the left (towards the origin). The reverse type (vertical leg on right, away from the origin) of truncated right triange has

$$\bar{x}_r = \frac{b}{2}\,(1+\bar{y}) \tag{3.18}$$

as the horizontal coordinate of the center of area. A truncated irregular triangle's horizontal coordinate also requires the lengths of the full triangle's diagonal sides (d_L and d_R for the left and right diagonals), and is given by

$$\bar{x}_u = \frac{b}{2} + \bar{y}\,\frac{d_L^2 - d_R^2}{2b}. \tag{3.19}$$

Given the centers for all of the membership functions for a specific output equation, the weighted average of those centers provides the singular output value as per

$$x = \frac{\sum_{\forall i} \bar{x}_i \bar{y}_i}{\sum_{\forall i} \bar{y}_i}. \tag{3.20}$$

As mentioned previously in 3.4.2.2, a small amount of zero bias ($\beta > 0$) is added to the output equations to prevent divide-by-zero errors in the experimental implementation. (The simulation software default is to take the center of the range in the presence of zero-area output functions, which for the storage elements is always zero.) This alters (3.20) to form

$$x = \frac{\sum_{\forall i} \bar{x}_i \bar{y}_i}{\left(\sum_{\forall i} \bar{y}_i\right) + \beta}. \tag{3.21}$$

For very small values of β, this bias has negligible effect on the output if other rules trigger a membership function.

3.4.2.5 Exerting Control

Like the cascaded PI controller above, there is an inner PI loop to track the phase current to a reference value. The difference is in where the reference values come from. In the classical PI controller, they come from the outer PI control loops. With the fuzzy controller, the values come from the defuzzified outputs of the FL control system.

3.5 Remarks

The methodology given above describes both the physical configuration of the micro-grid system under consideration and the details of the control systems that will be implemented and compared in this research. The first step in analyzing the system is to construct a mathematical model of the system and simulate it. This is detailed in the next chapter.

Chapter 4

Simulation

With the methodology established previously, the next step is to emulate the system as a computational simulation.

4.1 Microgrid Models

4.1.1 General Approach

With MATLAB/Simulink being the primary software package chosen for use with this research work, the most straightforward approach was to create a model of the system based on the differential equations that describe the electrical circuitry in the system. The graphical programming of Simulink was utilized to construct and connect the pieces of the microgrid system and control.

For subsystems shown here, their masks include dialogues to set the various parameters of the system.

4.1.2 Ballast and Bus

The ballast load (mentioned earlier in 3.3.5) forms the basis for calculating the bus voltage level. The individual device currents are summed, and the current into the ballast load is calculated to determine the voltage imposed upon the ballast (and therefore the bus). Figure 4.1 shows the outer block that contains the ballast model, and Figure 4.2 shows the inner ballast system.

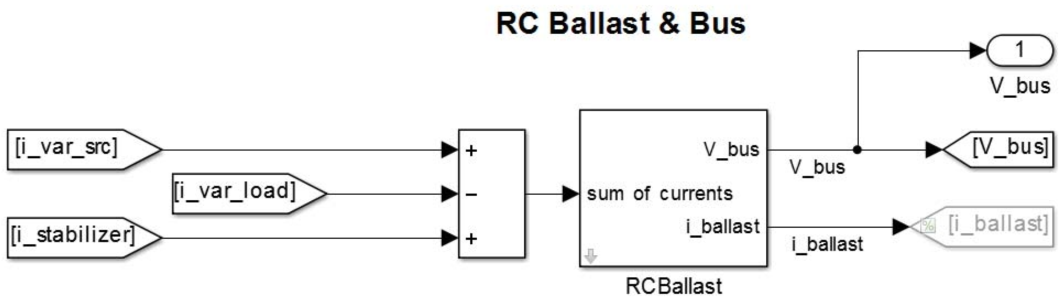


Figure 4.1: Bus Ballast - Outer Mask & Interface

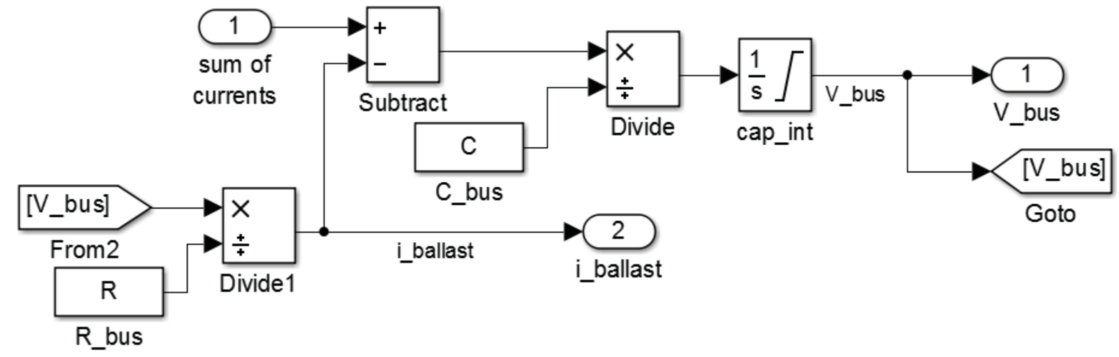


Figure 4.2: Bus Ballast - Subsystem Contents

4.1.3 Random Load

For simulation purposes, the random load was implemented directly as a random resistance (calculated from a random power draw at nominal voltage). Figure 4.3 shows the Simulink block diagram that implements the random load.

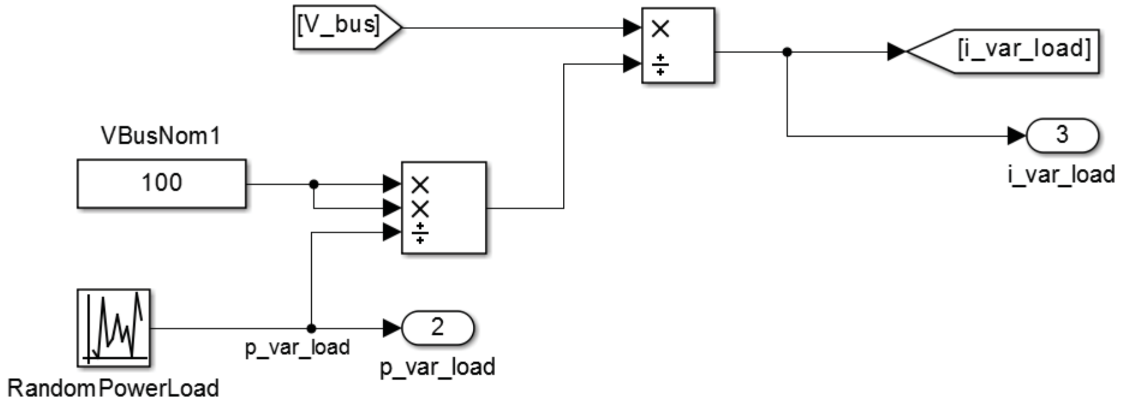


Figure 4.3: Random Load - Simulation Implementation

The random power load is a uniform random number in the interval $P_{nom} \in [50, 305]$ W. That number is converted to a nominal resistance value which is then imposed upon the bus. The "Random Number" block generates numbers at a fixed sample rate (in this case, 3 seconds per sample). The output is held for the entire period until a new sample is generated.

4.1.4 Random Source

The random source is modelled as a boost converter. Figure 4.4 shows the subsystem mask of the unit with the relevant interfaces. The detailed contents of the subsystem are given as Figure B.1 in Appendix B. Because the source is intended only to produce power and not consume any externally-sourced power, a saturation nonlinearity was imposed upon the inductor, limiting the inductor to uni-directional operation.

The source acts by taking in a switch signal (which can be either a full switching-mode signal or an average duty cycle) and using it with the low-side input voltage to boost current up to the bus voltage (which is an input as well, to solve KCL for the high-side filter capacitance). Both the low and high-side currents are tracked, the latter being used in the bus ballast equation to determine the bus voltage.

4.1.5 Stabilizer

Figure 4.5 shows the subsystem mask of the stabilizer unit and its interfaces. Figures B.2 through B.6 in Appendix B show the full details of the stabilizer model underneath the mask.

Each phase operates independently, taking in the duty cycle (or a full switching-mode signal, if desired) to control the phase switches. For the storage elements, the duty cycle is the low-side switch; for the discharge phase, the opposite is true. The block

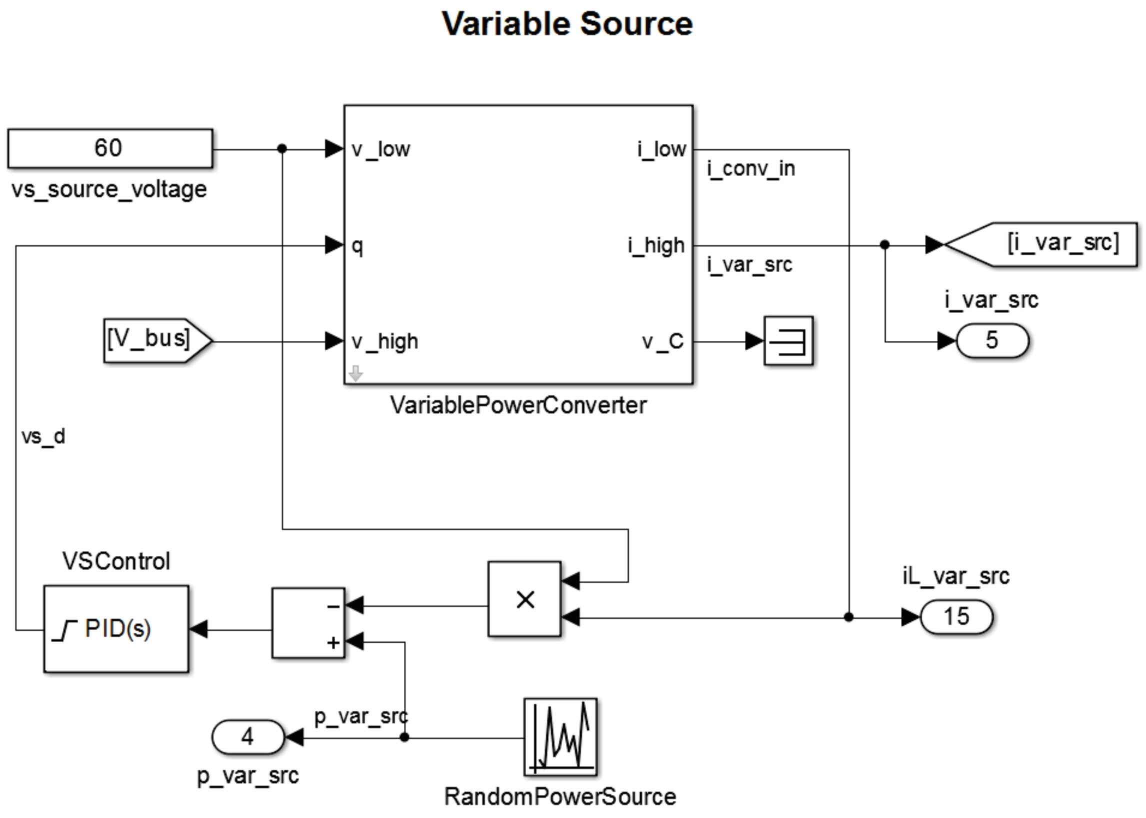


Figure 4.4: Random Source - Simulation Implementation

uses the bus voltage to solve KCL for the phases and high-side filter capacitance. The high-side current is used in the ballast to determinte the bus voltage. Additionally, the phase currents and voltages are tracked.

4.1.6 Control and Interfacing

Both types of control systems utilize the same inner-loop current-tracking PI controls with the same gains. This helps isolate the differences in performance to that of the outer-loop PI and FL control systems. The inner-loop gains were chosen to provide

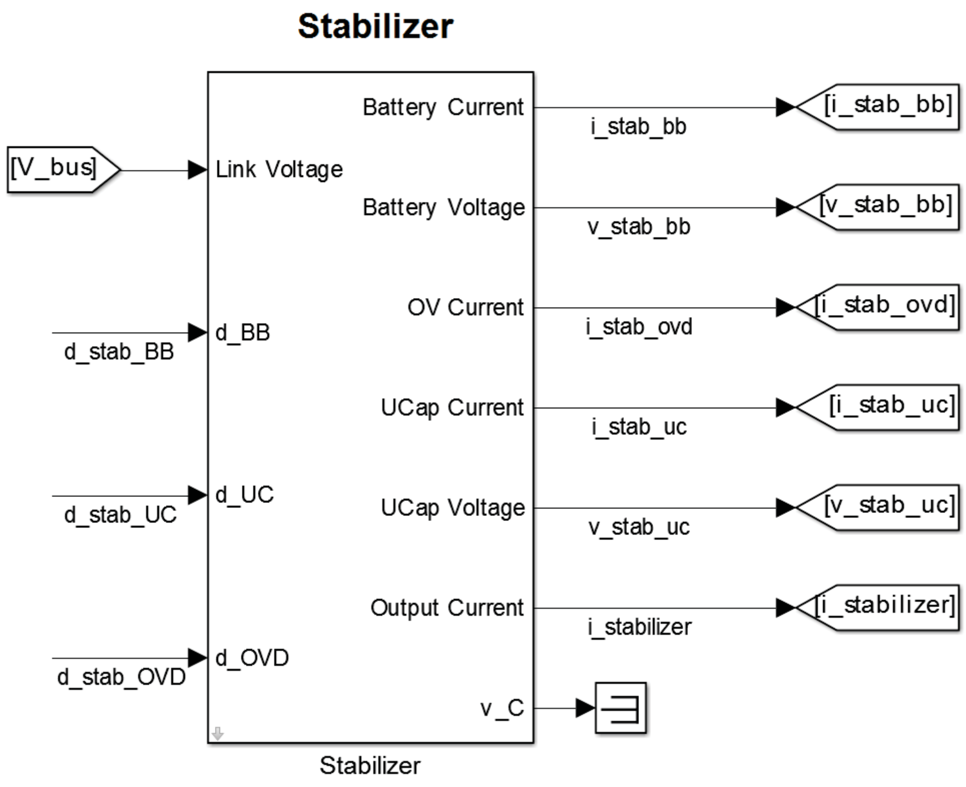


Figure 4.5: Grid Stabilizer - Simulation Implementation

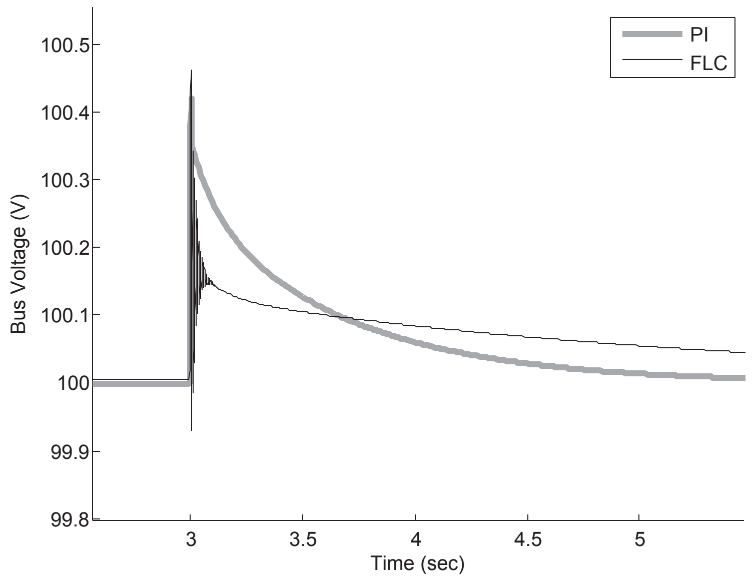


Figure 4.6: Control System Comparison

a reasonably-rapid response to the current reference with minimal overshoot.

The issue of gains and control tuning is ever-present. Control systems are designed to meet “acceptable” performance, with the definition of “acceptable” varying based on the situation. In this research, the purpose is to show that an FL system can regulate bus voltage on a level similar to that of a PI control. Therefore, the absolute responses of the systems are not as important as the relative differences in the systems. To that end, both control systems were adjusted to have similar settling times and overshoot. Figure 4.6 shows an example of the two control systems reacting to a decrease in load (increased resistance). Both systems have similar levels of overshoot, with the FL control producing a smaller overall deviation from nominal and the PI control reaching a steady-state level more quickly.

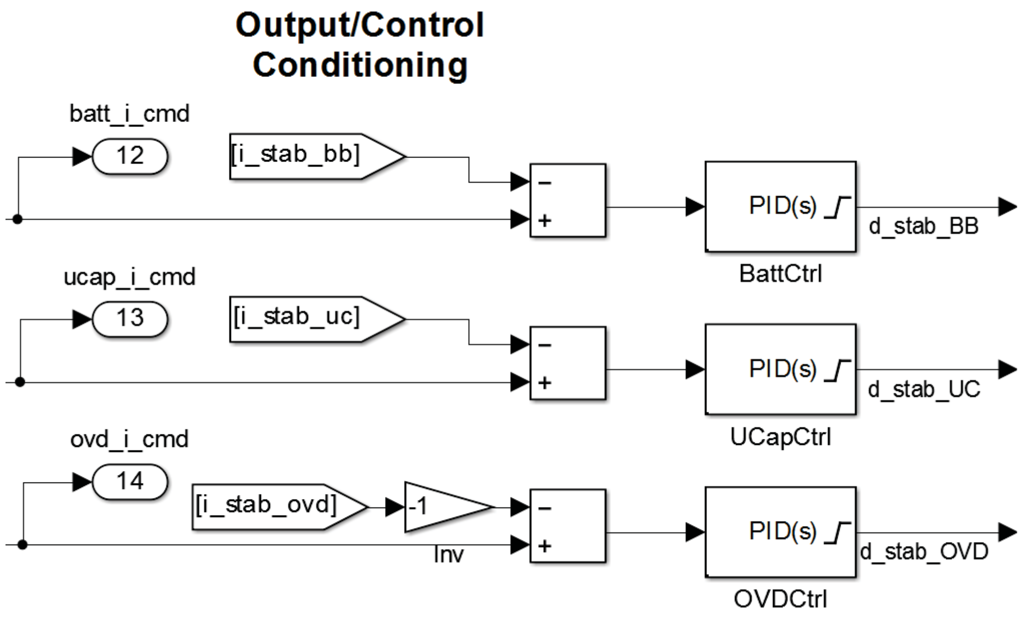


Figure 4.7: Inner Current Control Loop

The inner control loop common to both control systems is shown in Figure 4.7. PI blocks take in a current reference and output the duty cycle for a phase to track to that reference. The PI gains are $k_p = 0.1$ and $k_i = 0.01$ for all the phases, with duty cycle saturations of $[0.2, 0.8]$ for the battery phase, $[0.1, 0.9]$ for the ultracapacitor phase, and $[0, 1]$ for the OVD phase.

4.1.6.1 PI Control Interface

Figure B.7 in Appendix B shows the outer-loop interface for the PI control system. The storage phases use a common signal for bus voltage error, while the OVD phase uses a combination of storage phase voltage levels to generate a signal for that phase's outer PI control. The two storage voltages are scaled for equal representation of SoC levels.

4.1.6.2 FL Control System Interface

The outer control for the FL system is shown in Figure B.8 in Appendix B. Both the input and the output variables are scaled for use in the FL engine (which is normalized as described previously in 3.4.2.1). To generate the controlling fuzzy system, the MATLAB function "*fuzzy*" was used to access the Fuzzy Inference System editor. From that editor, the input membership functions, output membership functions, fuzzy rules, and engine configuration were entered to form one system.

When compared to the PI control system, the FL controller has an additional 1 kHz low-pass filter added to the input signals. This is to reduce the amount of ringing that can occur, as the FL control system outputs can produce higher short-term variation than a traditional PI controller.

4.2 Simulation Setup

One of the choices made during the development process was whether to implement the model in full switch-mode (where the individual toggling of the power switches was taken into account) or in average-mode (where the average duty-cycle was used instead). Because the simulations were intended to model the performance of the situation over a period of many minutes or even hours, it was determined to use averaging mode. A supporting reason for this was that, while there are significant differences in operation between the PI and FL control systems, the switching performances would likely be very similar.

One last item to note is the choice of numeric solver. The solver used in the simulations was a variable-step stiff solver ("ode23tb"). The benefit of variable-step solvers is that they reduce the step size when signals have high derivative values (which increases accuracy), while increasing the step size when signals vary slowly (speeding computation). The "stiffness" of a system is often difficult to quantify. One of the general descriptions of a stiff system is one who has widely-varying time constants present [36]. Closed-loop power converters are one example, as switching dynamics occur in the spectrum of tens of kilohertz, while output dynamics typically take place on the order of tens of hertz. Even though the simulation is modeled in average-mode, the load and source variations happen on the order of seconds, while the stabilization dynamics take place on the sub-second scale.

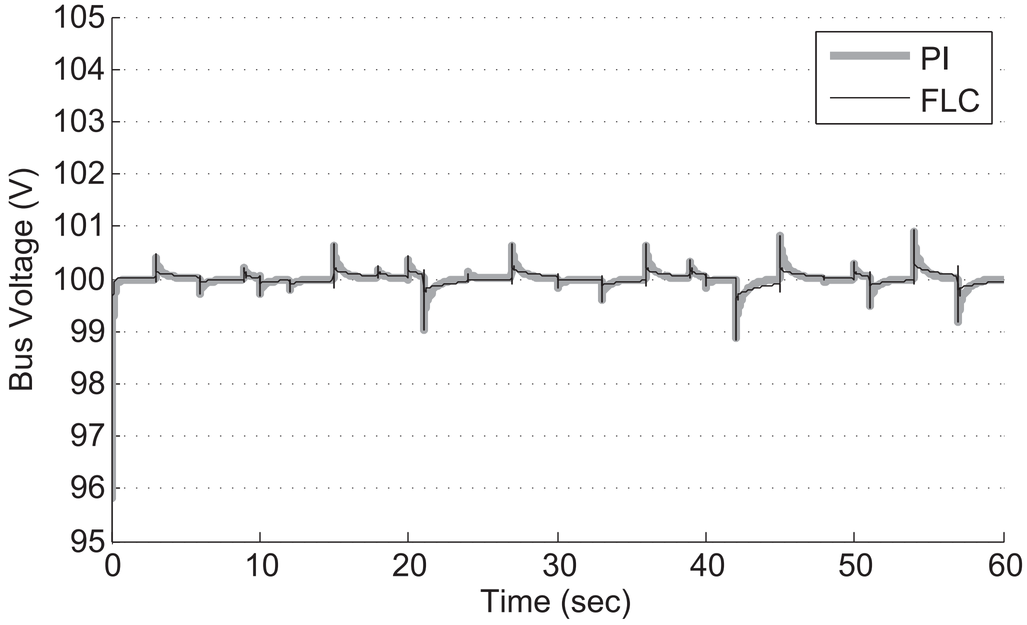


Figure 4.8: Bus Voltage Stabilization - Short-Term Simulation

4.3 Results

4.3.1 Short-Term Performance

The first consideration of the simulation is to verify whether the control systems stabilize the bus voltage to a nominal value in the presence of random source and load fluctuations. Given update rates of 10sec and 3sec for the source and load (respectively), a 60sec simulation was performed to observe this. Figure 4.8 shows the performance of both control systems. Both storage elements were initialized to 50% SoC.

Both control systems show a stabilizing effect on the bus voltage. The PI system exhibits better steady-state regulation, while FL control produces reduced peak deviation from the nominal voltage. Both systems provide tight regulation from the standpoint of percentage tolerance; each regulates the bus voltage to within about 1% of nominal voltage.

The systems utilize each of the storage elements in the process of regulating the bus voltage. Figure 4.9 shows the overall stabilizer current. The two control systems result in very similar input/output current from the stabilizer, with the FL system producing higher bursts of current at changes in the load or source.Figures 4.10 and 4.11 show the storage phases' currents. The OVD phase, for both control systems, had a zero reference command and no actual draw in this test. These results show that the FL control system seldom utilizes the battery and instead relies mostly on the ultracapacitor bank for stabilization.

4.3.2 Long-Term Performance

Several metrics are used to compare and contrast the performance differences between the two systems. The system states of particular interest are bus voltage, battery bank current and voltage, and ultracapacitor bank current and voltage, and OVD current. The bus voltage was subject to two calculations: numeric integrations of the deviation

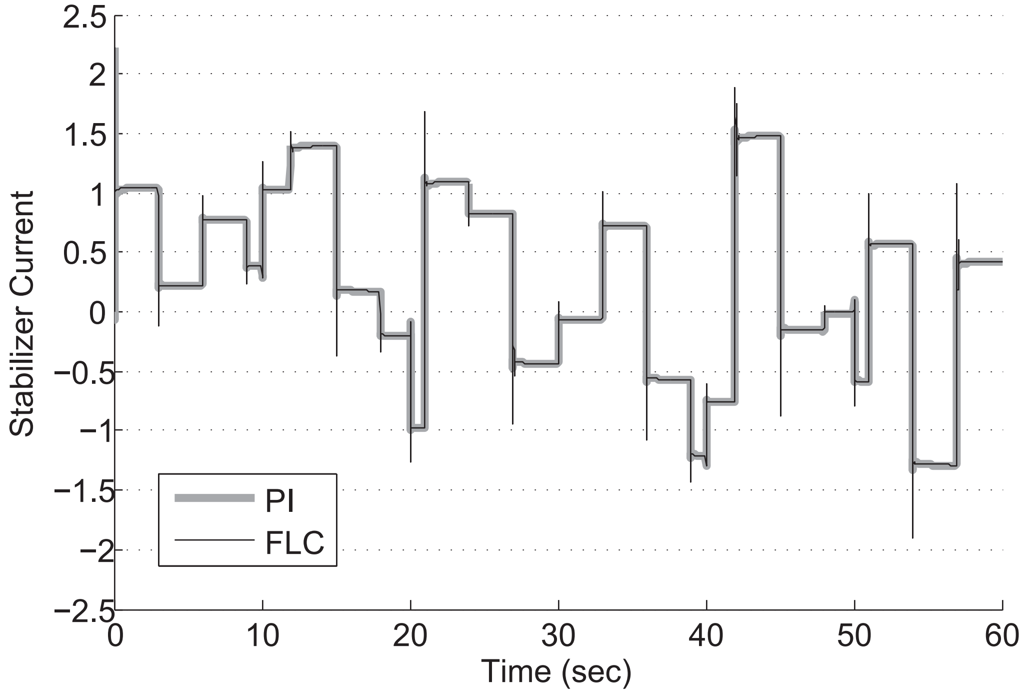


Figure 4.9: Stabilizer Bus Current - Short-Term Simulation

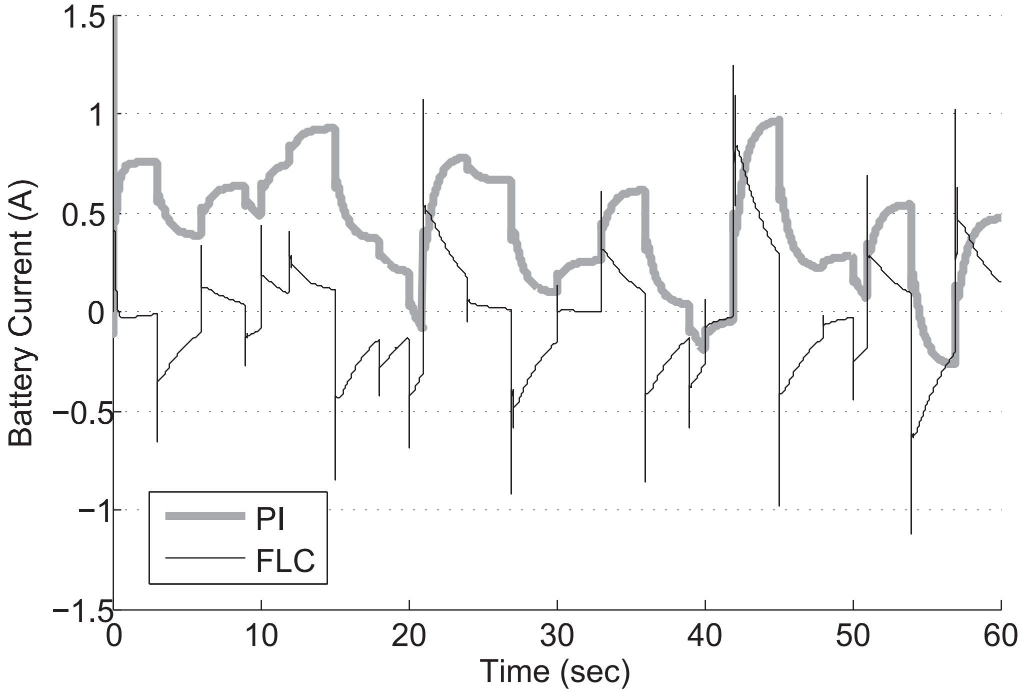


Figure 4.10: Battery Phase Current - Short-Term Simulation

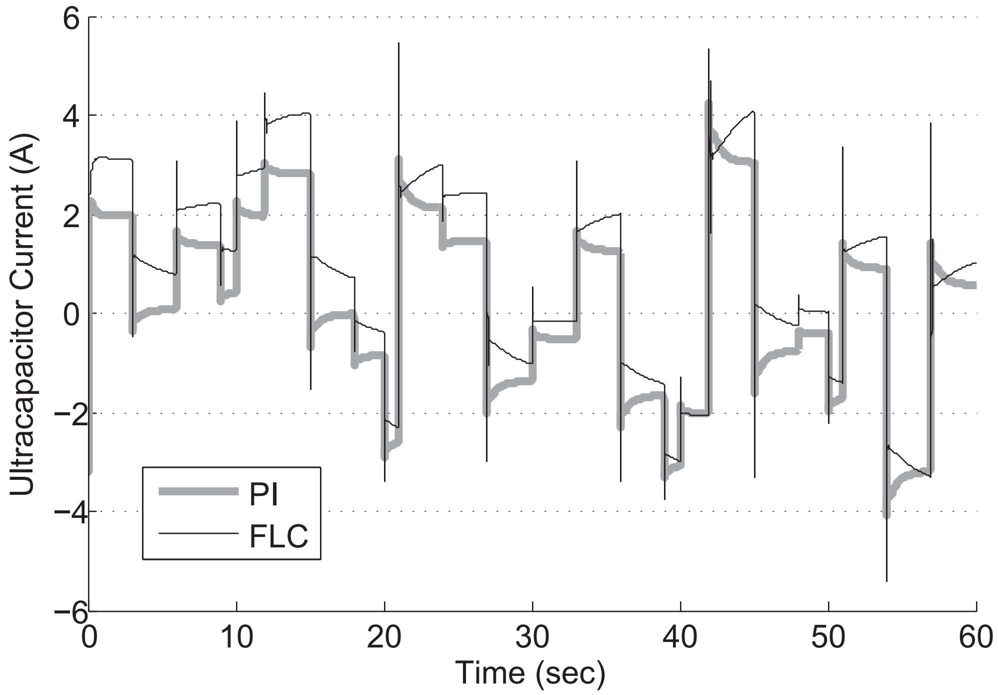


Figure 4.11: Ultracapacitor Phase Current - Short-Term Simulation

from nominal voltage and the square of that deviation. The former metric describes the overall ability of a control system to regulate the voltage, while the latter describes the tendency of a control system to control peaks in the deviations. The battery and ultracapacitor bank currents are also numerically integrated to describe the demand imposed upon the storage elements. The OVD current is integrated to show the amount of energy dissipated by the system. Finally, the storage bank voltages are measured to track changes in SoC.

Because of the heavy dependency on pseudo-random numbers in the operation of the simulations, a single simulation run is less descriptive of the average performance than an aggregation of several runs. To that end, each scenario was executed for 10

Table 4.1
Numerical Results - Normal Regulation Simulation

Parameter	PI	FL
$\int \lvert v_{err} \rvert dt$	42.404	44.643
$\int v_{err}^2 dt$	10.135	6.092
$\int \lvert i_{batt} \rvert dt$	200.89	130.05
$\int \lvert i_{ucap} \rvert dt$	1128.1	1260.6
$\int \lvert i_{ovd} \rvert dt$	0	0
$v_{batt-final}$	49.002	49.000
$v_{ucap-final}$	37.602	37.609

minutes of simulation time, and repeated 20 times (with the random number seed changing each time based on the system clock). The average values of each run's performance metrics were calculated and used as the basis for comparing control system performance.

4.3.2.1 Normal Regulation

The first simulation scenario was that of standard bus regulation. The storage banks were initialized to 50% SoC (49 V for the battery bank, 35 V for the ultracapacitor bank), and the stochastic source was configured to not include any significant surplus or deficit of power from the normal levels. Figure 4.12 shows the bus voltage regulation for the last simulation of the 20-run set. Table 4.1 shows the average numerical results of the systems' performances.

As with the shorter verification simulation (see 4.3.1), the PI control tends to provide slightly better total voltage error performance, while the FL system has reduced peak

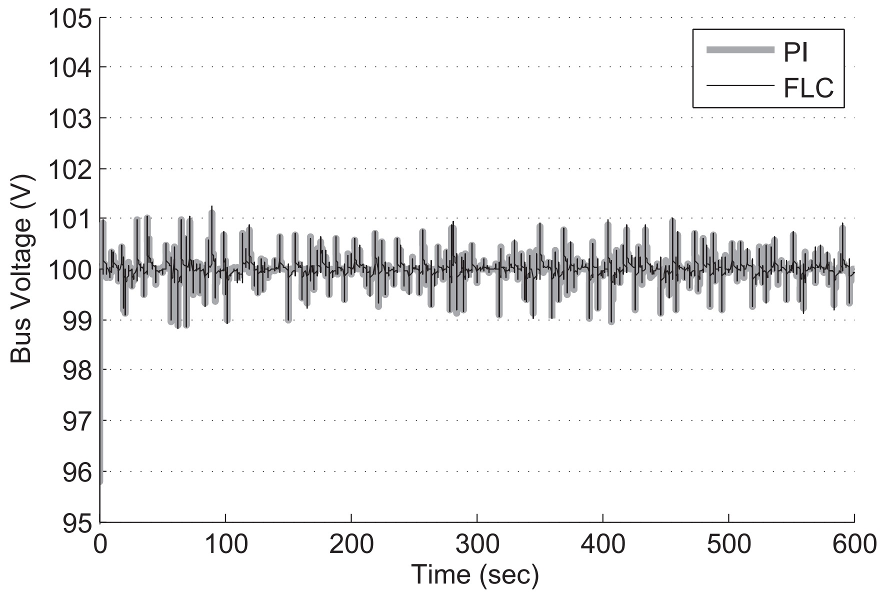


Figure 4.12: Bus Voltage - Normal Regulation Simulation

deviation from nominal voltage. Again, both systems regulate the voltage to within about 1% of the nominal level.

4.3.2.2 Depleted Ultracapacitor Bank

In this scenario, the battery bank is almost fully charged (95% SoC) but the ultracapacitor bank is depleted (5% SoC). Figure 4.13 shows the bus voltage regulation for the last simulation of the set. Figures 4.14 and 4.15 show the voltages for the battery and ultracapacitor banks respectively for the same simulation. Table 4.2 shows numerical averages of the systems' performances in this scenario.

Because of the particular configuration of the PI control system's OVD phase, some

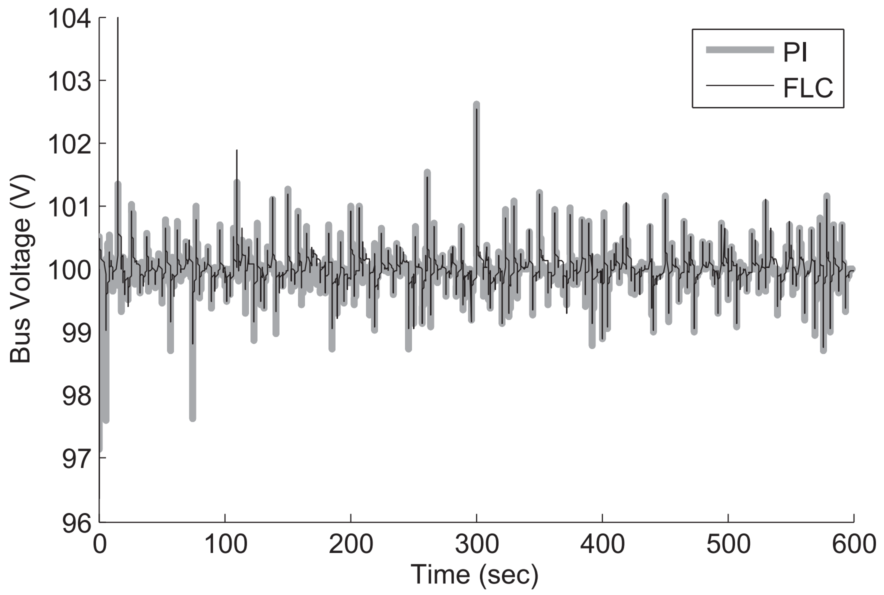


Figure 4.13: Bus Voltage - Depleted Ultracapacitor Bank Simulation

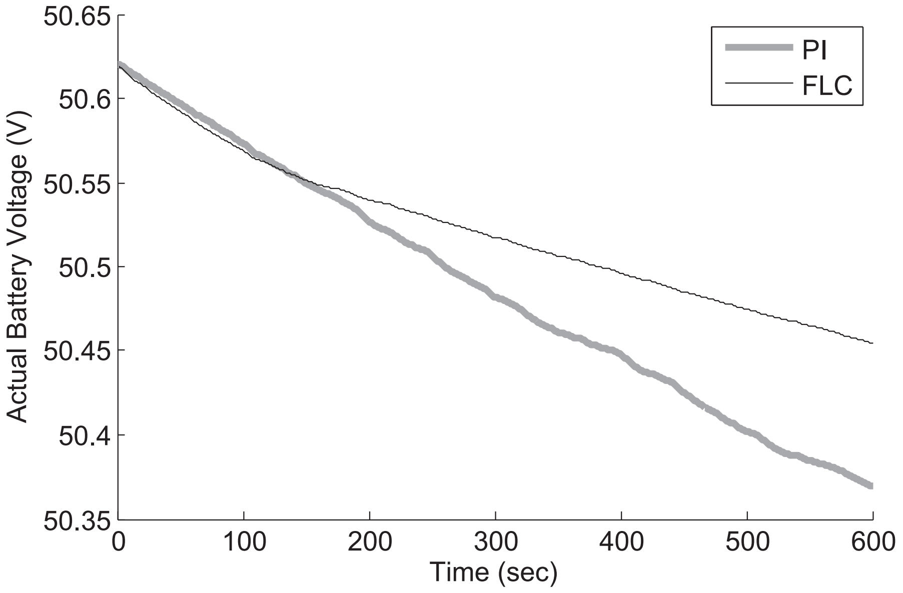


Figure 4.14: Battery Bank Voltage - Depleted Ultracapacitor Bank Simulation

energy is unnecessarily wasted. The FL control system, on the other hand, properly

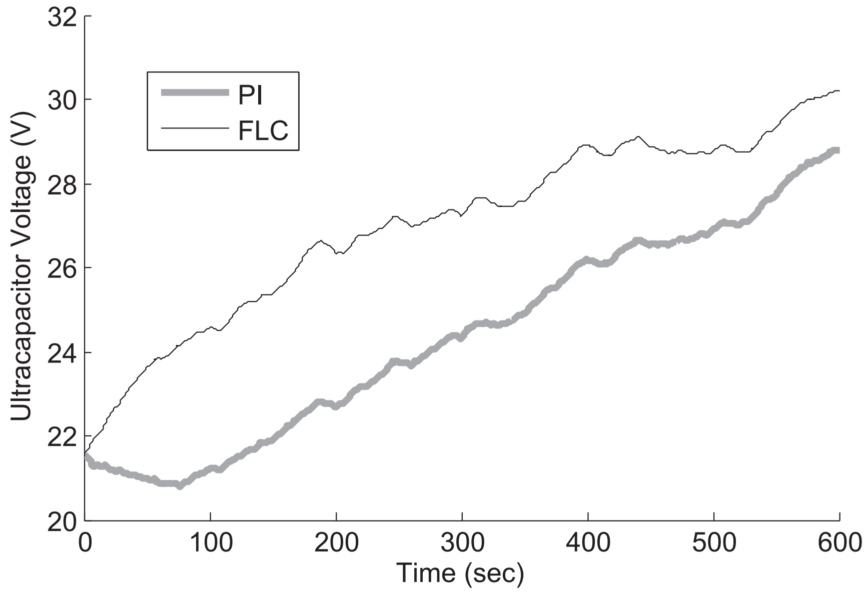


Figure 4.15: Ultracapacitor Bank Voltage- Depleted Ultracapacitor Bank Simulation

Table 4.2
Numerical Results - Depleted Ultracapacitor Bank Simulation

Parameter	PI	FL
$\int \lvert v_{err} \rvert$	62.068	68.509
$\int v_{err}^2$	22.043	15.488
$\int \lvert i_{batt} \rvert$	803.51	515.29
$\int \lvert i_{ucap} \rvert$	1859.1	1969.7
$\int \lvert i_{ovd} \rvert$	517.91	0
$v_{batt-final}$	50.352	50.448
$v_{ucap-final}$	29.675	30.571

transfers excess energy from the battery bank to the ultracapacitor bank. The improved apportioning by the FL system results in an extra 2.98% SoC increase on the ultracapacitor bank and a 2.67% SoC savings with the battery bank. Additionally, as shown in Figure 4.15, the FL control system increases the ultracapacitor bank's SoC more rapidly than that of the PI controller.

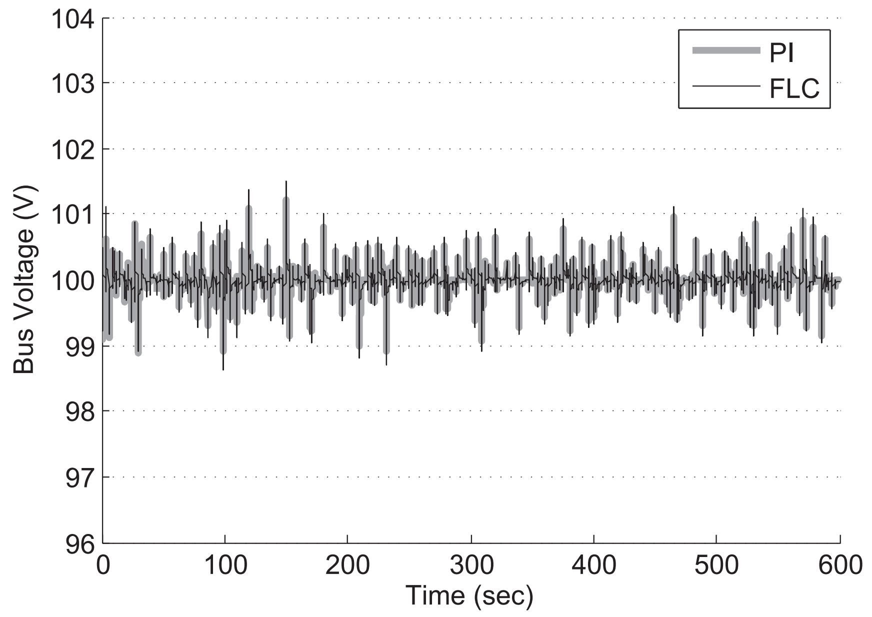


Figure 4.16: Bus Voltage - Depleted Battery Bank Simulation

4.3.2.3 Depleted Battery Bank

Though the FL control system is biased towards maintaining good battery SoC levels at the expense of the ultracapacitor bank's SoC, it is possible that the battery bank may be depleted at the same time the ultracapacitor bank is full. This scenario starts with the battery bank at 5% SoC and the ultracapacitor bank at 95% SoC. Figures 4.16 through 4.18 respectively show the bus voltage regulation, battery bank voltage, and ultracapacitor bank voltage of the final simulation in the set of 20 runs. Table 4.3 gives the average numerical results of the simulations.

As noted previously, the OVD control configuration of the PI system results in wasted energy. The two control systems provide comparable levels of voltage regulation. The

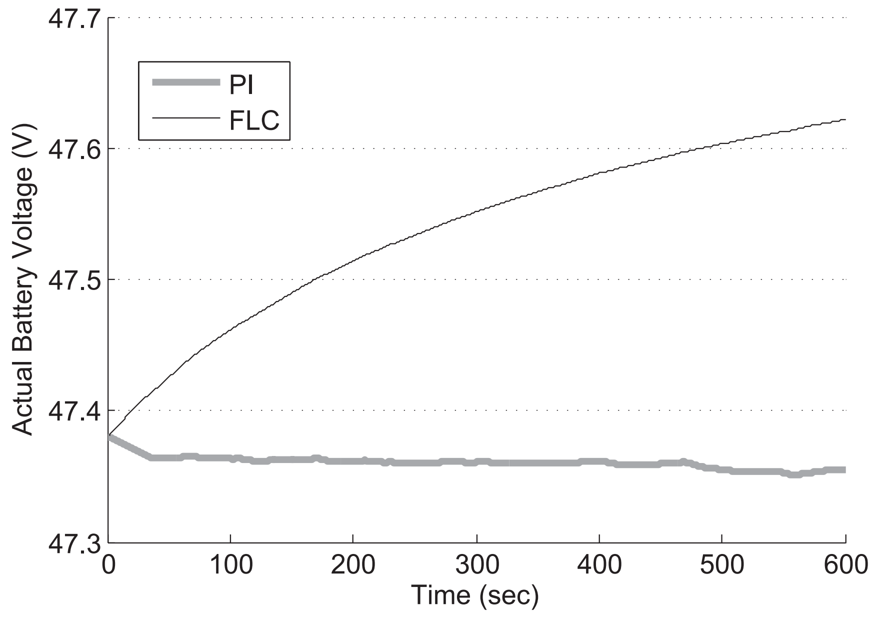


Figure 4.17: Battery Bank Voltage - Depleted Battery Bank Simulation

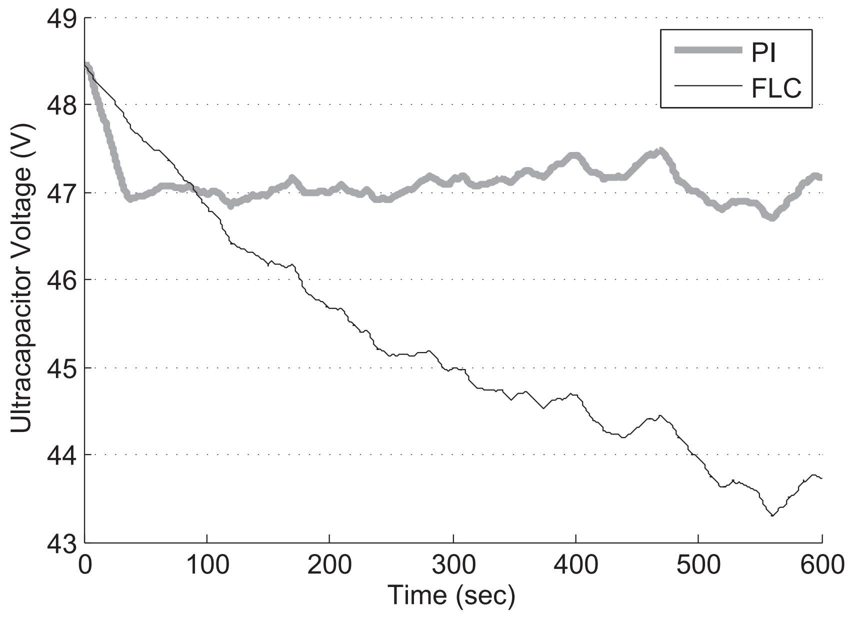


Figure 4.18: Ultracapacitor Bank Voltage - Depleted Battery Bank Simulation

Table 4.3
Numerical Results - Depleted Battery Bank Simulation

Parameter	PI	FL
$\int \lvert v_{err} \rvert$	37.513	42.026
$\int v_{err}^2$	7.156	5.722
$\int \lvert i_{batt} \rvert$	318.14	826.92
$\int \lvert i_{ucap} \rvert$	1183.6	1136.4
$\int \lvert i_{ovd} \rvert$	655.02	0
$v_{batt-final}$	47.308	47.656
$v_{ucap-final}$	47.209	44.458

FL control system increases the battery bank SoC by an additional 9.67% without extra current demand from the ultracapacitor bank (which falls an extra 9.17% in SoC when compared to the PI system). The battery bank voltage (Figure 4.17), under FL control, is seen approaching the upper limit for the fuzzy membership set *low*, slowing the transisiton as it approaches that limit. The ultracapacitor bank voltage is shown to continue transferring energy to the battery bank.

4.3.2.4 Energy Deficit

Another possible scenario is that where the storage elements are depleted and a deficit of power exists on the microgrid. This scenario highlights the self-protecting functionality of the FL control system. Figures 4.19 through 4.21 show the final simulation's bus voltage, actual and measured battery voltage, and ultracapacitor voltage respectively.

This simulation shows the breakdown of the FL controller's ability to regulate bus

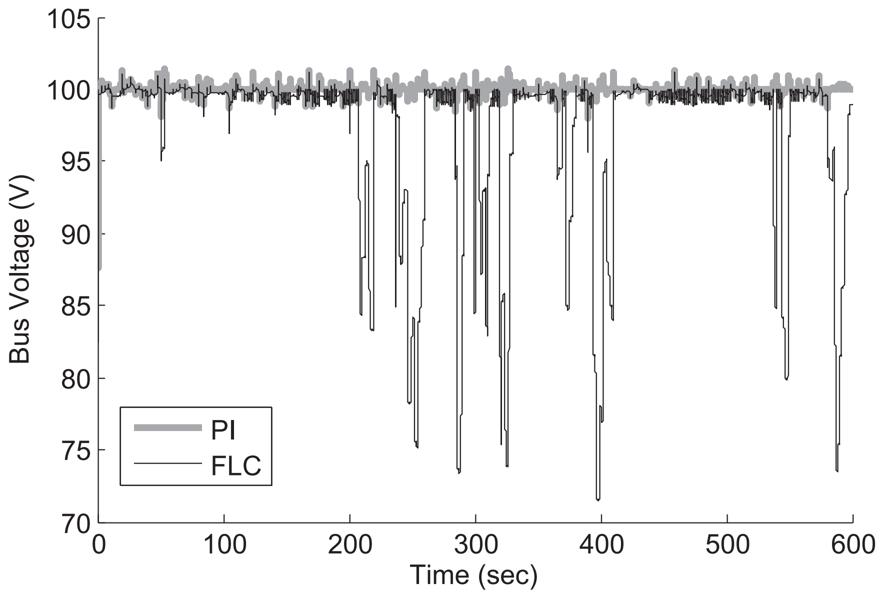


Figure 4.19: Bus Voltage - Energy Deficit Simulation

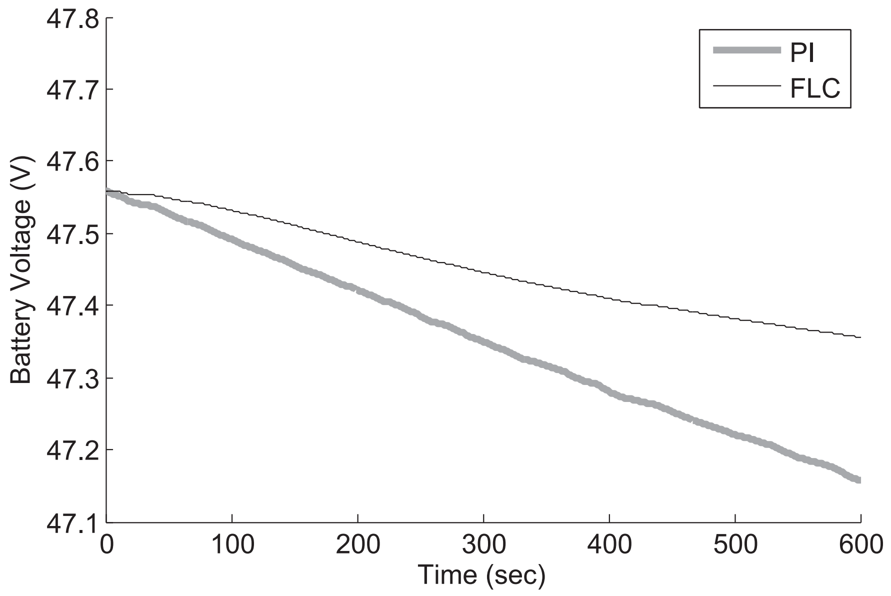


Figure 4.20: Battery Bank Voltage - Energy Deficit Simulation

voltage. However, this reduction in regulation ability results in the benefit of preserving the battery bank SoC. The FL control system results in the battery bank maintaining an extra 4.17% SoC vs. the PI control system (see again Figure 4.20). It

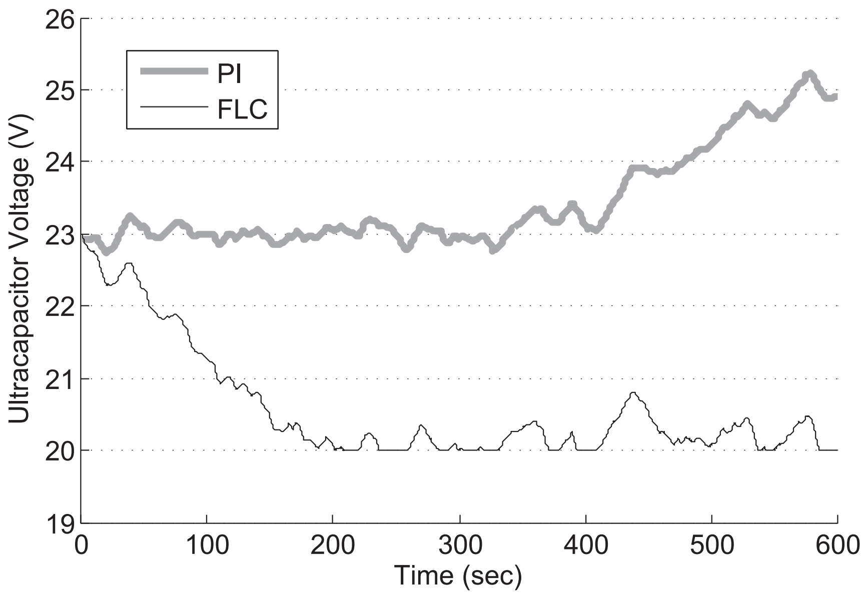


Figure 4.21: Ultracapacitor Voltage - Energy Deficit Simulation

Parameter	PI	FL
$\int \lvert v_{err} \rvert$	63.968	1376.7
$\int v_{err}^2$	29.444	20851
$\int \lvert i_{batt} \rvert$	968.1	577.58
$\int \lvert i_{ucap} \rvert$	1674.4	2037.2
$\int \lvert i_{ovd} \rvert$	0	0
$v_{batt-final}$	47.238	47.368
$v_{ucap-final}$	23.871	20.132

Table 4.4
Numerical Results - Energy Deficit Simulation

is also evident that the FL system utilizes the ultracapacitor bank as much as possible in an effort to avoid depleting the battery bank. The depletion of the ultracapacitor bank corresponds with the start of the breakdown in bus voltage regulation by the FL control system.

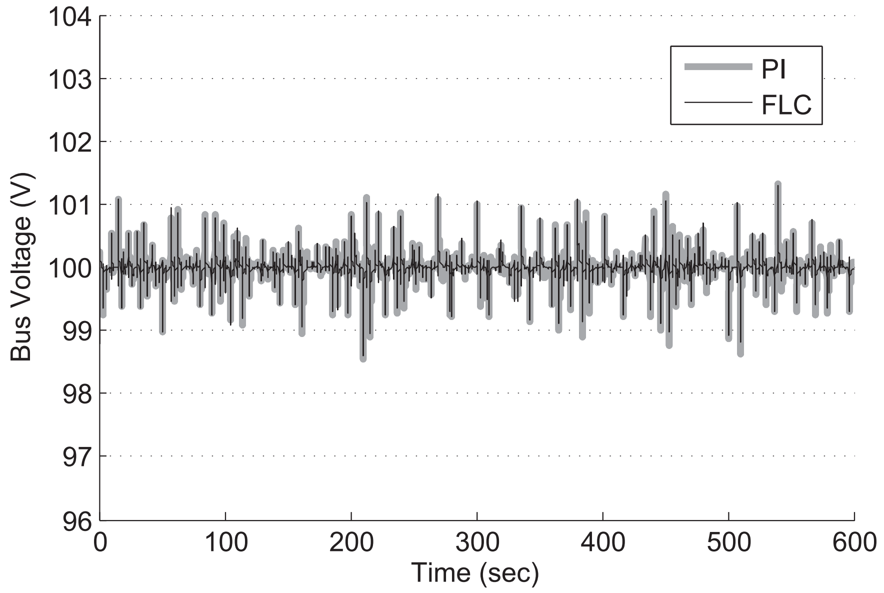


Figure 4.22: Bus Voltage - Energy Surplus Simulation

4.3.2.5 Energy Surplus

In the opposite case of the previous "deficit" scenario (see 4.3.2.4 above), the two storage banks were set at high SoC values, and the variable source was set to produce an average surplus of 100 W. Figures 4.22 through 4.25 show the final simulation's bus voltage, battery voltage, ultracapacitor voltage, and OVD current respectively. Table 4.5 shows the average numerical data regarding the systems' performances.

Both systems avoid excessively-high SoC conditions with the storage elements. In this situation, unlike those previous, the FL control system produced better bus voltage regulation than the PI control system. This is likely due to the integral control present on the PI system's OVD channel outer control loop, which then slows the stabilizing

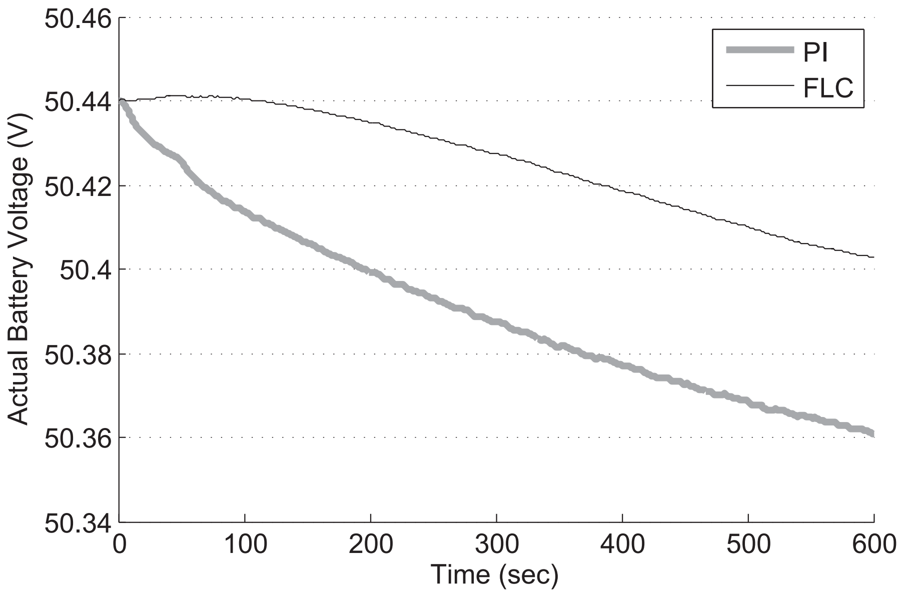


Figure 4.23: Battery Bank Voltage - Energy Surplus Simulation

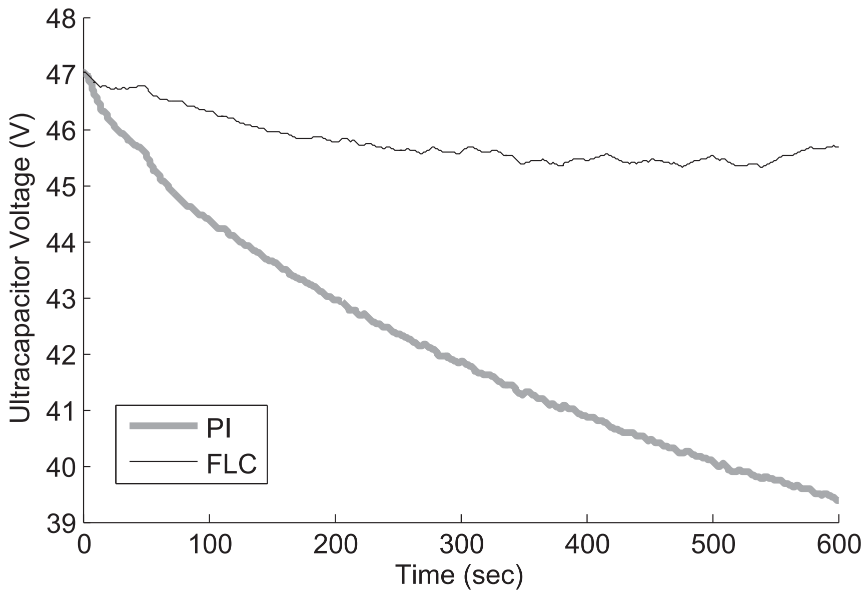


Figure 4.24: Ultracapacitor Bank Voltage - Energy Surplus Simulation

effect of the storage element control loops.

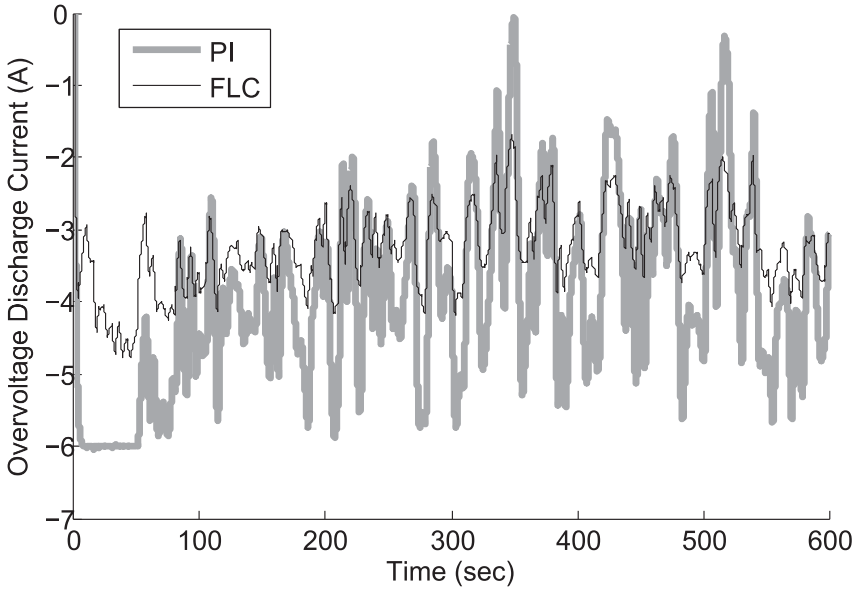


Figure 4.25: OVD Current - Energy Surplus Simulation

Parameter	PI	FL
$\int \lvert v_{err} \rvert$	51.302	27.798
$\int v_{err}^2$	10.598	3.149
$\int \lvert i_{batt} \rvert$	259.72	113.94
$\int \lvert i_{ucap} \rvert$	1340.1	755.62
$\int \lvert i_{ovd} \rvert$	2452.8	2034.6
$v_{batt-final}$	50.361	50.407
$v_{ucap-final}$	39.557	45.619

Table 4.5
Numerical Results - Energy Surplus Simulation

4.4 Remarks

The simulations conducted and presented here indicate that the FL controller can regulate bus voltage in a manner similar to that of a PI control system. Additionally, the rules present in the FL engine allow for the additional functionalities of self-protection, storage bank balancing, and startup. These simulations are validated through hardware experiments in the next chapter.

Chapter 5

Experiment

5.1 Implementation

5.1.1 General Considerations

Because the control systems are meant to respond rapidly to excursions in bus voltage, the choice was made to implement the two control systems operate on a fixed update rate, with no numeric solver for the integral functions. The lack of a solver reduces numeric precision, but allows for an increased system update frequency.

To avoid heavy beat frequencies, the PWM generators for the stabilizer were set to different frequencies: 15 kHz, 13.63 kHz, and 12.5 kHz. The PWM generators of the source and load were set to 10 kHz.

Power switching is an inherently noisy operation, both physically and electrically speaking. While the physical noise is usually a mere minor irritation, the electrical noise can cause spurious operation and needs to be suppressed. The current transducers in the power inverter have a 1st-order low-pass filter built in, which is tuned in this instance to have a 10 kHz corner frequency. This alone is insufficient, as even a PCB trace mostly shielded between power planes can still pick up interference (especially at connector crossings). An additional low-pass RC filter at the input to the ADC has a corner frequency of 1.1 kHz. Finally, a software filter was implemented in the stabilizers to further reduce noise. This filter is implemented as a one-state IIR filter. The difference equation and transfer function are

$$y[n] = \frac{1}{2}y[n-1] + \frac{1}{2}x[n] \qquad \rightarrow \qquad H[z] = \frac{1}{2 - z^{-1}}. \tag{5.1}$$

Given the system update and sampling period of 0.0002 seconds, (5.1) results in a -3 db corner frequency of 576 Hz. Because the ADC module outputs readings as unsigned integers, implementing this filter requires only two bit-shifts and an addition to form the new output.

The voltage measurements undergo active 4th-order low-pass filtering before being sampled by the ADC, with a corner frequency of approximately 1 kHz. This reduces the need for extra filtering, but the same software-filter treatment is given the voltage signals as the current signals.

The PI program blocks used in this and all the hardware control systems have integrators that do not take the sampling time into account; entering an integral gain of 0.1 into the block parameters results in an effective gain of 500 (given the system update rate of 5000 Hz). The integral gains taken from the simulation were scaled appropriately before use.

For the random source and load, the process by which random numbers are generated relies on the noise inherent in sampling signals. Each update tick, an ADC channel's output is added to a running summation. When the time comes to generate a new random number, thousands of individual measurements will have been summed. The lowest byte is then taken as the new random number. While Simulink includes a block for a zero-order hold, the use of said block results in the Embedded Coder generating a rate-monotonic scheduling system. To avoid this, a custom zero-order hold subsystem was created based on running counters. Though more complex to program, it avoids the overhead costs associated with implementing a real-time task scheduler.

The entire microgrid was implemented with a common two-line bus, one for positive voltage and voltage return. Each element was given its own pair of fused connections to the main bus. Two oscilloscopes captured measurements on the relevant system states. Figure 5.1 shows a picture of the microgrid system. The bus and ballast are shown on the top bench shelf on the upper right; the storage elements, loads, and power supplies are on the shelving unit to the left; the power inverters and control systems are on sliding drawers in the black cabinet, bottom center.

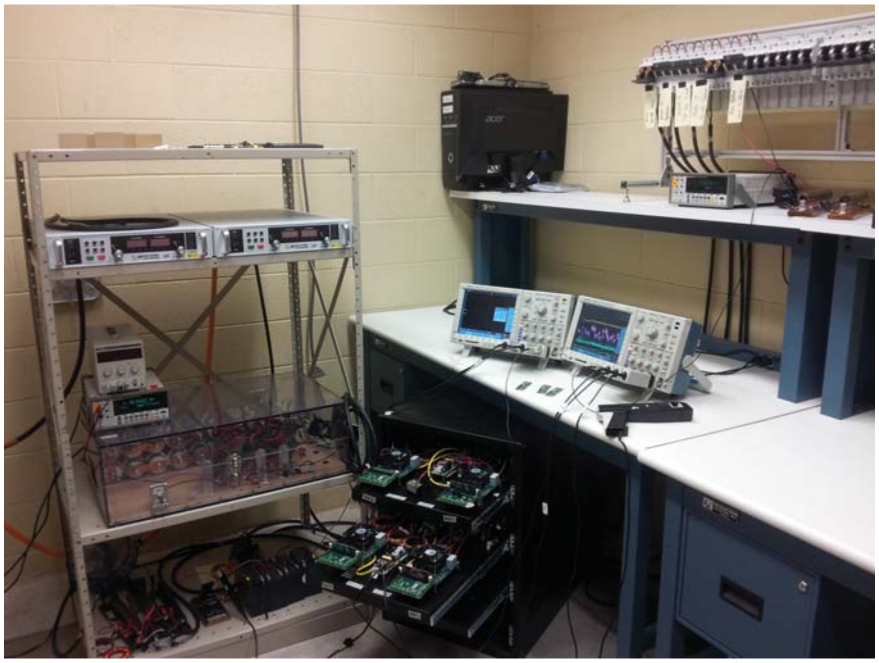


Figure 5.1: Experimental Microgrid System

5.1.2 Stochastic Source

The stochastic source is implemented as a single-phase boost converter. The low leg consists of a 1 mF inductor connected to a MagnaPower dc supply operating at

60 V with a 13 A current limit. The specific reference value for output power is determined as an offset to a pseudo-random number (generated as described above in 5.1.1). The specific offset is based on the nominal ballast power draw, a small amount of conversion losses, and any surplus/deficit required by the scenario under consideration. A PI control system tracks the input current and voltage to match the commanded power. (Given that the input voltage is fixed at 60 V, the control system primarily reacts to the input current.)

5.1.3 Stochastic Load

The experimental implementation of the stochastic impedance load is that of a two-phase interleaved buck converter. The low-side legs each consist of a 1 mF inductor, a 25 Ω, 250 W power resistor, and a 1 μF film capacitor for transient suppression. The power draw commanded by the whole converter is split equally between two phases.

The power draw P_{nom} is determined as a pseudo-random number in a range of powers, but with a fixed offset of 50 W. The control system is then responsible for maintaining a voltage across a set of power resistors that corresponds to the appropriate impedance. To maintain a constant impedance instead of a constant power, the exact level of output varies proportionally to the square of the bus voltage. That relationship has been shown previously in 3.6.

5.1.4 Stabilizer

The program that implements the cascaded PI control is created almost entirely through MATLAB/Simulink, with only the software ADC filters added manually. The system is very similar to that implemented in the simulation, with conditioned ADC signals comprising the inputs to the control system and the output being adjusted for use with the PWM interfaces. The Simulink code that comprises the experimental PI control system is given in Appendix C.

While Simulink provides a convenient API for generating fuzzy logic systems, the code it produces is far too resource-intensive to be implemented on the MCU selected for this research. Instead, a copy of the PI control system was used as the base. The outer PI loop was removed and the FL control algorithms were inserted into its place through manual programming in C. Fortunately, the Embedded Coder ACG retains all the C source and support files when producing an MCU-executable object from a Simulink model. Generating a custom executable is as straightforward as modifying the original source code, initiating the makefile, and loading the resulting executable onto the MCU. The source code for the FL control system is included in Appendix D.

5.2 Test Cases

It is difficult to match battery SoC conditions from test to test without an excessive charging and resting time between individual runs to reset the battery bank. With this in mind, the system was placed in similar conditions with respect to the fuzzy logic input functions. Starting with the battery bank at **high** might result in a battery voltage of 51.4 V on one run and 51.6 V the next. Starting with the bank in a neutral state would mean the battery would be within the range 49.2 $\pm$0.25 V. Useful analysis of the system can still be made if the states cross through similar ranges of values (with respect to the fuzzy input functions and SoC levels).

The test cases used to compare the two control systems mirror some of the scenarios performed in the simulations. One noteable exception to this is the "high battery, low ultracapacitor" startup scenario. In the simulation, the ultracapacitor bank started at 5% SoC, which is equivalent to a bank voltage 21.5 V. In the physical tests, the ultracapacitor bank was discharged even further (typically around 10 V or less).

5.3 Results

The general test process was to configure the microgrid for the given scenario, start both the variable source and load, start recording data with two oscilloscopes, run the system without stabilization for about 20 seconds, activate the stabilizer and continue recording data for about 6 minutes, then deactivate the stabilizer and finish recording after 20 more seconds.

Data was recorded at 25 kHz, then reduced to 2.5 kHz by taking the average of every 10 samples. Finally, the data was processed through the same IIR filter as implemented in the controllers (see Equation 5.1, but with a sample rate of 2.5 kHz instead of 5 kHz). The cumulative effect of these operations was to reduce noise while still allowing for the capture of transient behavior as system states shift.

5.3.1 Normal Regulation

For this scenario, both storage elements were charged towards the middle of their SoC ranges and the random source was set to produce no significant average surplus or deficit of power. Both control systems were examined in this scenario. The data captured for this simulation was clipped to approximately 6 minutes of data to match the quantity between experiments. Figure 5.2 shows the bus voltage regulation of the two control systems, and Figure 5.3 shows the battery bank current for the same.

Because this experiment comprises the baseline for "normal" performance, the measured nominal voltage of the system (as produced by the stabilizer systems) was calculated from the resulting data. Averaging the bus voltage measurements across both systems' data produced a value of 99.91 V for the measured nominal bus voltage. This small decrease from the programmed nominal voltage of 100.0 V is likely due to line losses, as the stabilizer measures the voltage at the output filter capacitor and the oscilloscopes were connected to the ballast to measure the bus voltage.

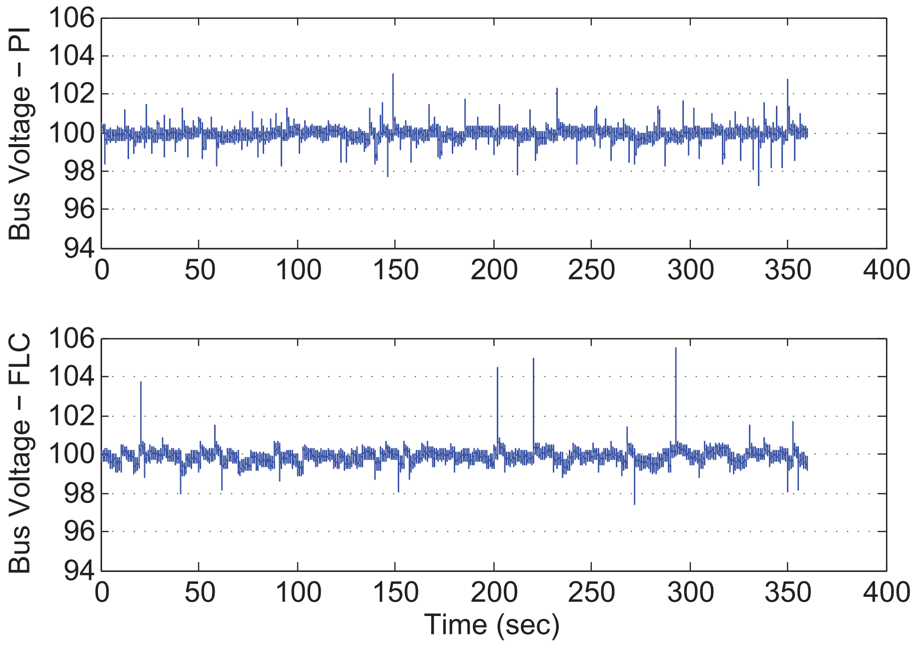


Figure 5.2: Bus Voltage - Normal Regulation Experiment, Top: PI Control, Bottom: FL Control

The numerical results for this experiment show an integrated bus voltage error of

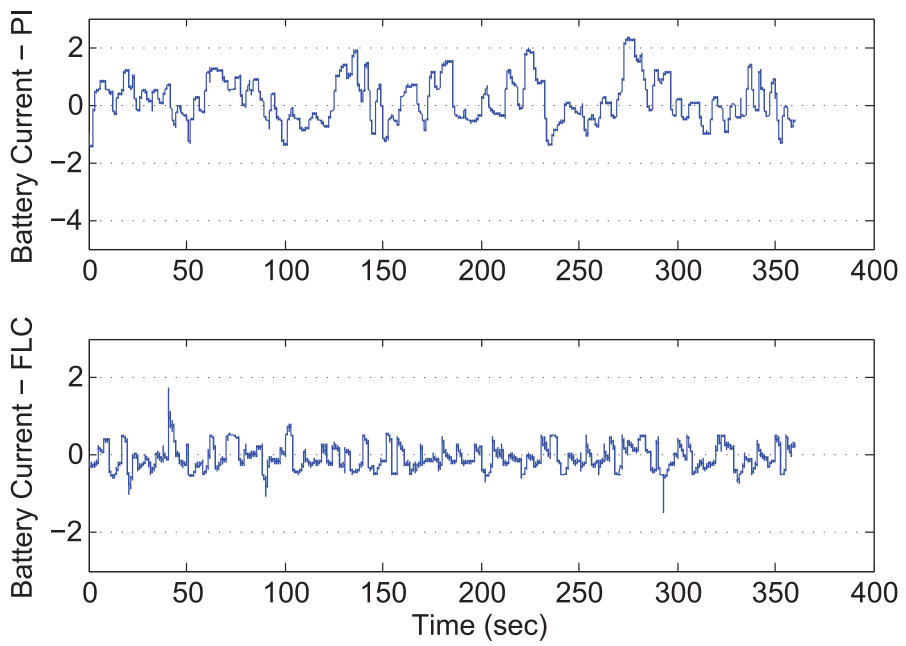


Figure 5.3: Battery Bank Current - Normal Regulation Experiment, Top: PI Control, Bottom: FL Control

59.109 $V \cdot sec$ for the PI control system and 76.555 $V \cdot sec$ for the FL control system. The integrated error squared results in 20.062 $V^2 \cdot sec$ and 30.806 $V^2 \cdot sec$ respectively. The battery current measured 226.38 $A \cdot sec$ and 96.68 $A \cdot sec$ respectively. From the numerical results, the FL control system appears to have noticeably decreased performance in bus voltage regulation when compared to the PI control system. However, when viewing the actual bus voltage, it appears that a moderate portion of this error arises from a few isolated voltage spikes, whereas the PI control system has smaller but more frequent peak deviations from nominal voltage. Despite the modest decrease in regulation performance, the FL control system exhibits significantly less demand upon the battery.

5.3.2 Energy Deficit

The next scenario tested was that of an extended energy deficit. Both energy storage elements were discharged to low SoC levels, and the variable source was configured to produce a long-term average deficit of 100 W. The system was operated for approximately 6 minutes, and 350 seconds of data was taken for comparison between the two control systems' performances.

Figure 5.4 shows the bus voltage regulation by the two control systems. Figures 5.5 and 5.6 show the battery and ultracapacitor bank currents, and Figure 5.7 shows the voltages of the storage elements.

In this instance, the FL control system performs poorly at regulating bus voltage while the PI control system performs reasonably well. However, this comes at the expense of potentially overdischarging the battery bank; the PI control system results in a 12.2% SoC reduction, while the FL control system causes a drop of only 4.7% SoC.

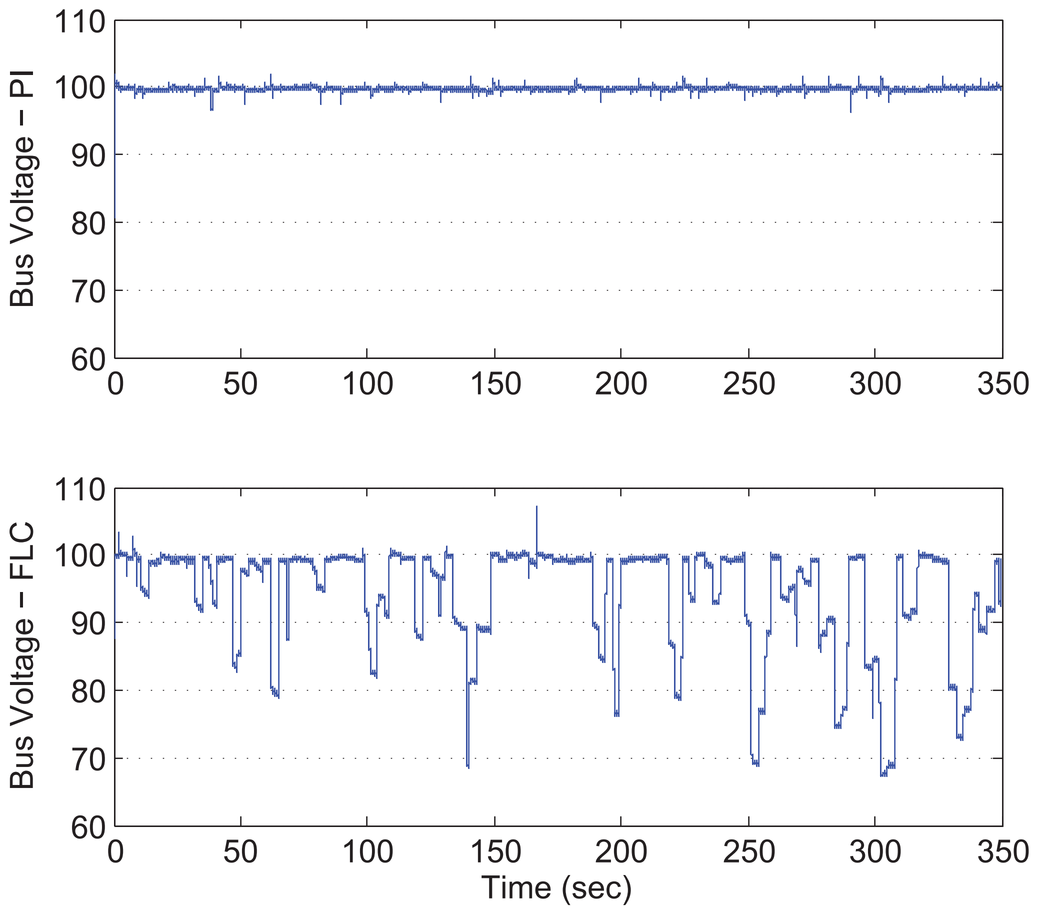


Figure 5.4: Bus Voltage - Energy Deficit Experiment, Top: PI Control, Bottom: FL Control

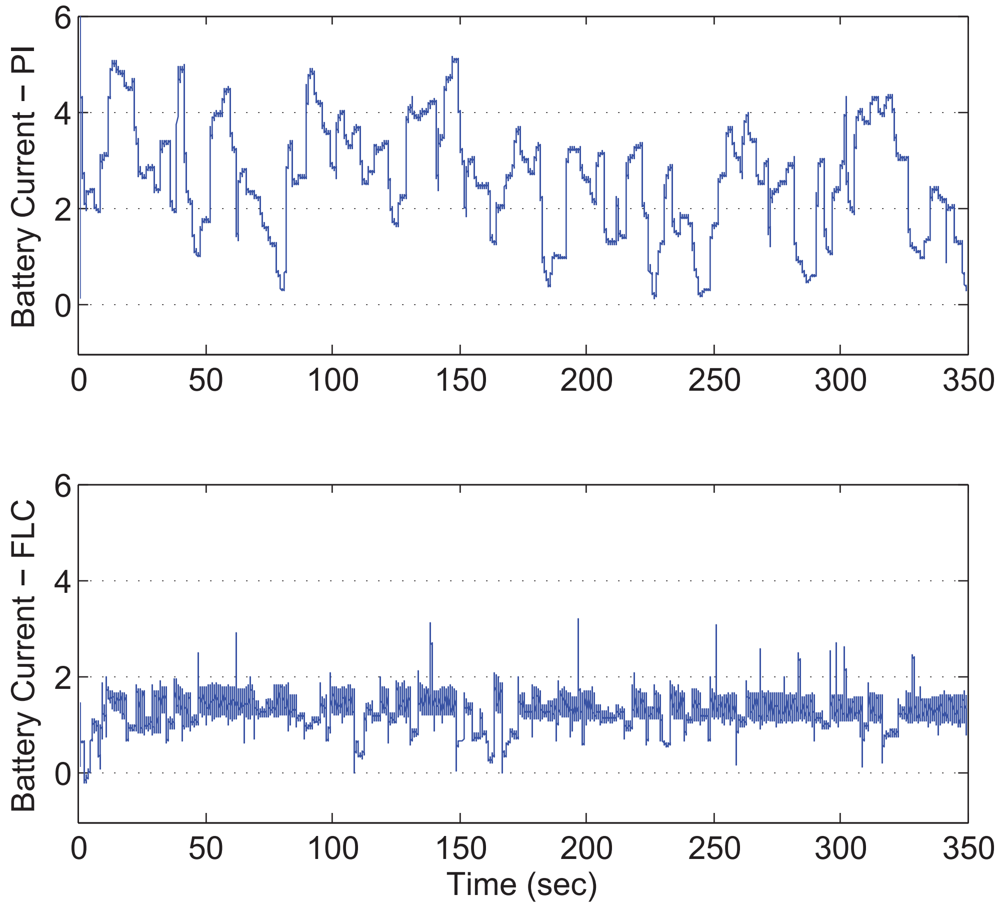


Figure 5.5: Battery Bank Current - Energy Deficit Experiment, Top: PI Control, Bottom: FL Control

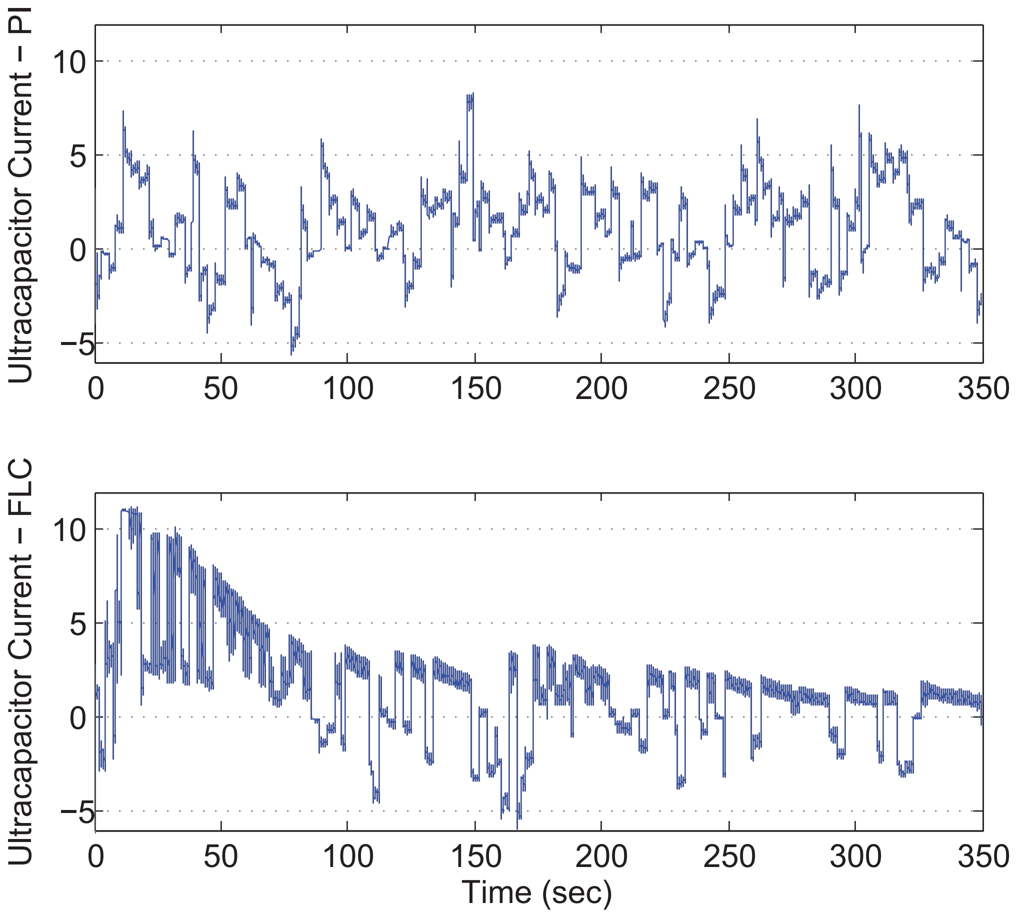


Figure 5.6: Ultracapacitor Bank Current - Energy Deficit Experiment, Top: PI Control, Bottom: FL Control

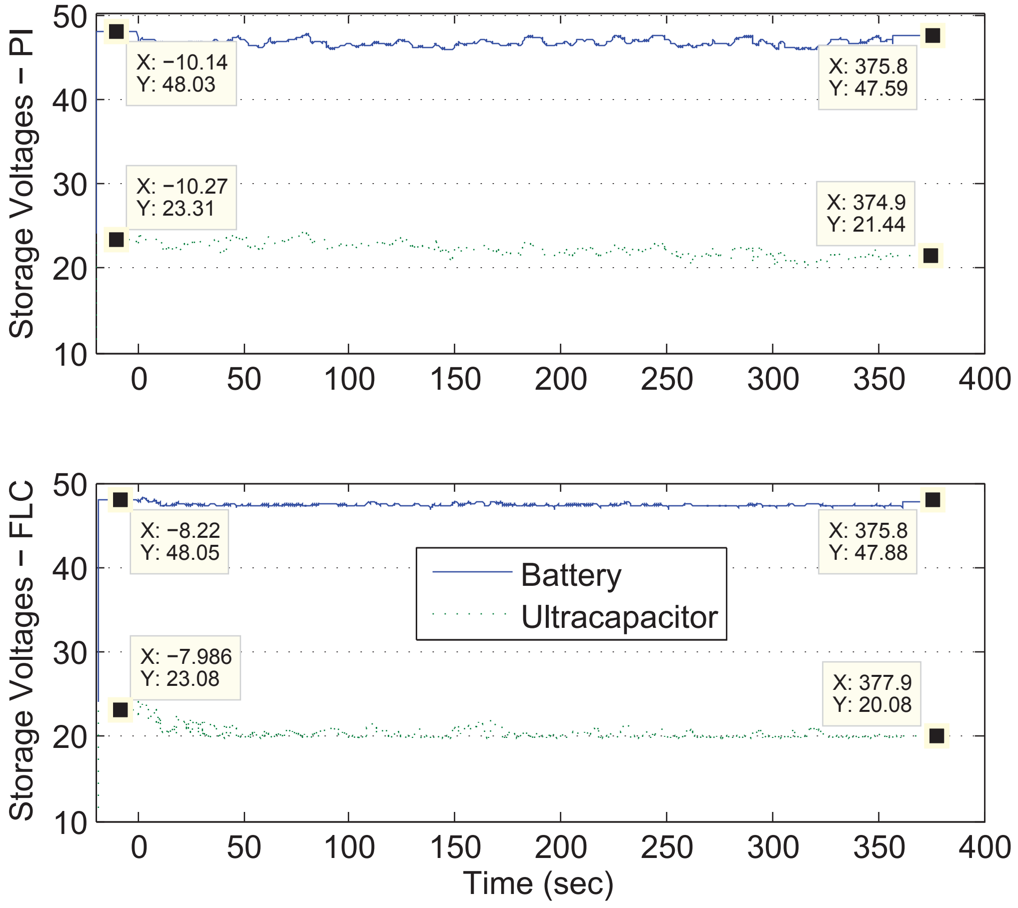


Figure 5.7: Storage Element Voltages - Energy Deficit Experiment, Top: PI Control, Bottom: FL Control

5.3.3 Depleted Battery Bank

Another scenario tested was that of an imbalanced battery and ultracapacitor bank: the battery was drained to below 20% SoC and the ultracapacitor bank was charged to above 95% SoC. Both control systems were tested in this scenario. The system was operated for approximately 6 minutes, and the datasets were trimmed to 350 seconds for uniformity between the two experiments.

Figure 5.8 shows the bus voltage regulation by the control systems. Figures 5.9 and 5.10 show the storage phase currents for both control systems, and Figure 5.11 shows the storage phase voltages[1]. Lastly, Figure 5.12 shows the OVD phase current produced by the control systems.

The results of this experiment show how the FL controller can successfully transfer energy between highly-imbalanced storage phases (see Table 5.1). The FL control system results in the battery bank gaining 15.0% SoC while the PI control system results in a 7.78% SoC decrease. At the same time, despite the significant gain in battery bank SoC, the FL system still results in a large reduction in battery current (in contrast to the PI control system). Bus voltage regulation is significantly

[1]While most datasets in this scenario are trimmed to 360 seconds, the storage voltage dataset is presented in its entirety, including non-stabilizing portions, to show beginning and ending bank voltages.

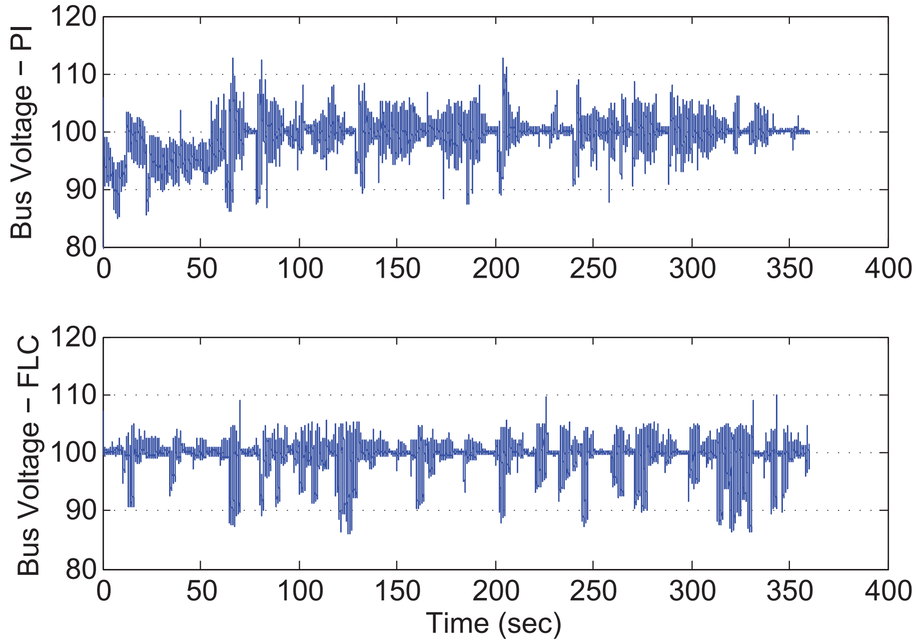


Figure 5.8: Bus Voltage - Depleted Battery Bank Experiment, Top: PI Control, Bottom: FL Control

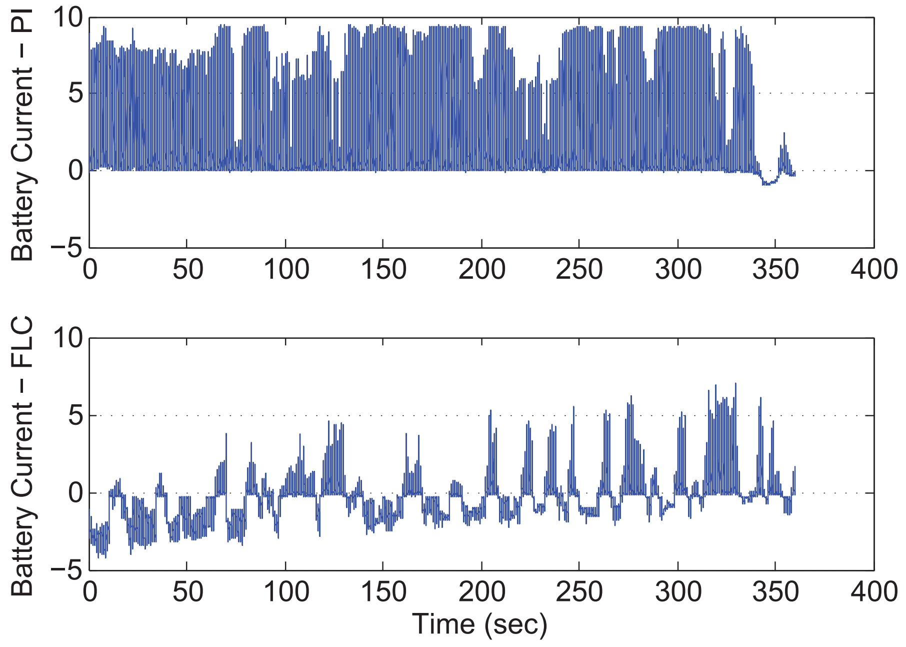


Figure 5.9: Battery Bank Current - Depleted Battery Bank Experiment, Top: PI Control, Bottom: FL Control

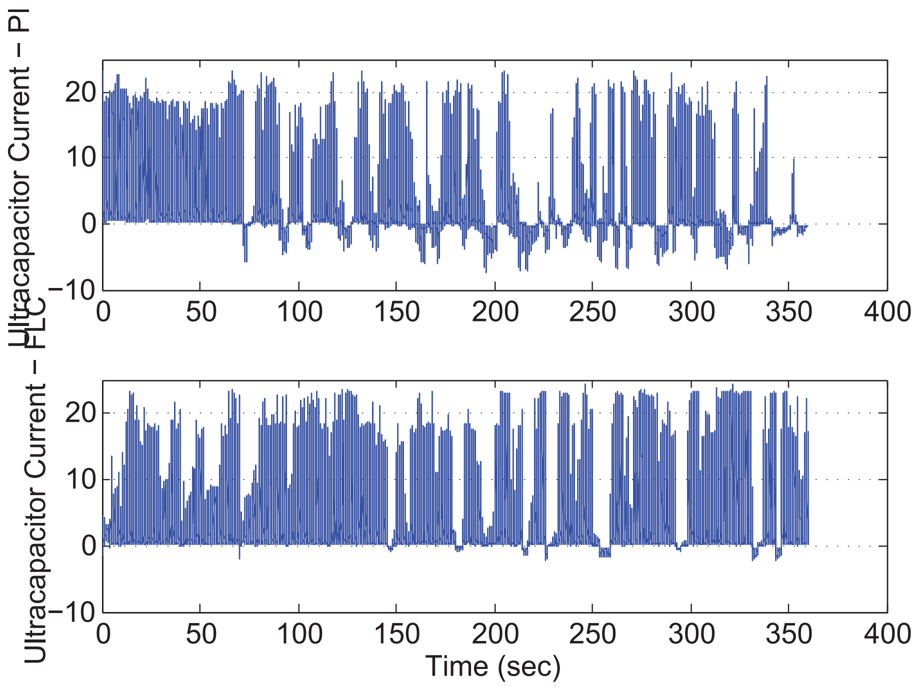


Figure 5.10: Ultracapacitor Bank Current - Depleted Battery Bank Experiment, Top: PI Control, Bottom: FL Control

degraded in both systems compared to normal regulation (see 5.3.1); the actual performance metrics are very similar between control systems. Finally, the PI control system (needlessly) sinks over four times more current in the OVD phase than the FL controller.

Table 5.1
Numerical Results - Depleted Battery Bank Experiment

Parameter	PI	FL
$\int \lvert v_{err} \rvert$	683.08	548.71
$\int v_{err}^2$	3624.2	3158.6
$\int \lvert i_{batt} \rvert$	453.62	274.24
$\int \lvert i_{ucap} \rvert$	581.73	635.6
$\int \lvert i_{ovd} \rvert$	628.05	150.00
$v_{batt-start}$	47.94	48.01
$v_{batt-final}$	47.63	48.52
$v_{ucap-start}$	48.89	49.44
$v_{ucap-final}$	46.72	45.74

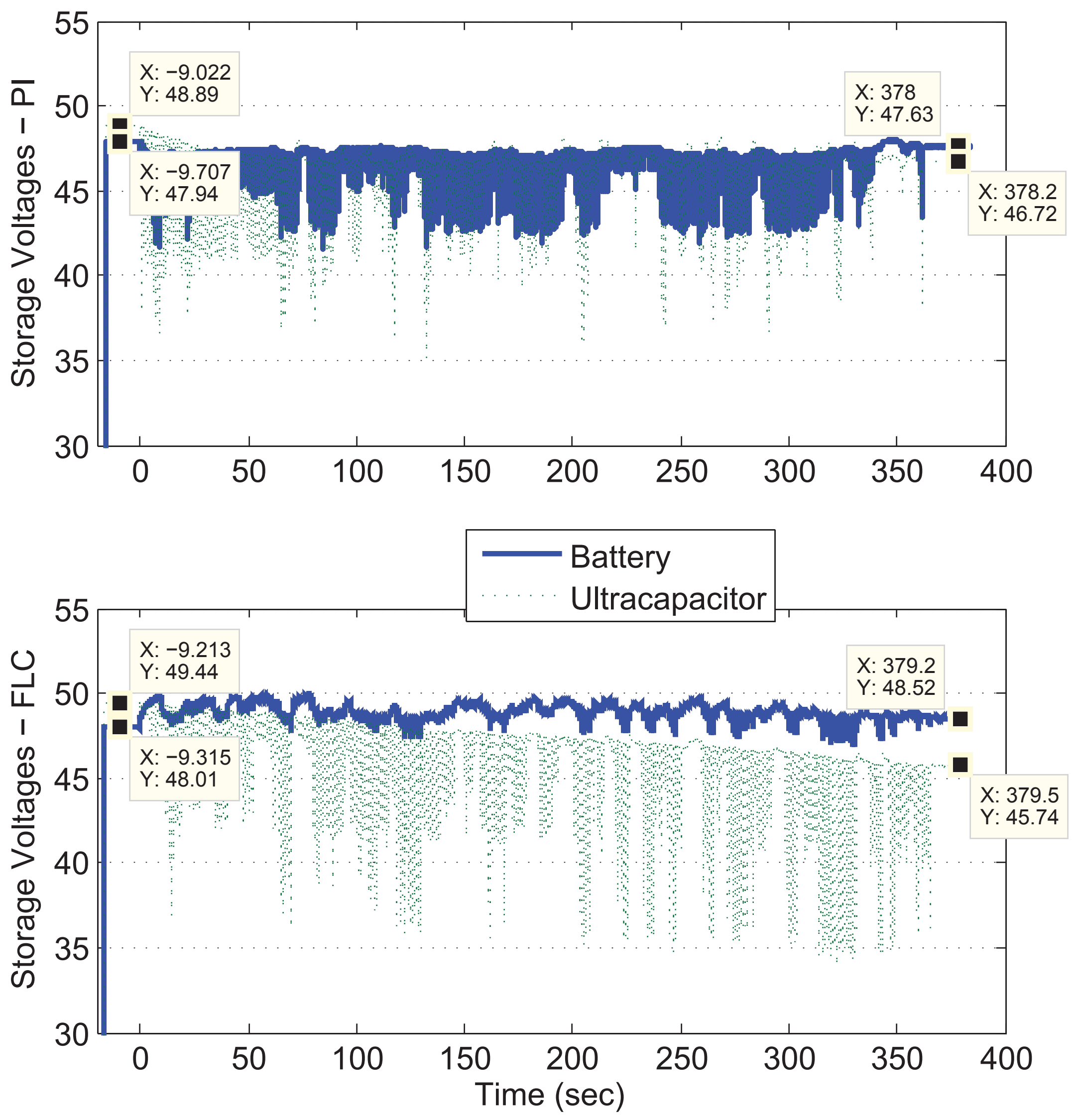


Figure 5.11: Storage Element Voltages - Depleted Battery Bank Experiment, Top: PI Control, Bottom: FL Control

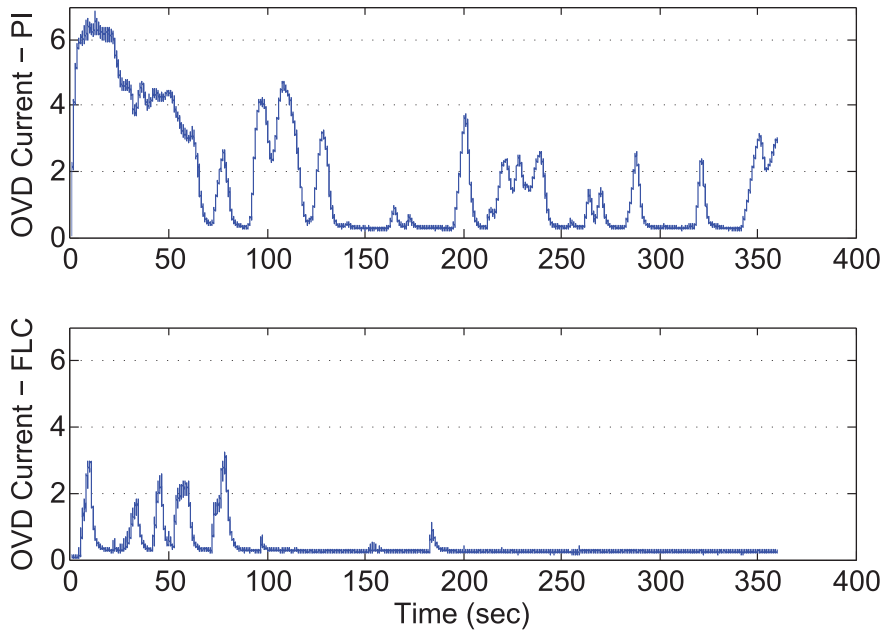


Figure 5.12: OVD Phase Current - Depleted Battery Bank Experiment, Top: PI Control, Bottom: FL Control

5.3.4 Depleted Ultracapacitor Bank

This scenario started with the battery bank fully charged and the ultracapacitor bank depleted down to approximately 10 V. The FL-controlled stabilizer was run for 6 minutes, and the relevant data is given below in Figures 5.13 through 5.16. Data was not gathered on the PI control system's performance, as it triggered a thermal event with rapid oxidization and photonic emission that led to the hardware's permanent malfunction.

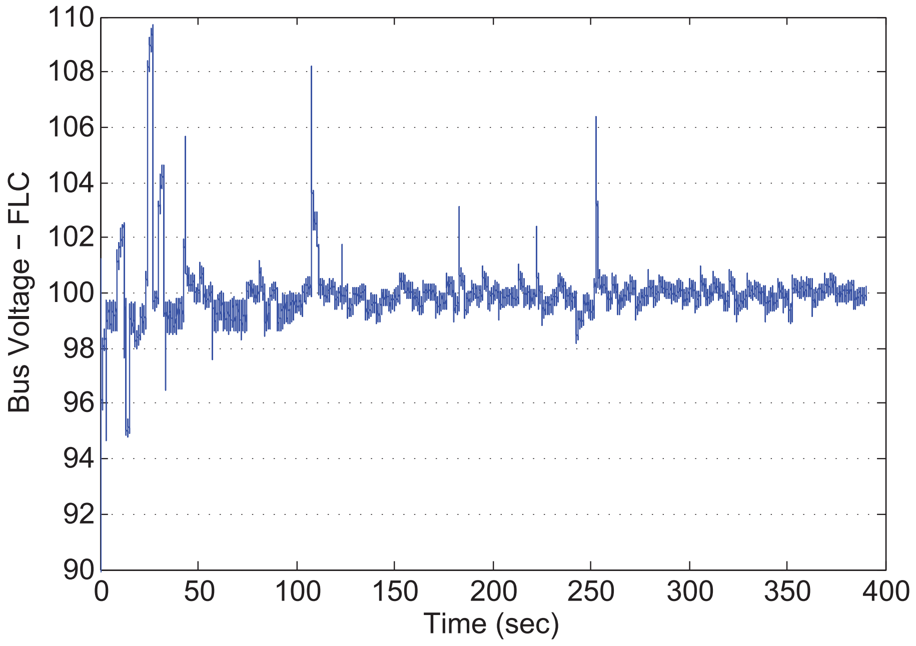


Figure 5.13: Bus Voltage - Depleted Ultracapacitor Bank Experiment

While the bus voltage undergoes high variations early into the startup process, the FL control system quickly charges the ultracapacitor bank to working voltage levels

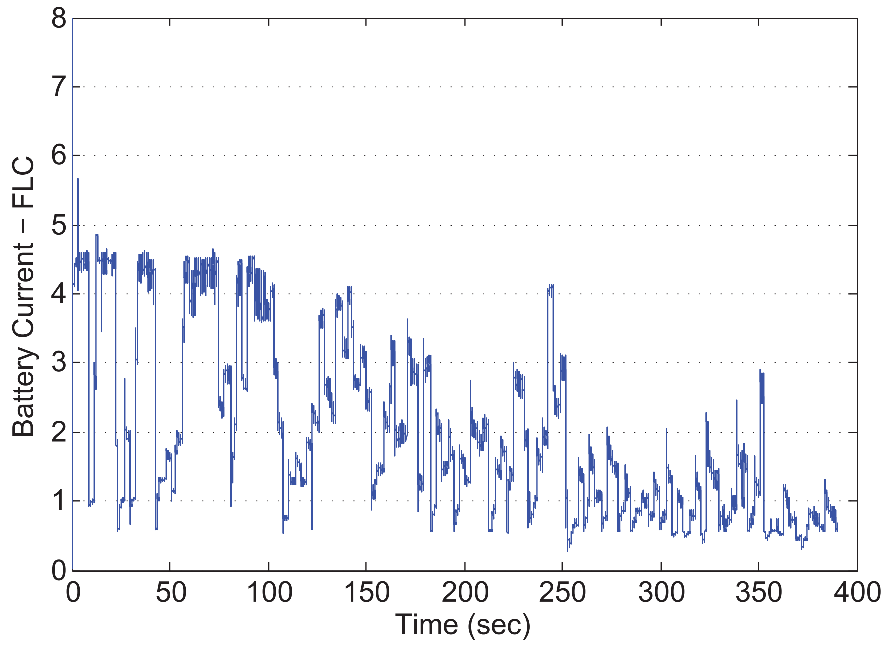


Figure 5.14: Battery Current - Depleted Ultracapacitor Bank Experiment

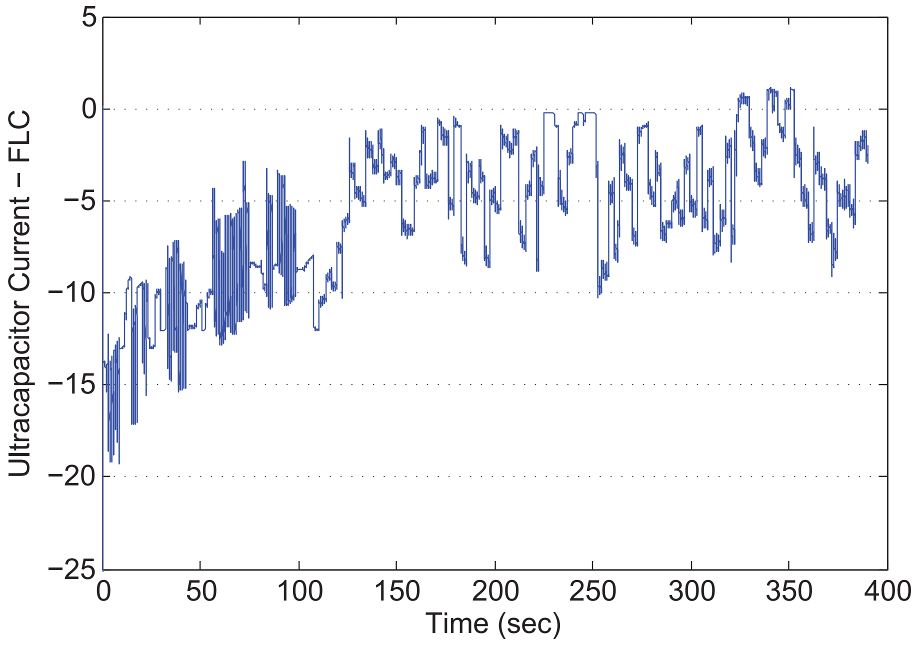


Figure 5.15: Ultracapacitor Current - Depleted Ultracapacitor Bank Experiment

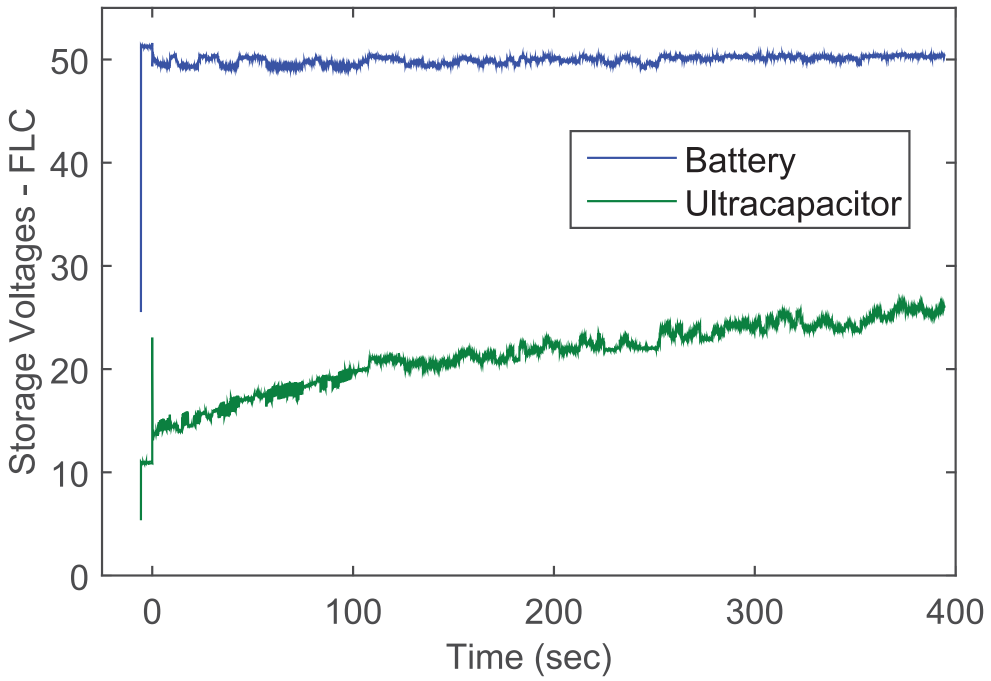


Figure 5.16: Storage Element Voltages - Depleted Ultracapacitor Bank Experiment

from a heavily-depleted state, moving from an initial level of 11.5 V up to a level of approximately 26 V.

Chapter 6

Discussion & Conclusions

6.1 Simulation vs Experiment

The Simulink model of the system performs well with predicting the general behavior of the system. While there is a certain amount of inaccuracy inherent in every mathematical model of a real system, sufficient similarity exists to make reasonable comparisons between simulated performance and actual performance.

One factor wherein the simulation is *more* accurate than the real system is the issue of sampling time. The computer simulations utilized a variable-step solver with a minimum step size of 10^{-10} seconds. This is far faster than the experimental system performs. It is likely that the computation of the implemented control systems could be sped up to a certain degree, but not nearly enough to approach the simulation's fine granularity. The general effects of a sampled system are increased phase delay and decreased damping [37].

Most of the discrepancies in accuracy between the simulations and experiments is inconsequential to the overall goal of verifying general functionality. Both the simulation and the experiments show the operation of the microgrid system under a variety of likely operating conditions.

6.2 PI Control vs FL Control

The ability to regulate bus voltage in the presence of fluctuations has been sufficiently demonstrated with both control systems. Additionally, it has been shown how the FL control system can accomplish secondary goals of battery optimization, storage balancing, and self-protection; the PI control system, in contrast, is unable to properly implement these functionalities.

The FL control system occasionally suffers from inaccuracies in measuring the storage element voltages. The battery bank, in particular, can cause the system to cross into operating conditions that do not reflect the true state of the system. This could be largely accounted for by characterizing the losses inherent in the cabling and storage elements and adjusting for these impedances in software.

The difference in how the FL controller tends to reduce peaking while the PI control system typically produces better overall regulation is likely caused by the integral term. In the FL engine rule base (see 3.4.2.3), there are no rules for regulating bus voltage when the error is negative and the accumulated (integrated) error is positive (or vice versa). When the source and/or load changes states, the PI control system integrator is at a state that provides regulation for the prior state, and can impart a brief detrimental effect until the error can accumulate to match the new state. The FL controller, on the other hand, acts to largely ignore the integrator until or unless it changes sufficiently to match the standard error signal. With no integral term, the FL control system then briefly acts similar to a nonlinear proportional control. This also explains the small but noticeable amount of ringing in the bus voltage under FL control (see again Figure 4.8).

This specific type of FL control system is *not* suited for integration with systems based on droop control. By driving the system to a specific voltage reference value, the storage elements would be quickly depleted in an effort to maintain nominal bus voltage. A more appropriate situation for usage would be that of an isolated power grid with a significant percentage of renewable energy sources. The variability of said renewable sources can be counteracted by the FL-based stabilizer.

This system, as with many other stabilization systems, suffers from the “sizing problem:” the storage element capacities and inverter capabilities must be matched to the expected requirements of the system. A 1 kW stabilizer likely cannot regulate a 50 kW system, and it would be a waste to use a 30 kW stabilizer to regulate a 5 kW system.

The PI control system also provides insight into how interleaving the different storage elements onto one converter can allow for the additional functionality provided by the FL control system vs. a system that has the elements separated onto individual converters and controllers. The PI control loops have no information transfer between them, both in implementation and in theory, and as such cannot operate in consideration of each other’s states. Separating the storage elements and not implementing a communication system would inhibit energy transfer functionality and lead to decreased efficiency of energy usage.

6.3 Conclusions

This research has demonstrated that an FL control system can provide dc bus voltage regulation with performance similar to that of a traditional PI controller. It has also been shown that the FL controller can provide additional optimized functionality when dealing with different types of storage elements, whereas a PI controller generally cannot. The extra functionalities demonstrated here include self-protection, energy storage balancing, and prioritized usage of certain types of storage elements (specifically the battery bank, in this research).

Chapter 7

Future Work

The most direct extension of this work is implementing the FL control system for an ac power grid. This would add the complexity of accounting for not only voltage level, but real and reactive powers and frequency regulation. Capacitor banks are already used for providing power factor correction, and active interfacing via power electronics would improve on such capability. The high power ratings of ultracapacitors could be harnessed during synchronization events.

The system implemented in this research is completely isolated from a communications perspective. One area in which to increase functionality would be integration with microgrid management algorithms. Providing SoC levels to a central controller could help inform generation scheduling decisions. For example, if the stabilizer system was operating with the storage elements at a lower SoC, additional power generation could be scheduled. Knowing the stabilizer would absorb excess power, a controlled surplus of power would bring the stabilizer back to regular SoC ranges.

While the FL controller implemented here utilizes batteries and ultracapacitors, the system could be modified to accommodate other types of storage systems. Flywheels are a good candidate for further research efforts into fuzzy logic grid stabilization.

References

[1] N. Hatziargyriou, H. Asano, R. Iravani, and C. Marnay, “Microgrids,” *IEEE Power and Energy Magazine*, vol. 5, no. 4, pp. 78–94, July 2007.

[2] V. Prado, T. Seager, A. Mechtenberg, and E. Bennett, “A systemic thermodynamic analysis of fuel consumption at forward operating bases,” in *IEEE International Symposium on Sustainable Systems and Technology*, May 2011, pp. 1–6.

[3] P. T. Krein, *Elements of Power Electronics.* New York, NY: Oxford University Press, 1998.

[4] J. Richmond, S. Leslie, B. Hull, M. Das, A. Agarwal, and J. Palmour, “Roadmap for megawatt class power switch modules utilizing large area silicon carbide mosfets and jbs diodes,” in *IEEE Energy Conversion Congress and Exposition*, Sept 2009, pp. 106–111.

[5] C. e. a. Augustine, *Renewable Electricity Generation and Storage Technologies.*

Golden, CO: National Renewable Energy Laboratory, 2012, vol. Vol 2. of Renewable Electricity Futures. [Online]. Available: http://www.nrel.gov/analysis/re_futures

[6] "Energy Storage Technology Roadmap," International Energy Agency, Tech. Rep., March 2014.

[7] A. Jessop, "Geothermal Energy from Old Mines at Springhill, Nova Scotia, Canada," in *World Geothermal Congress.* International Geothermal Association, 1995.

[8] "Water Electrolysis and Renewable Energy Systems," Fuel Cell Today, Tech. Rep., 2013, [Accessed Nov. 11, 2014]. [Online]. Available: http://www.fuelcelltoday.com/analysis/surveys/2013/water-electrolysis-renewable-energy-systems

[9] D. Jenkins, J. Fletcher, and D. Kane, "Lifetime prediction and sizing of lead-acid batteries for microgeneration storage applications," *IET Renewable Power Generation*, vol. 2, no. 3, pp. 191–200, September 2008.

[10] J. Miller, "A brief history of supercapacitors," *Batteries and Energy Storage Technology*, pp. 61–78, 2007, [Accessed Nov. 17, 2014]. [Online]. Available: http://www.cantecsystems.com/ccrdocs/brief-history-of-supercapacitors.pdf

[11] J.-S. Lai, S. Levy, and M. Rose, "High energy density double-layer capacitors for

energy storage applications," *IEEE Aerospace and Electronic Systems Magazine*, vol. 7, no. 4, pp. 14–19, April 1992.

[12] L. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, pp. 338–353, 1965.

[13] W. Leekwijck and E. Kerre, "Defuzzification: criteria and classification," *Fuzzy Sets and Systems*, vol. 108, pp. 159–178, 1999.

[14] P. Thounthong, S. Sikkabut, A. Luksanasakul, P. Koseeyaporn, P. Sethakul, S. Pierfederici, and B. Davat, "Fuzzy logic based dc bus voltage control of a stand alone photovoltaic/fuel cell/supercapacitor power plant," in *11th International Conference on Environment and Electrical Engineering*, May 2012, pp. 725–730.

[15] R. Sathishkumar, S. Kollimalla, and M. Mishra, "Dynamic energy management of micro grids using battery super capacitor combined storage," in *India Conference (INDICON), 2012 Annual IEEE*, Dec 2012, pp. 1078–1083.

[16] A. Stienecker, T. Stuart, and C. Ashtiani, "A combined ultracapacitor-lead acid battery storage system for mild hybrid electric vehicles," in *IEEE Conference on Vehicle Power and Propulsion*, Sept 2005, pp. 6 pp.–.

[17] D. Haifeng and C. Xueyu, "A study on lead acid battery and ultra-capacitor hybrid energy storage system for hybrid city bus," in *International Conference on Optoelectronics and Image Processing*, vol. 1, Nov 2010, pp. 154–159.

[18] E. Bilbao, H. Gaztaaga, L. Mir, I. Etxeberria-Otadui, and A. Milo, "Design and development of a supercapacitor-based microgrid dynamic support system," in *13th European Conference on Power Electronics and Applications*, Sept 2009, pp. 1–10.

[19] O. Onar and A. Khaligh, "A novel integrated magnetic structure based dc/dc converter for hybrid battery/ultracapacitor energy storage systems," *IEEE Transactions on Smart Grid*, vol. 3, no. 1, pp. 296–307, March 2012.

[20] C. Papadimitriou and N. Vovos, "A fuzzy-logic control strategy for a hybrid fuel cell-battery system offering ancillary services," in *13th European Conference on Power Electronics and Applications*, Sept 2009, pp. 1–10.

[21] ——, "A fuzzy control scheme for integration of DGs into a microgrid," in *15th IEEE Mediterranean Electrotechnical Conference*, April 2010, pp. 872–877.

[22] *PS21765 Application Note*, Mitsubishi Electric, 2007, [Accessed Oct. 12, 2014]. [Online]. Available: http://www.pwrx.com/pwrx/app/mini_dipipm_ver4_rev2_e.pdf

[23] *IPC2221: Generic Standard on Printed Board Design*, Rigid Printed Board Committee (D-30) of IPC, 2011.

[24] (2014) TMS320F28335 controlCARD. Texas Instruments. [Accessed Nov. 23, 2014]. [Online]. Available: http://www.ti.com/tool/tmdscncd28335

[25] (2014) MATLAB/Simulink Product Overview. Mathworks, Inc. [Accessed Nov. 15, 2014]. [Online]. Available: http://www.mathworks.com/products/matlab/

[26] (2014) Code Composer Suite v6.0. Texas Instruments. [Accessed Nov. 2, 2014]. [Online]. Available: http://www.ti.com/tool/ccstudio

[27] A. Rahimi and A. Emadi, "Discontinuous-conduction mode dc/dc converters feeding constant-power loads," *IEEE Transactions on Industrial Electronics*, vol. 57, no. 4, pp. 1318–1329, April 2010.

[28] L. Siguang and Z. Chengning, "Study on battery management system and lithium-ion battery," in *International Conference on Computer and Automation Engineering, 2009*, March 2009, pp. 218–222.

[29] Z. He, G. Yang, H. Geng, N. Shen, and Z. Wang, "A battery modeling method and its verification in discharge curves of lead-acid batteries," in *IEEE Vehicle Power and Propulsion Conference*, Oct 2013, pp. 1–5.

[30] M. Ichimura, T. Horie, K. Yotsumoto, K. Takano, Y. Konya, and Y. Koizumi, "Measuring the internal resistance of a cell in assembled batteries," in *18th International Telecommunications Energy Conference*, Oct 1996, pp. 784–791.

[31] A. F. Burke, "Batteries and ultracapacitors for electric, hybrid, and fuel cell vehicles," *Proceedings of the IEEE*, vol. 95, no. 4, pp. 806–820, April 2007.

[32] *K2 Series Ultracapacitor Datasheet*, Maxwell Technologies, 2013, [Accessed Oct. 2, 2014]. [Online]. Available: http://www.maxwell.com/images/documents/k2series_ds_10153704.pdf

[33] C. Phillips and R. Harbor, Eds., *Feedback Control Systems.* Upper Saddle River, NJ: Prentice Hall, 2000.

[34] R. Vilanova and O. Arrieta, "Pid tuning for cascade control system design," in *Canadian Conference on Electrical and Computer Engineering*, May 2008, pp. 1775–1778.

[35] J. Marshall, M. Kazerani, and R. Shatshat, "Investigation of membership function shapes in a fuzzy-controlled hvdc system," in *IEEE International Symposium on Industrial Electronics*, vol. 3, July 2006, pp. 1800–1805.

[36] C. Moler. (2003) Stiff differential equations. Mathworks, Inc. [Accessed Nov. 2, 2014]. [Online]. Available: http://www.mathworks.com/company/newsletters/articles/stiff-differential-equations.html

[37] C. L. Phillips and R. D. Harbor, *Feedback Control Systems.* Upper Saddle River, NJ: Prentice Hall, 2000.

[38] *Sealed Lead-Acid Batteries - Technical Manual*, Power-Sonic Corporation, 2009, [Accessed Oct. 21, 2014]. [Online]. Available: https://www.ken-lab.com/attachments/article/13/powersonicmanual.pdf

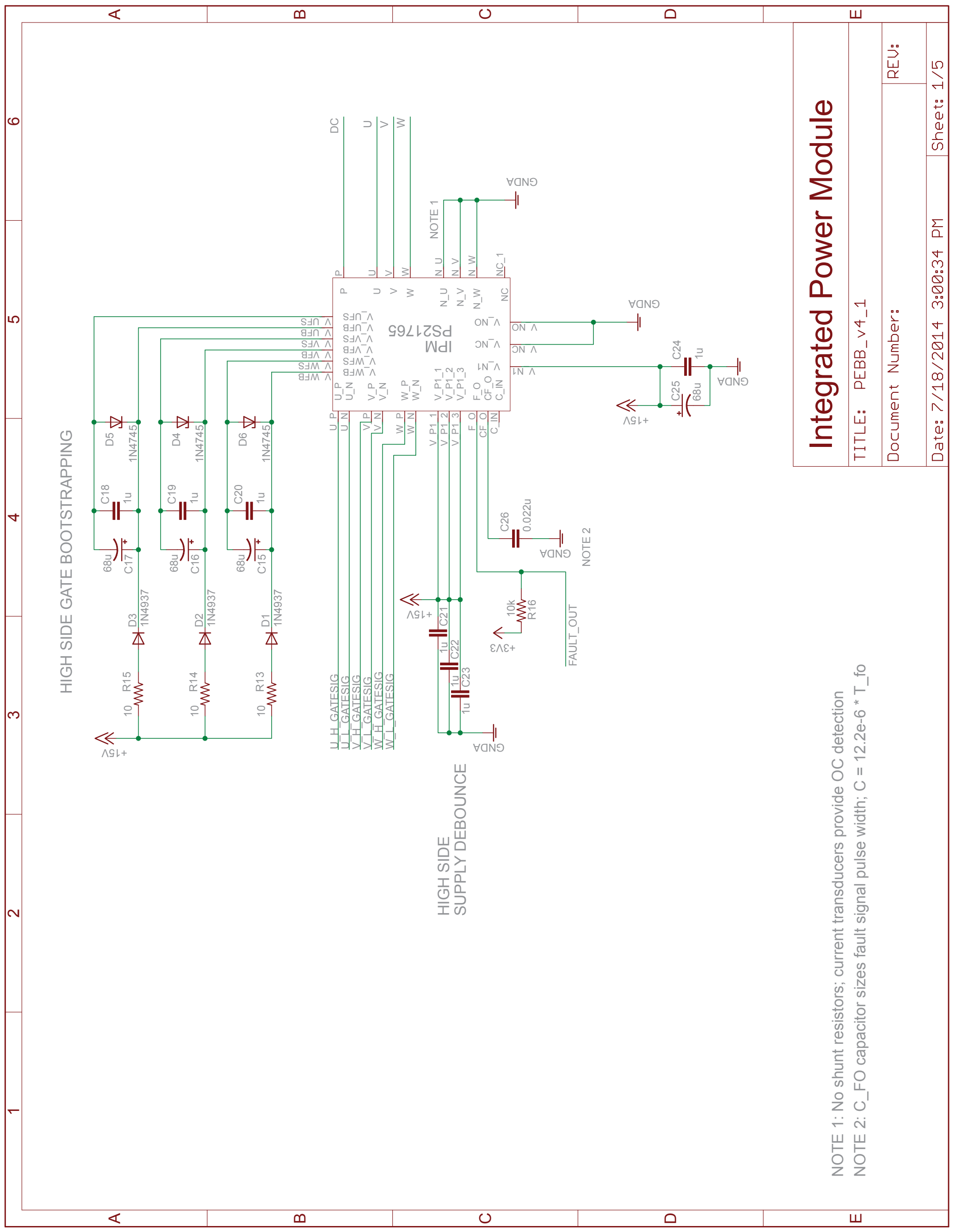
[39] D. Vutetakis and H. Wu, "The effect of charge rate and depth of discharge on the cycle life of sealed lead-acid aircraft batteries," in *IEEE International Power Sources Symposium*, June 1992, pp. 103–105.

[40] N. Kutkut, D. Divan, and D. Novotny, "Charge equalization for series connected battery strings," *IEEE Transactions on Industry Applications*, vol. 31, no. 3, pp. 562–568, May 1995.

Appendix A

Hardware Design Materials

A.1 Power Inverter - Schematic



Integrated Power Module
TITLE: PEBB_v4_1
Document Number:
REV:
Date: 7/18/2014 3:00:34 PM
Sheet: 1/5
HIGH SIDE GATE BOOTSTRAPPING
HIGH SIDE SUPPLY DEBOUNCE
IPM PS21765
NOTE 1: No shunt resistors; current transducers provide OC detection
NOTE 2: C_FO capacitor sizes fault signal pulse width; C = 12.2e-6 * T_fo
FAULT_OUT
GNDA
+15V
+3V3
DC
U
V
W
U_H_GATESIG
U_L_GATESIG
V_H_GATESIG
V_L_GATESIG
W_H_GATESIG
W_L_GATESIG

Figure A.1: Power Inverter Schematic, Page 1 of 5

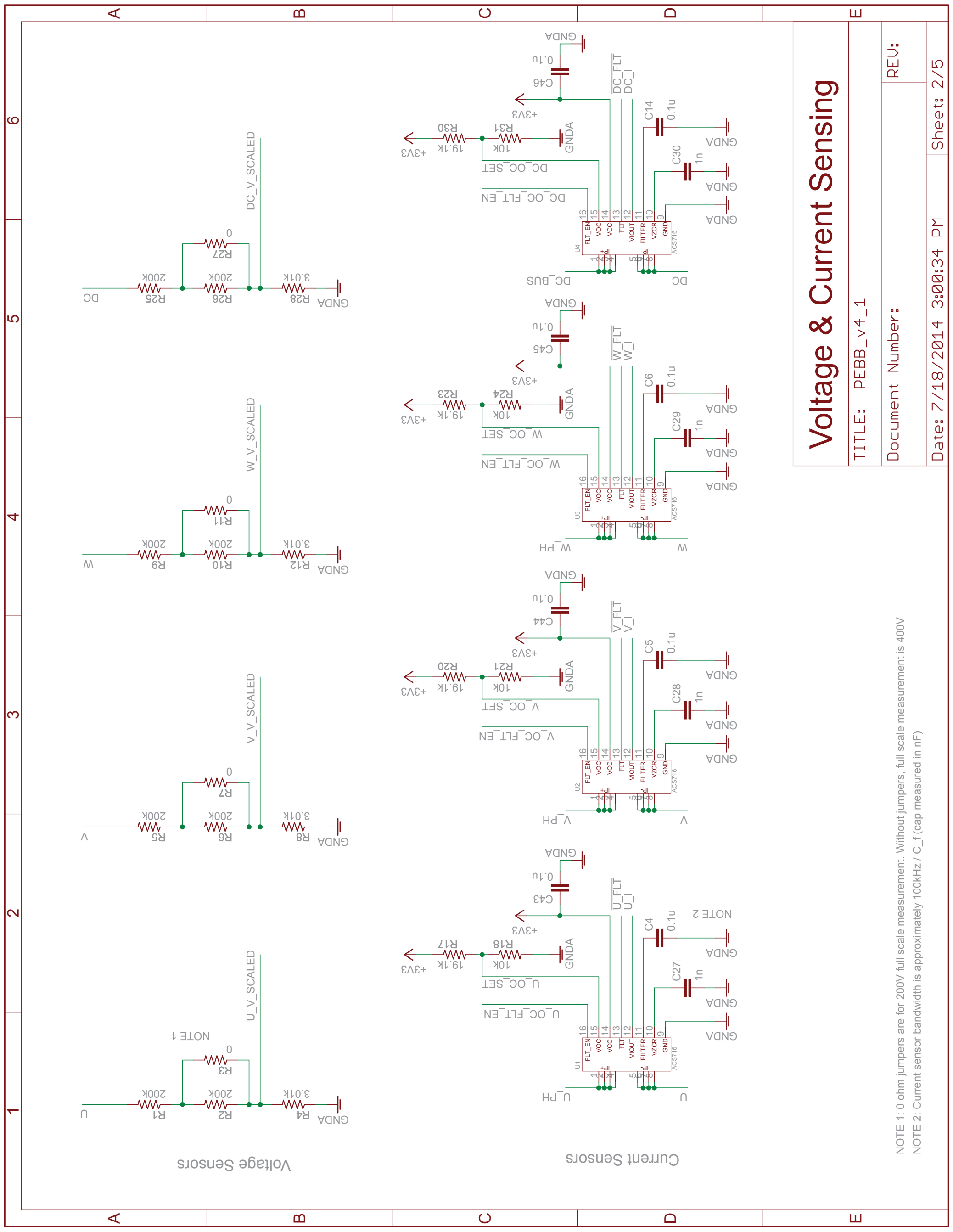

Voltage & Current Sensing
TITLE: PEBB_v4_1
Document Number:
REV:
Date: 7/18/2014 3:00:34 PM
Sheet: 2/5
Voltage Sensors
Current Sensors
U_V_SCALED
V_V_SCALED
W_V_SCALED
DC_V_SCALED
NOTE 1: 0 ohm jumpers are for 200V full scale measurement. Without jumpers, full scale measurement is 400V
NOTE 2: Current sensor bandwidth is approximately 100kHz / C_f (cap measured in nF)
ACS716
GNDA
+3V3

Figure A.2: Power Inverter Schematic, Page 2 of 5

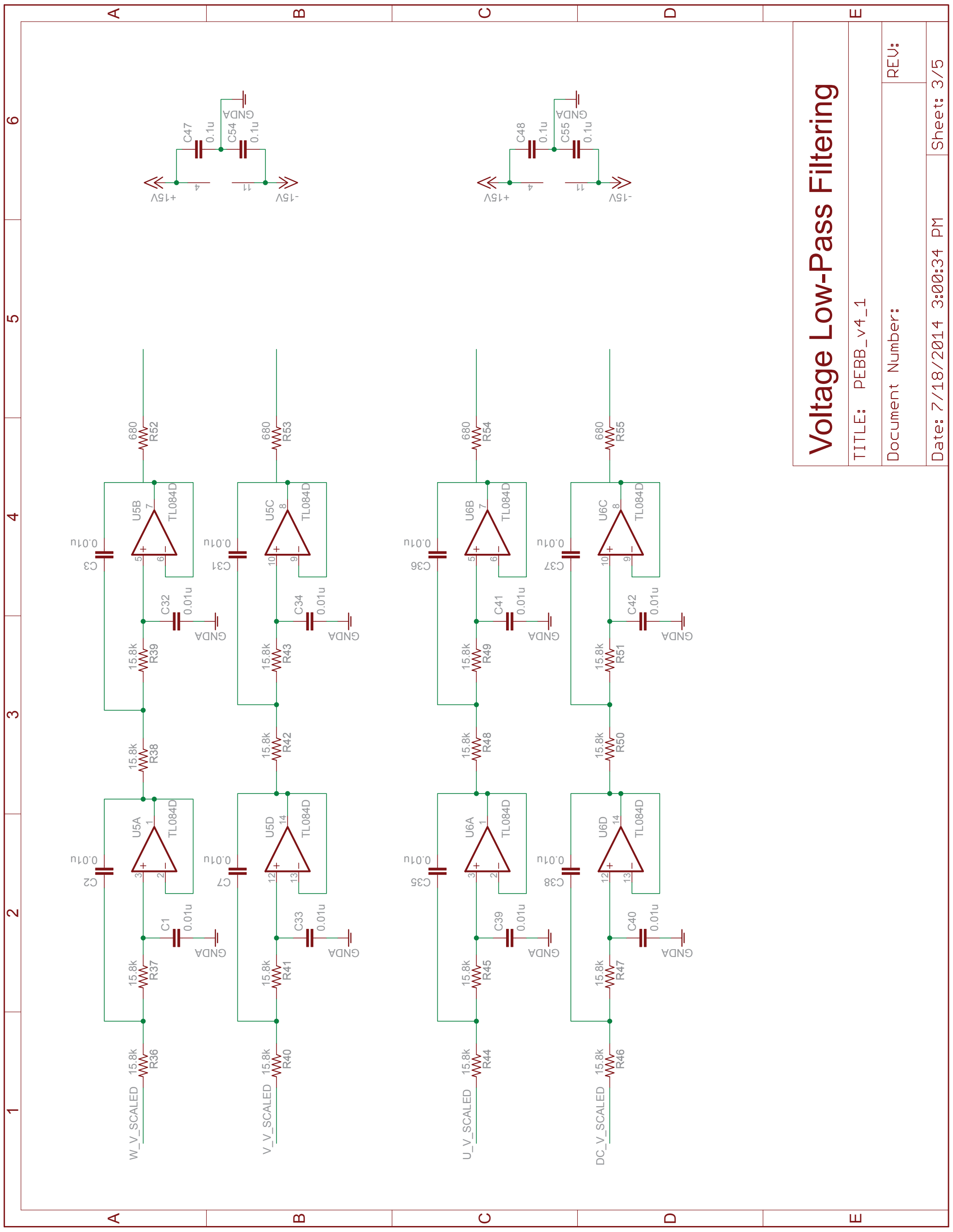


Figure A.3: Power Inverter Schematic, Page 3 of 5

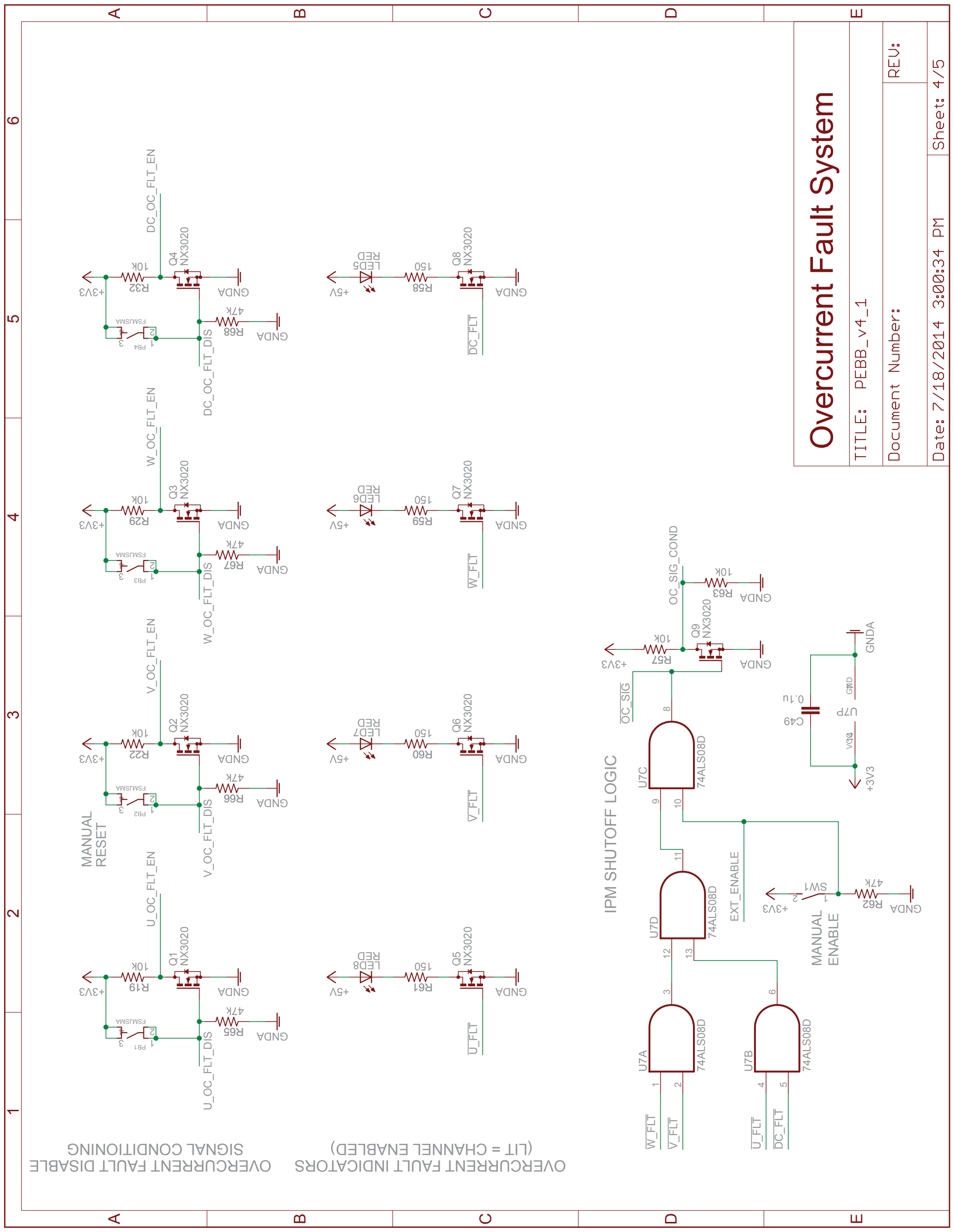


Figure A.4: Power Inverter Schematic, Page 4 of 5

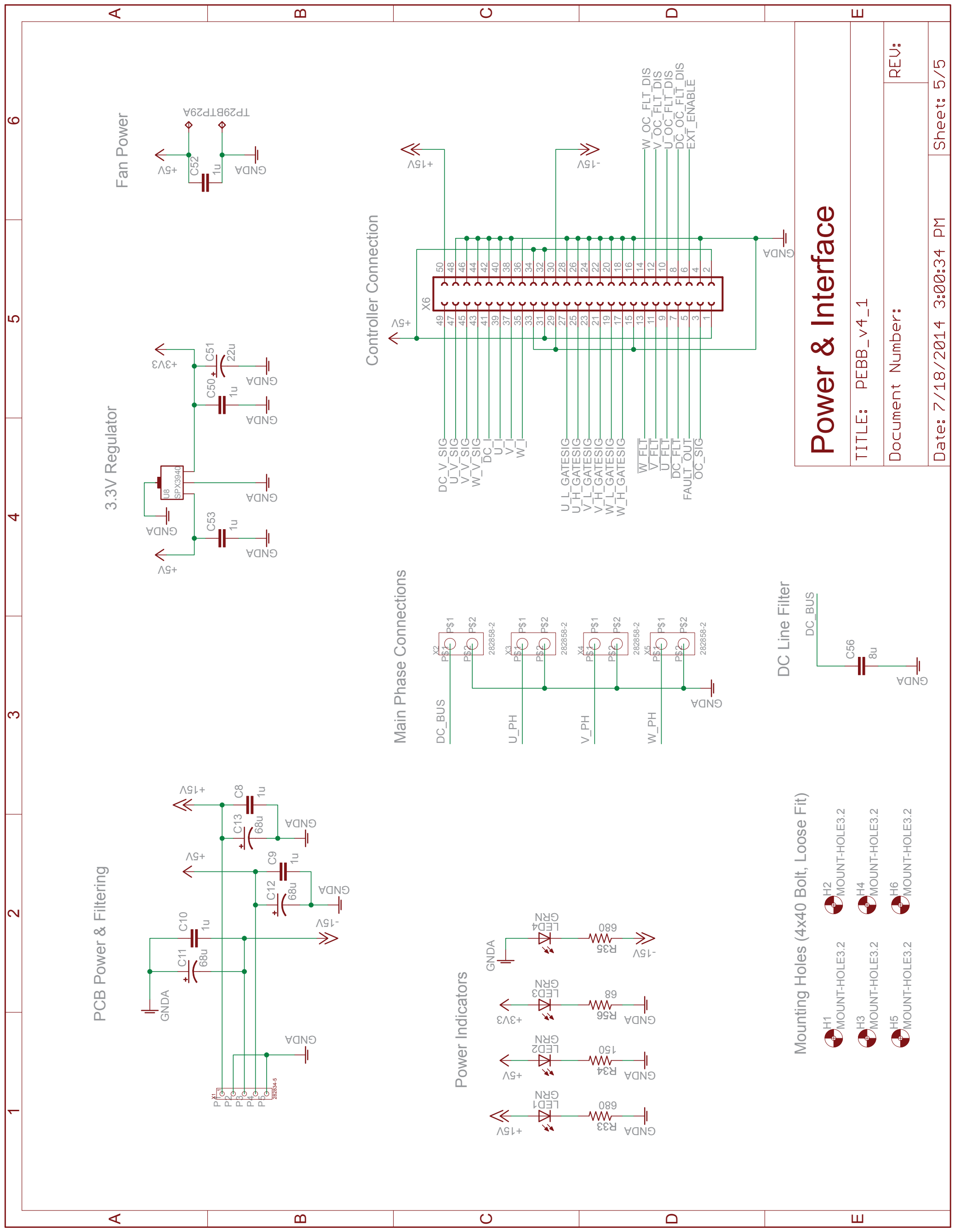


Figure A.5: Power Inverter Schematic, Page 5 of 5

A.2 Power Inverter - PCB Layout

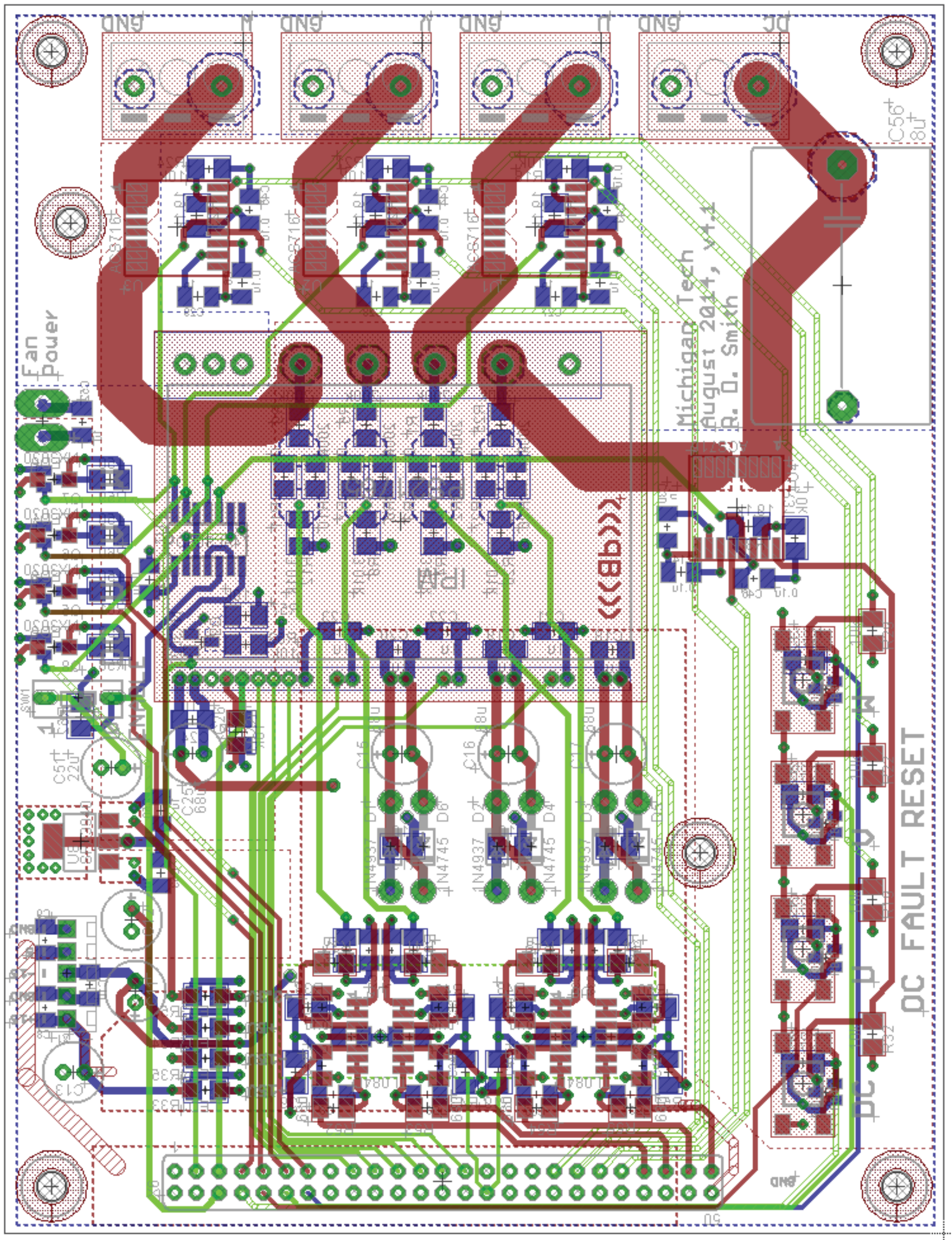


Figure A.6: Power Inverter PCB Layout

A.3 Inverter Controller - Schematic

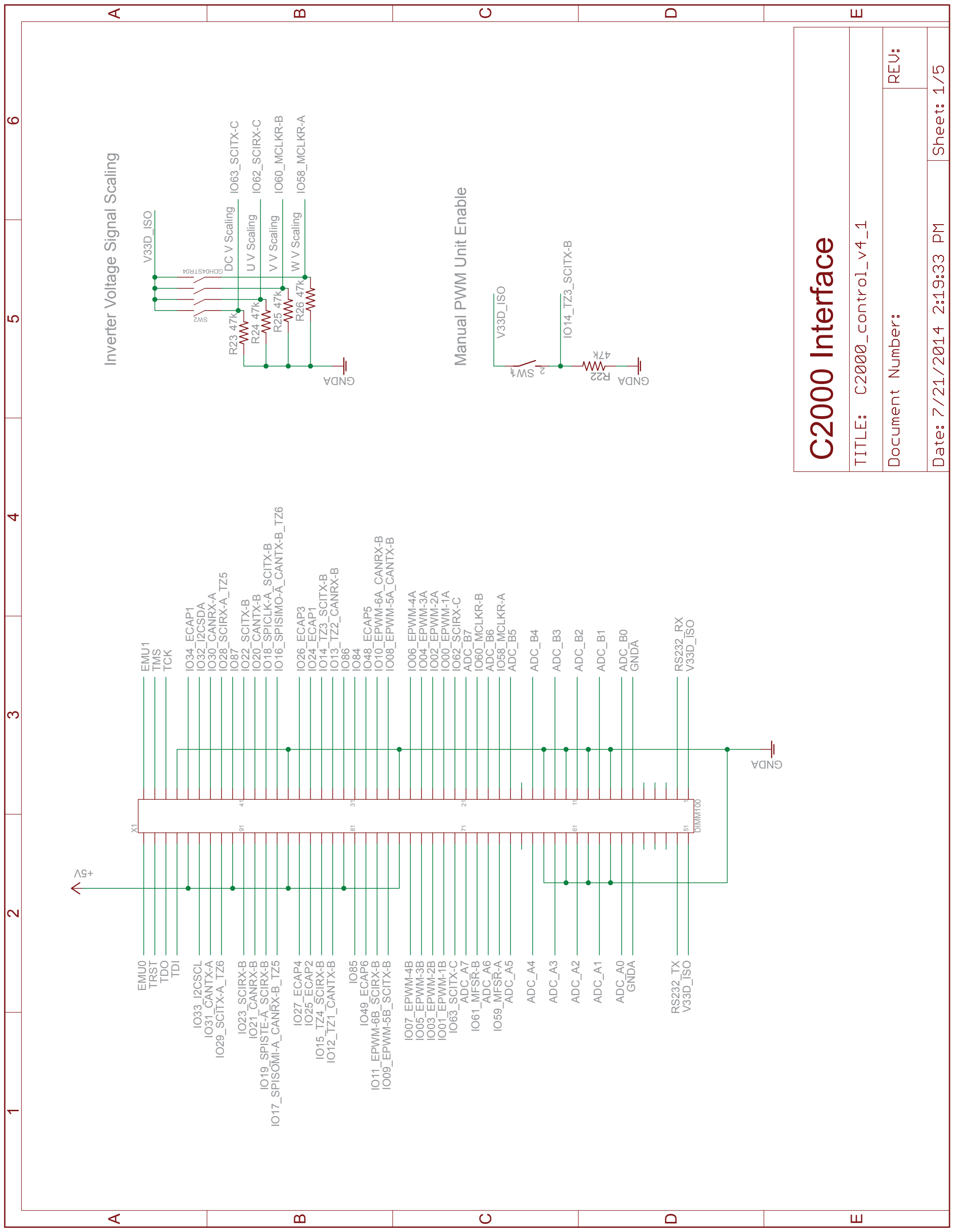


Figure A.7: Inverter Controller Schematic, Page 1 of 5

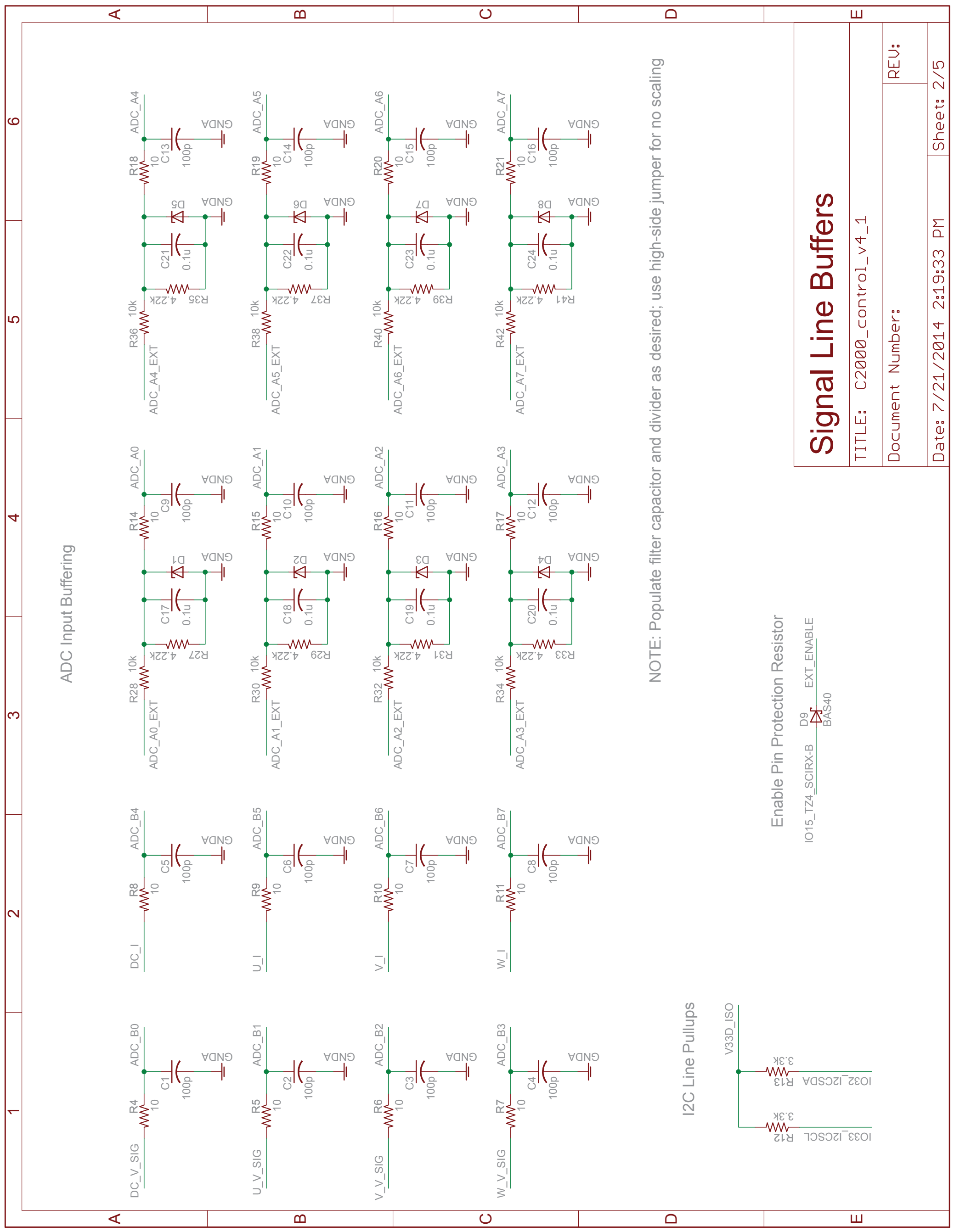


Figure A.8: Inverter Controller Schematic, Page 2 of 5

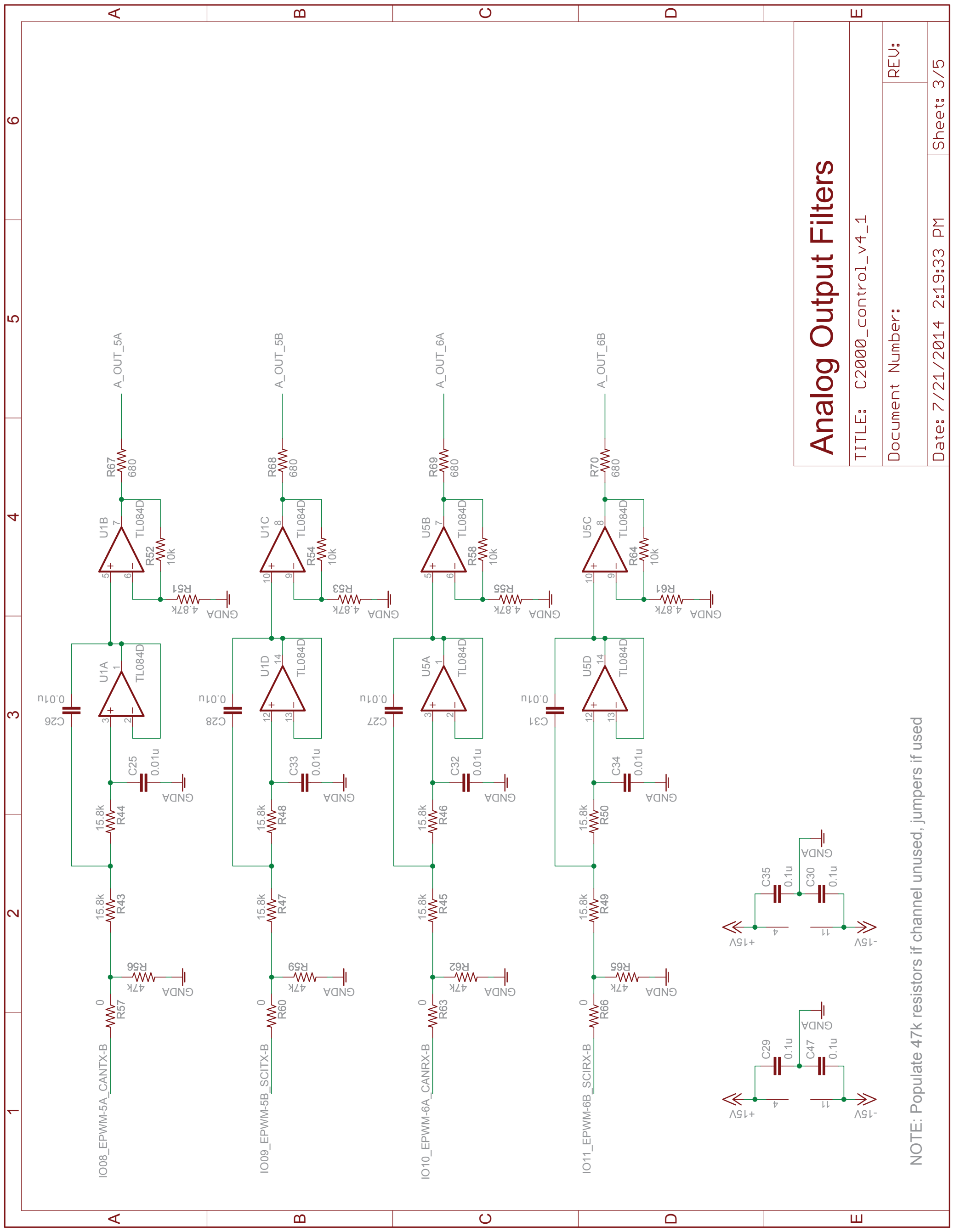


Figure A.9: Inverter Controller Schematic, Page 3 of 5

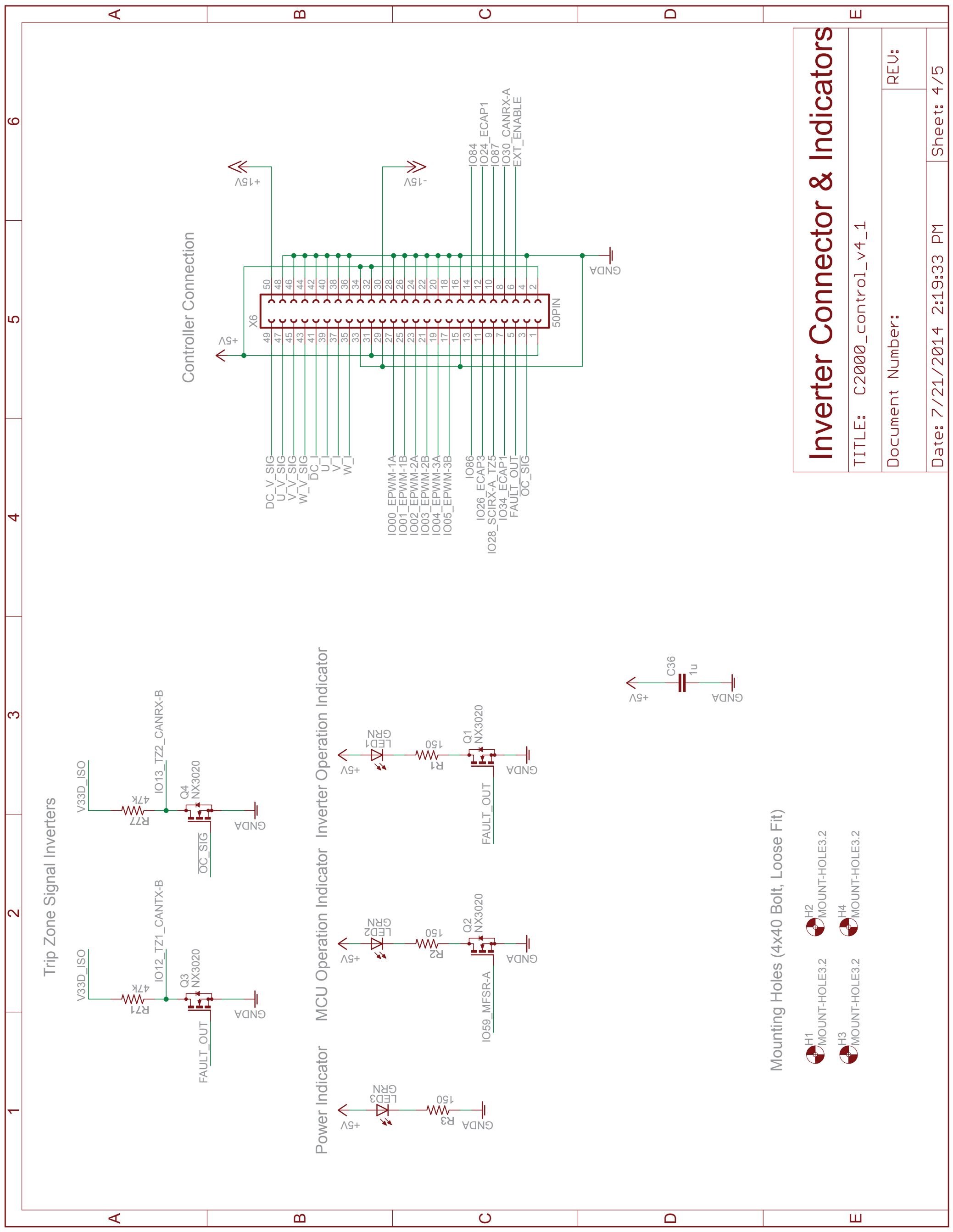


Figure A.10: Inverter Controller Schematic, Page 4 of 5

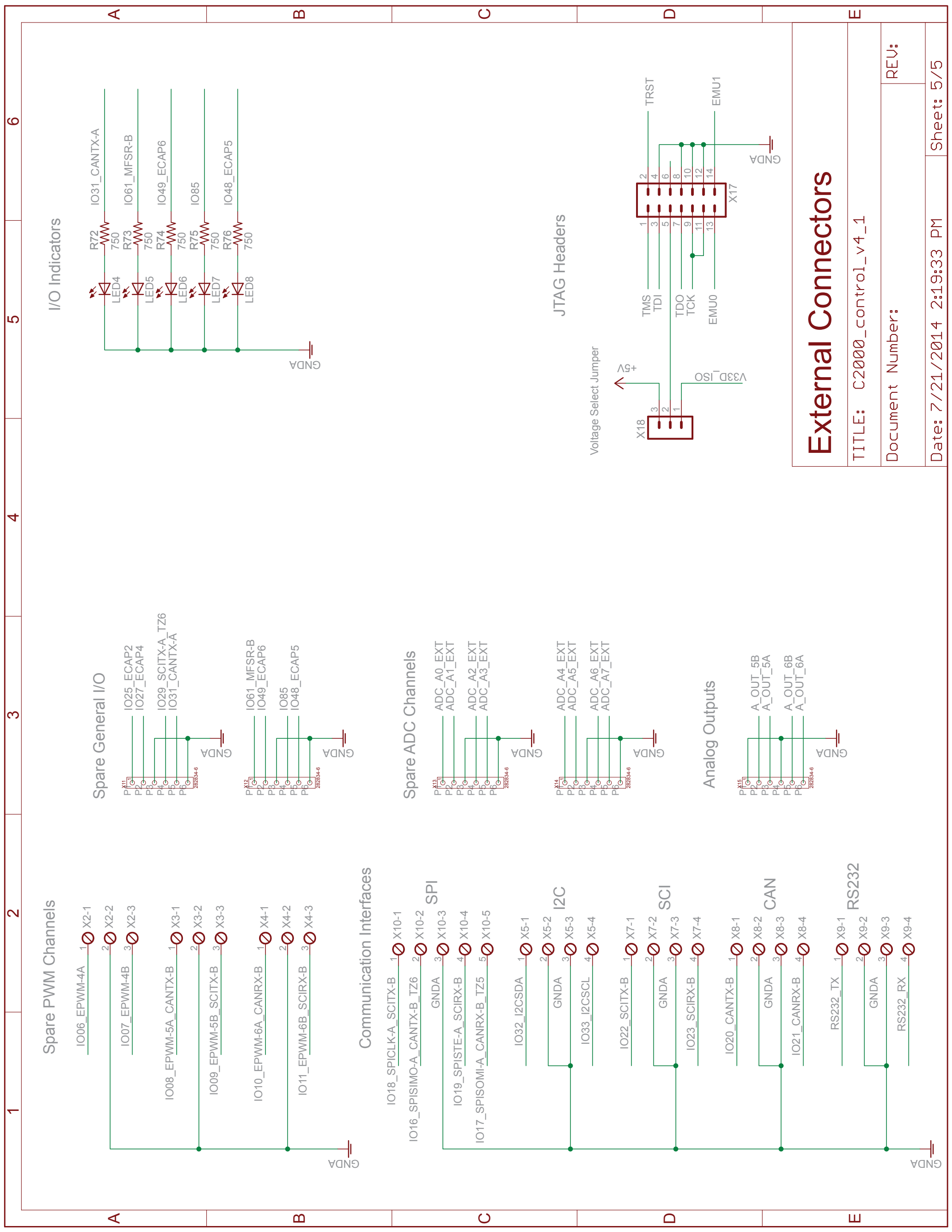


Figure A.11: Inverter Controller Schematic, Page 5 of 5

A.4 Inverter Controller - PCB Layout

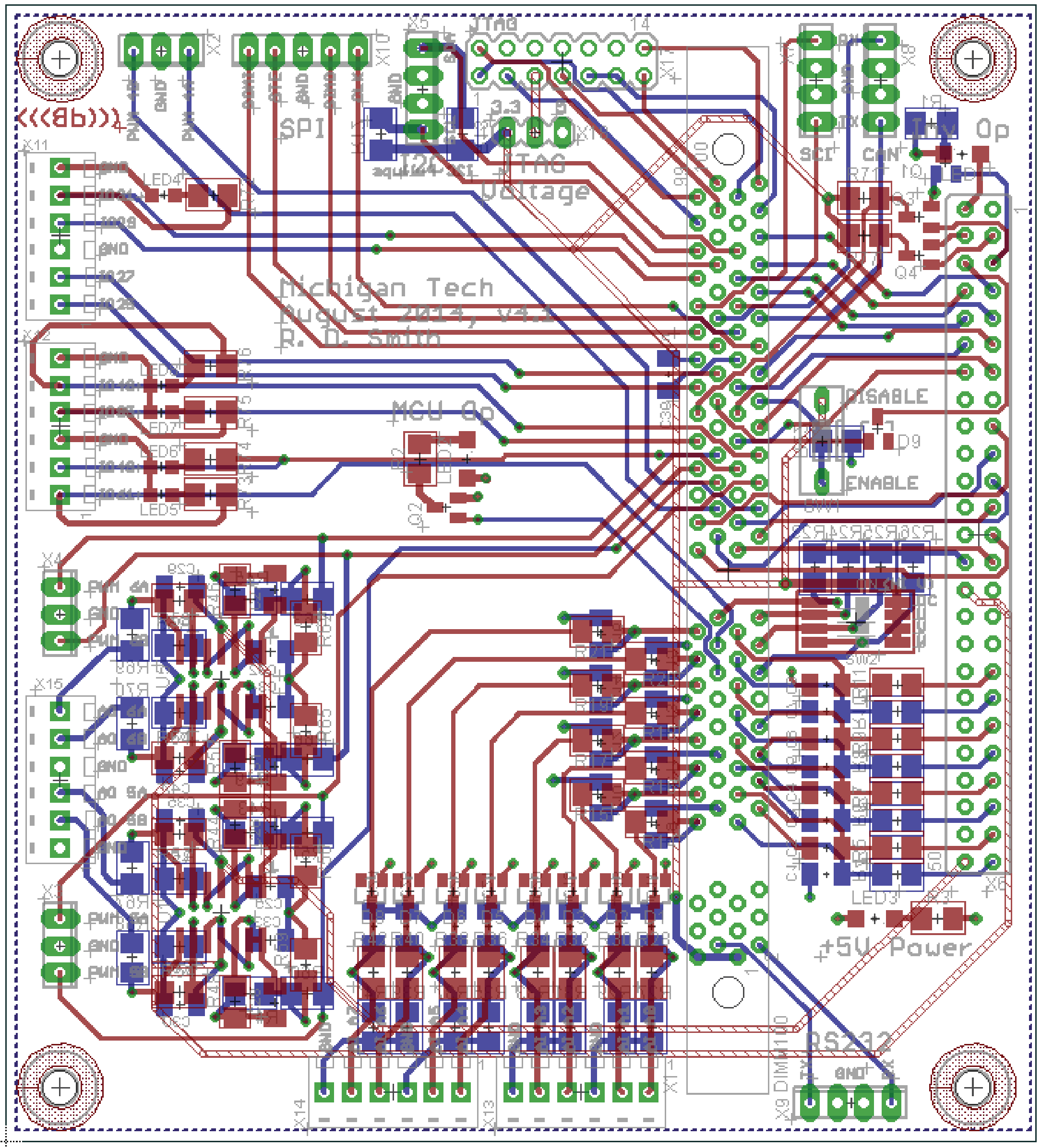


Figure A.12: Inverter Controller PCB Layout

Appendix B

Detailed Simulink Diagrams

B.1 Random Source

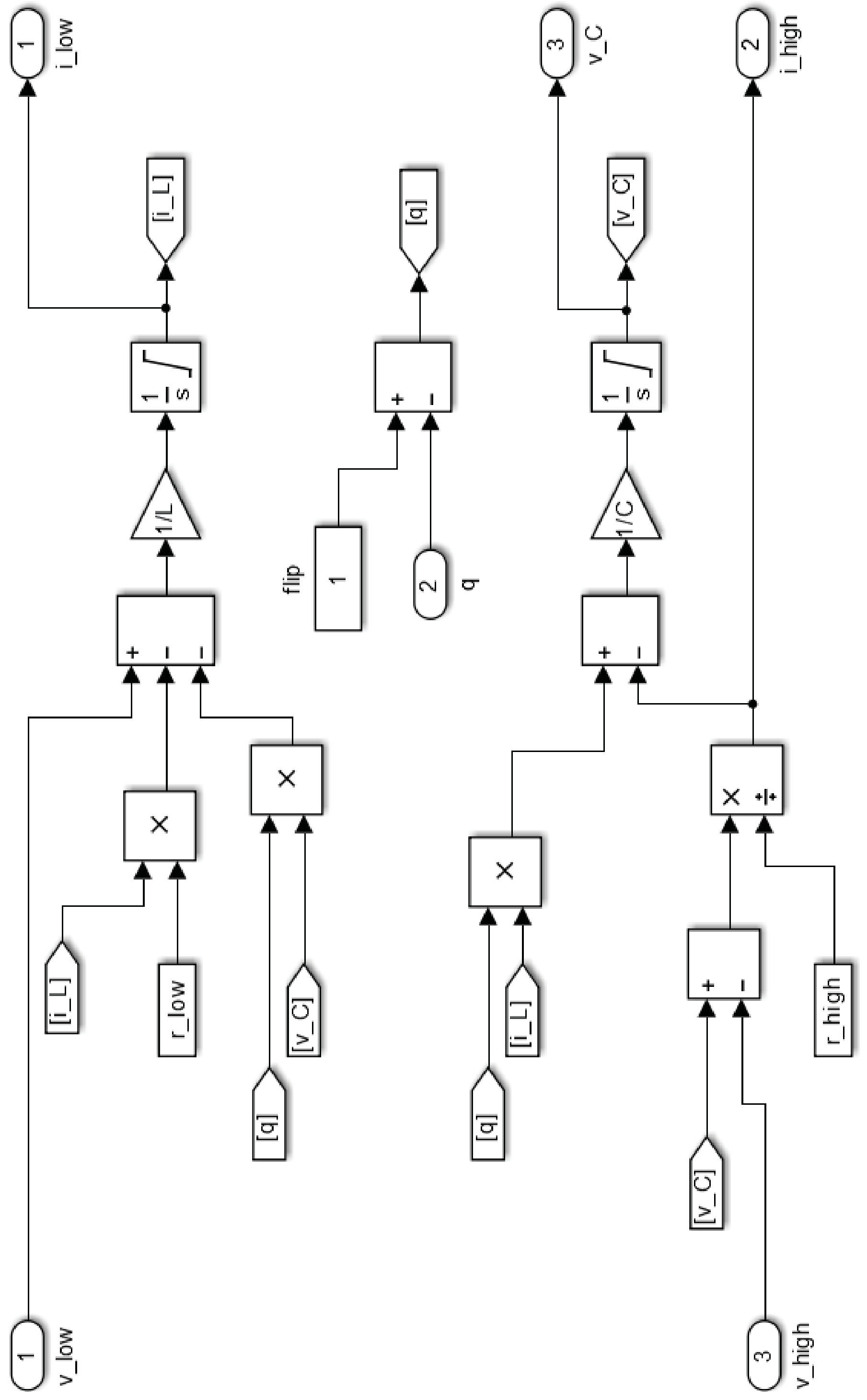


Figure B.1: Random Source - Inner Simulink Construction

B.2 Stabilizer

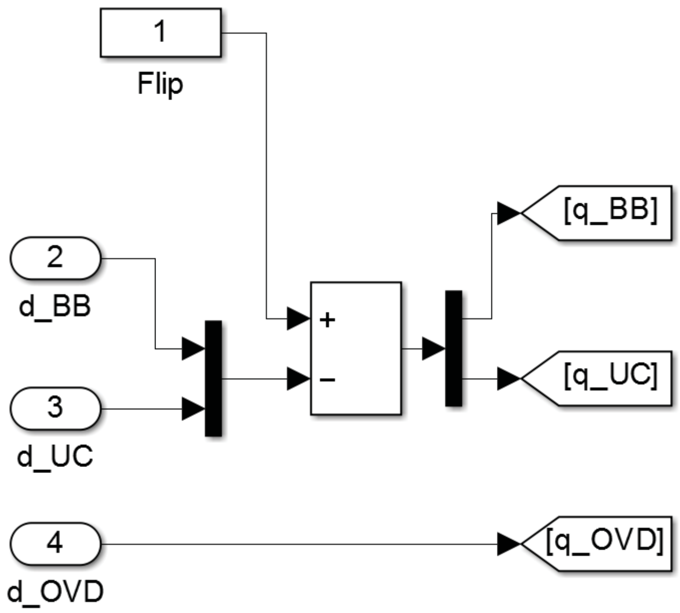


Figure B.2: Stabilizer - Inner Simulink Construction - Input Duty Signals

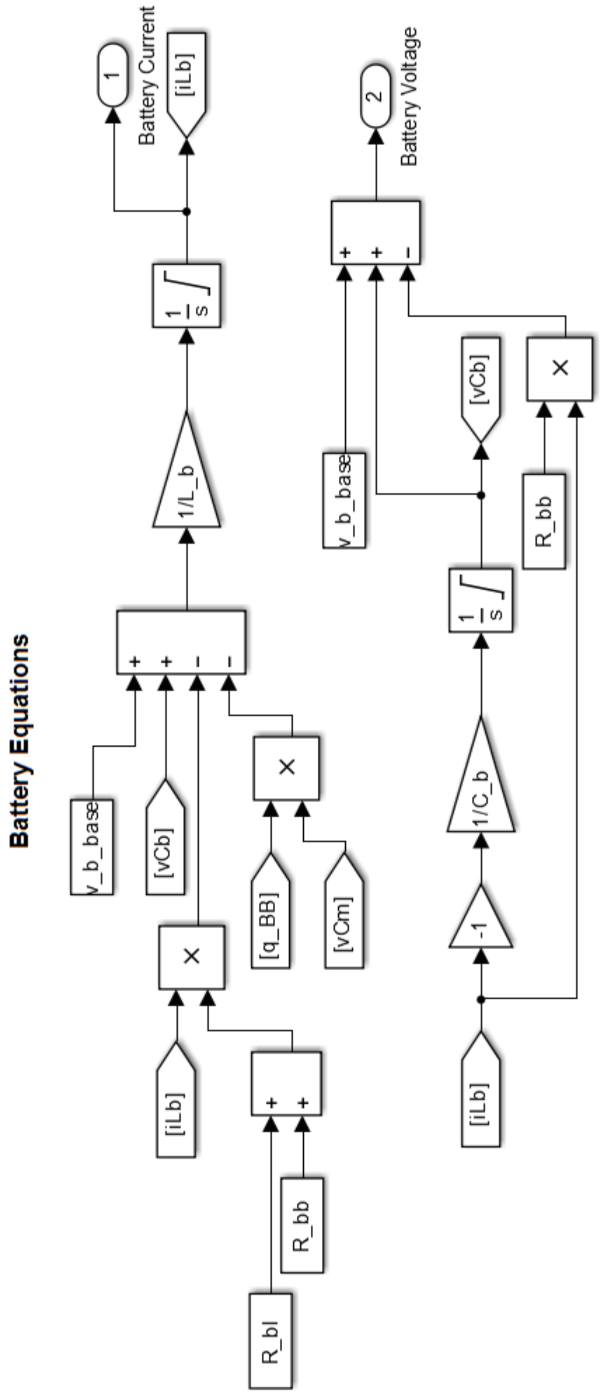


Figure B.3: Stabilizer - Inner Simulink Construction - Battery Bank

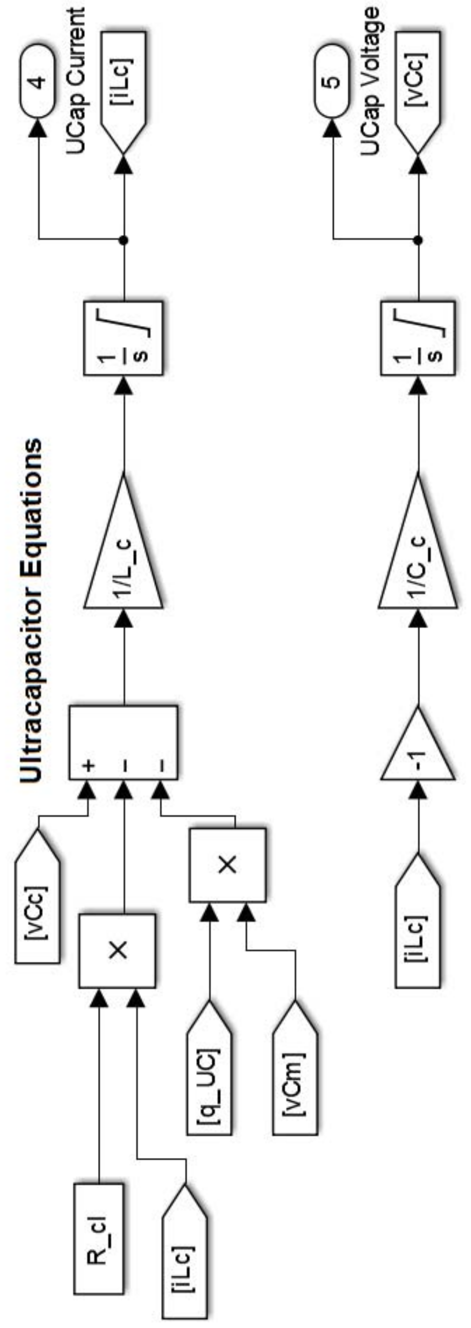


Figure B.4: Stabilizer - Inner Simulink Construction - Ultracapacitor Bank

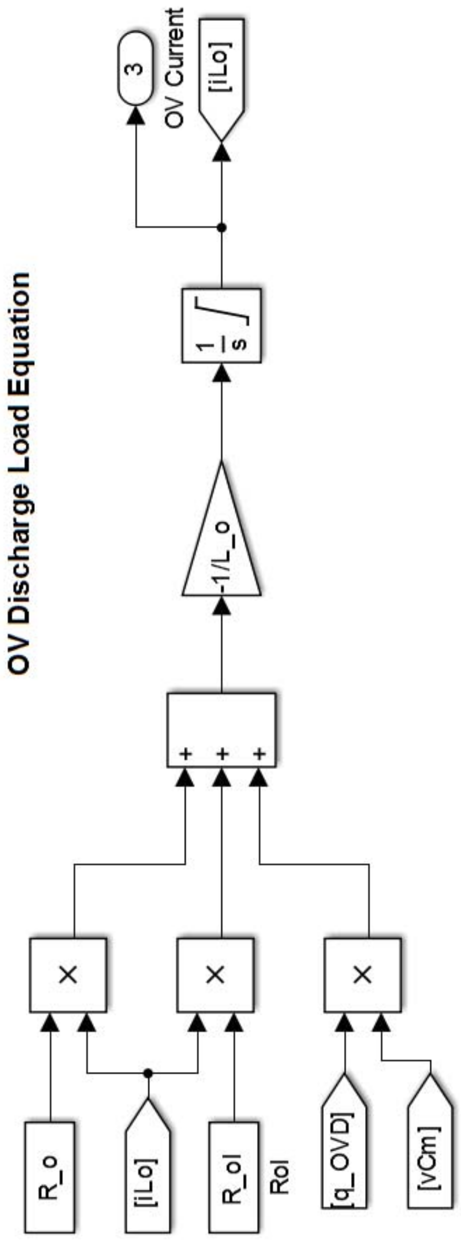


Figure B.5: Stabilizer - Inner Simulink Construction - Overvoltage Discharge Phase

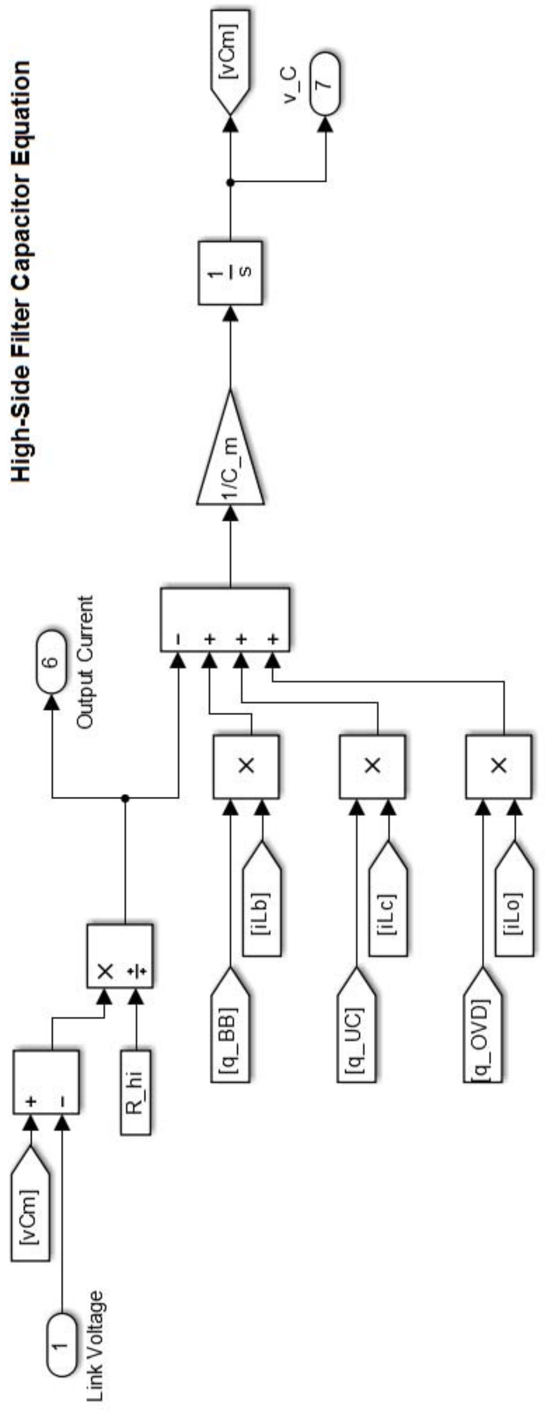


Figure B.6: Stabilizer - Inner Simulink Construction - Current Summation and High-Side Capacitor

B.3 PI Control System

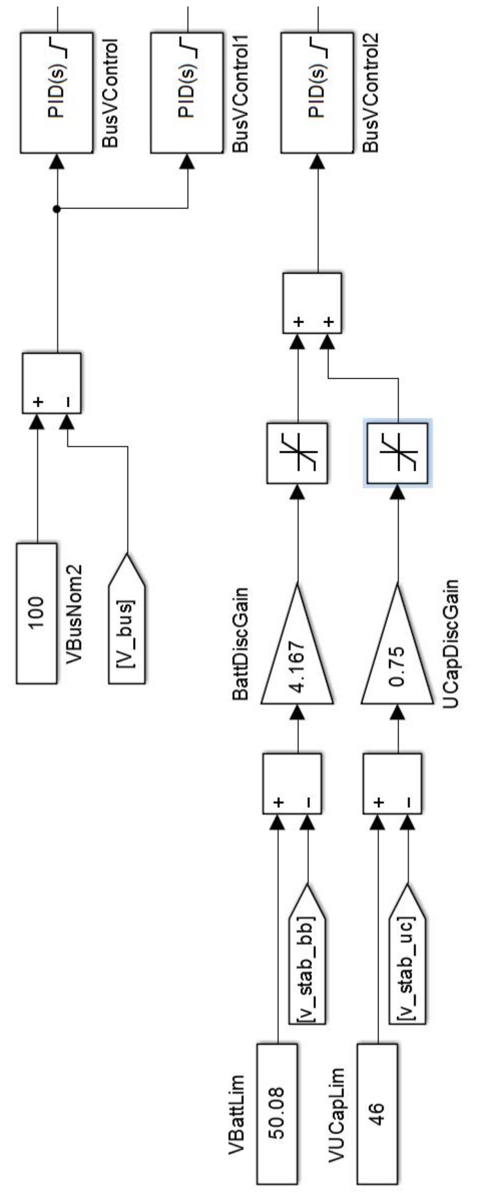


Figure B.7: Outer Control - PI System

B.4 Fuzzy Logic Control System

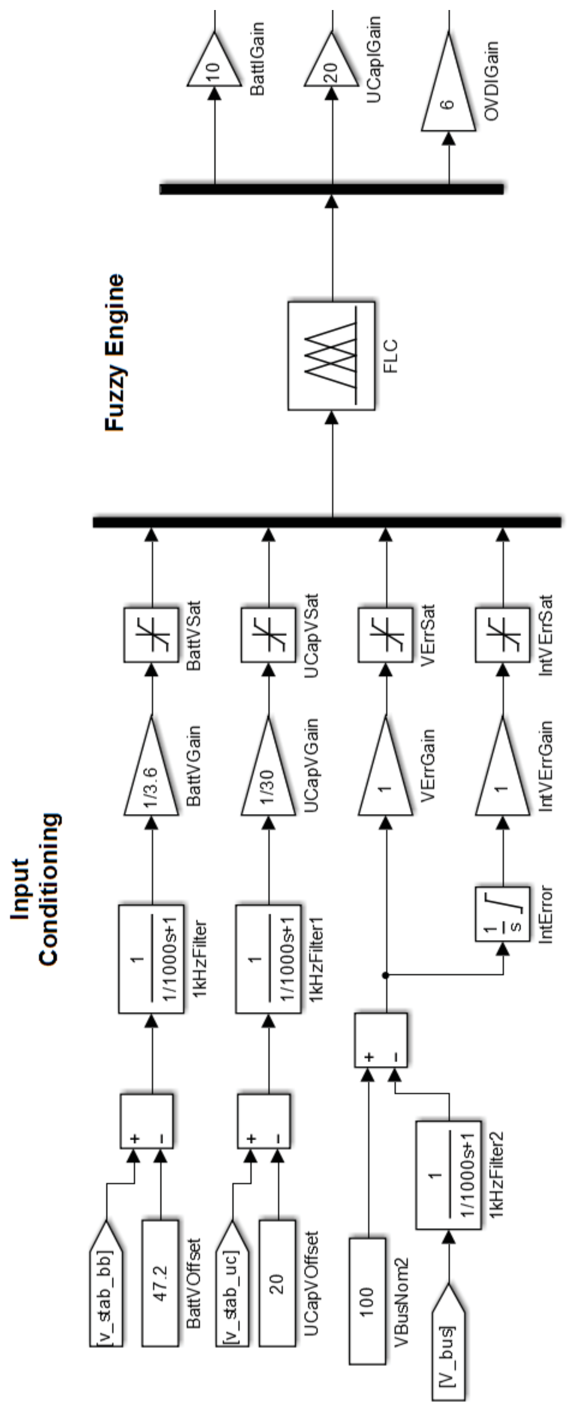


Figure B.8: Outer Control - FLC System

Appendix C

PI Control System - Experimental Implementation

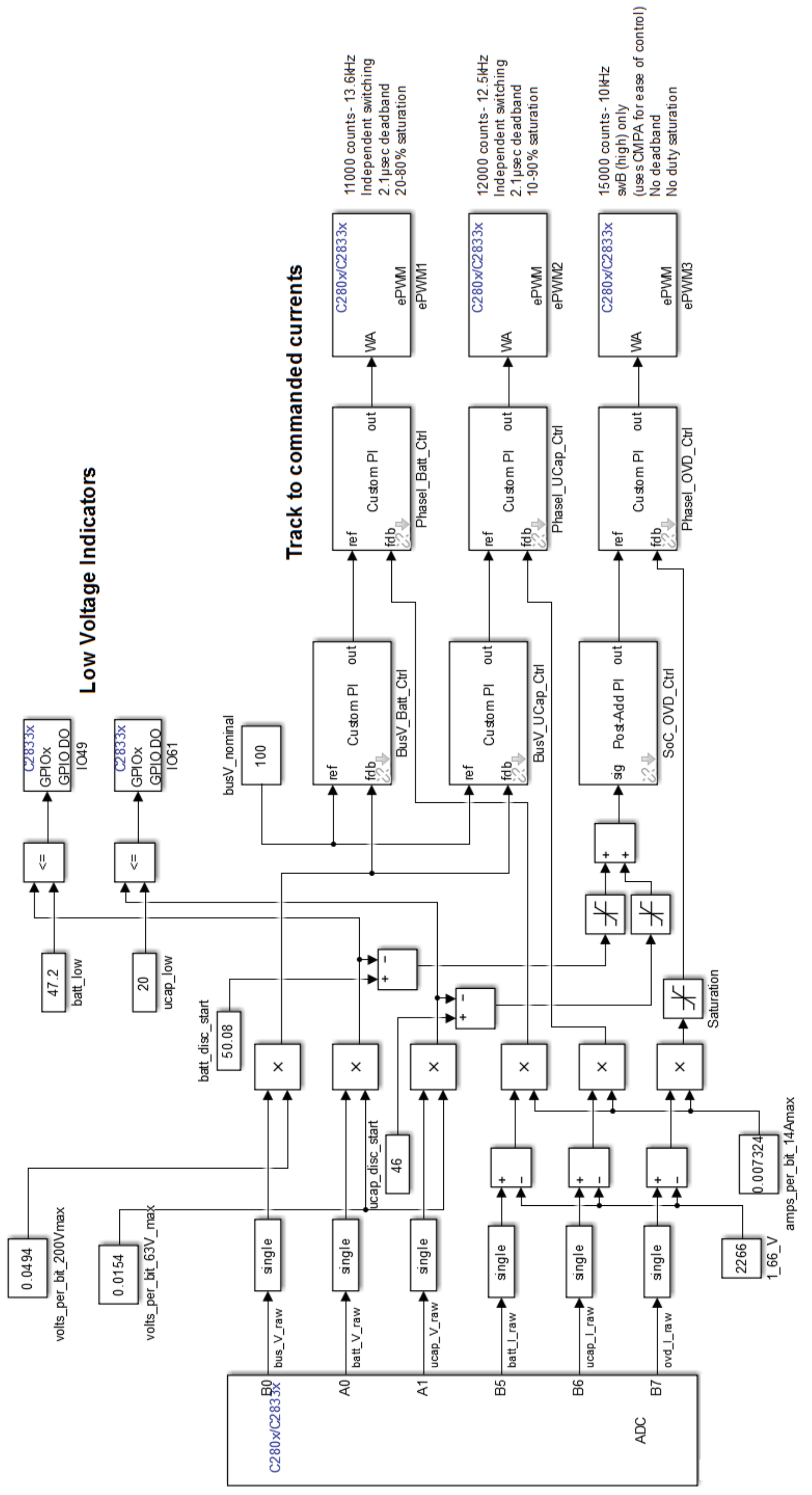


Figure C.1: PI Control - Experimental Simulink Implementation

Appendix D

Fuzzy Logic Control System - Experimental Implementation

The code given here is the program modified from the PI control system to form the FL control system.

```
/*
 * File: stab_fuzzy.c
 *
 * Code generated for Simulink model 'stab_fuzzy'.
 *
 * Model version                  : 1.58
 * Simulink Coder version         : 8.7 (R2014b) 08-Sep-2014
 * C/C++ source code generated on : Wed Nov 12 18:55:20 2014
 *
 * Target selection: ert.tlc
 * Embedded hardware selection: Texas Instruments->C2000
 * Code generation objectives: Unspecified
 * Validation result: Not run
 */

// ============================================================================
// CUSTOM DEFINITIONS

#define BVERR_FULL 0.8
#define BVERR_OFF 0.6

#define INTBVERR_GAIN 0.0002     // sampling time
#define INTBVERR_FULL 0.8
#define INTBVERR_OFF 0.4

#define UCAPV_MIN 20.0
#define UCAPV_MID_L 26.0
#define UCAPV_MID_R 44.0
#define UCAPV_MAX 50.0

#define BATTV_MIN 47.2
#define BATTV_MID_L 48.28
#define BATTV_MID_R 49.72
#define BATTV_MAX 50.8

// current output membership function bases
#define BATTI_VX_B 7.200

#define UCAPI_VX_B 12.000
#define UCAPI_X_B 16.000
#define UCAPI_DIAG_F 64.000
```

```
#define OVDI_L_B 6.000

#define OVDI_H_B 12.000



#define BUSV_NOM 100.0


// membership function vertical center at threshold is negligible (~ 5e-5)

#define THRESH 0.0001


// ============================================================================


#include "stab_fuzzy.h"

#include "stab_fuzzy_private.h"


/* Block signals (auto storage) */

B_stab_fuzzy_T stab_fuzzy_B;


/* Block states (auto storage) */

DW_stab_fuzzy_T stab_fuzzy_DW;


/* Real-time model */

RT_MODEL_stab_fuzzy_T stab_fuzzy_M_;

RT_MODEL_stab_fuzzy_T *const stab_fuzzy_M = &stab_fuzzy_M_;


// ============================================================================

// CUSTOM VERTICAL CENTER FUNCTION


// Given an input "h" as the proportional height of a truncated triangle, this

// function returns the vertical center of the shape. The return values are

// clipped for out-of-bounds inputs.

real32_T yBar(real32_T h)

{
    if (h > 1) return 0.333;                        // clip high

    if (h < 0) return 0.0;                          // clip low

    return ((h * 0.333) * (3 - 2*h) / (2 - h));     // actual output


} // end function "yBar"


// ===========================================================================




/* Model step function */

void stab_fuzzy_step(void)

{

    /* local block i/o variables */

    real32_T rtb_Saturation_b;

    real32_T rtb_Saturation_a;

    real32_T rtb_Saturation_m;

    real32_T rtb_Product8;

    real32_T rtb_Product7;

    real32_T rtb_Product9;

    real32_T rtb_Sum1_ih;

    real32_T rtb_Sum1_n;

    real32_T rtb_Sum1;

    real32_T rtb_Sum1_l3;

    real32_T rtb_Sum1_c4;

    real32_T rtb_Sum1_c;

    real32_T rtb_Sum1_j;

    real32_T rtb_Sum1_i;


    // =======================================================================

    // CUSTOM VARIABLES


    // *** input and filter variables ******************************

    static uint16_T bus_V_filt = 0;
    static uint16_T batt_V_filt = 0;

    static uint16_T ucap_V_filt = 0;

    static uint16_T batt_I_filt = 0;

    static uint16_T ucap_I_filt = 0;

    static uint16_T ovd_I_filt = 0;


    real32_T busVE = 0;

    real32_T battV = 0;

    real32_T ucapV = 0;


    // *** heartbeat memory ****************************************

    static uint16_T tog = 2;


    // *** fuzzy input variables ***********************************

    static real32_T bverr_int = 0;  // bus voltage error accumulator


    real32_T battv_L = 0;           // battery voltage membership levels

    real32_T battv_H = 0;


    real32_T ucapv_L = 0;           // ultracapacitor voltage membership levels

    real32_T ucapv_H = 0;


    real32_T busverr_P = 0;         // bus voltage error membership levels

    real32_T busverr_N = 0;


    real32_T intbusverr_P = 0;      // int. bus voltage error membership levels

    real32_T intbusverr_N = 0;


    // *** fuzzy output variables **********************************

    // initialize output weights to small, positive numbers to prevent /0 errors

    real32_T batti_VN = 0;          // battery current membership levels

    real32_T batti_N = 0;

    real32_T batti_AZ = 0;
```

```
real32_T batti_P = 0;

real32_T batti_VP = 0;


real32_T batti_pos = 0;         // battery current trigger accumulation

real32_T batti_wt = 0.001;

real32_T batti_ref = 0;         // battery current reference output


real32_T ucapi_VN = 0;          // ultracapacitor current membership levels

real32_T ucapi_N = 0;

real32_T ucapi_P = 0;

real32_T ucapi_VP = 0;


real32_T ucapi_pos = 0;         // ultracapacitor current trigger accumulation

real32_T ucapi_wt = 0.001;

real32_T ucapi_ref= 0;          // ultracapacitor current reference output


real32_T ovdi_L = 0;            // OV discharge current membership levels

real32_T ovdi_H = 0;


real32_T ovdi_pos = 0;          // OV discharge current trigger accumulation

real32_T ovdi_wt = 0.01;

real32_T ovdi_ref = 0;          // OV discharge current reference output


real32_T temp = 0;              // reduce number of calls to "yBar" function


// ======================================================================




/* S-Function (c280xadc): '<Root>/TheAlmightyADC' */

{

    AdcRegs.ADCTRL2.bit.RST_SEQ1 = 1;/* Reset SEQ1 module*/

    AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1;/*clear INT sequencer*/

    AdcRegs.ADCTRL2.bit.SOC_SEQ1 = 1;/* Software Trigger*/
    while (AdcRegs.ADCST.bit.INT_SEQ1 == 0) {

    }                              /*Wait for Sequencer INT bit to clear */


    asm(" RPT #11 || NOP");

    stab_fuzzy_B.bus_V_raw = (AdcRegs.ADCRESULT0) >> 4;

    stab_fuzzy_B.batt_V_raw = (AdcRegs.ADCRESULT1) >> 4;

    stab_fuzzy_B.ucap_V_raw = (AdcRegs.ADCRESULT2) >> 4;

    stab_fuzzy_B.batt_I_raw = (AdcRegs.ADCRESULT3) >> 4;

    stab_fuzzy_B.ucap_I_raw = (AdcRegs.ADCRESULT4) >> 4;

    stab_fuzzy_B.ovd_I_raw = (AdcRegs.ADCRESULT5) >> 4;

}


// ========================================================================

// CUSTOM 1-STATE IIR LOW-PASS FILTER AND INPUT CONDITIONING


// *** filter inputs ******************************************

// Experience shows that systems perform better when this filter is added.

// Filter equation:  Y[N] = Y[N-1] + X[N]  ===>  H(z) = (2-z^-1)^-1

// Approximate -3dB corner frequency: f_c = (0.12 * F_update)

// (Example: if the software update period is 0.2msec, f_c is about 600Hz)

bus_V_filt = (bus_V_filt >> 1) + (stab_fuzzy_B.bus_V_raw >> 1);

batt_V_filt = (batt_V_filt >> 1) + (stab_fuzzy_B.batt_V_raw >> 1);

ucap_V_filt = (ucap_V_filt >> 1) + (stab_fuzzy_B.ucap_V_raw >> 1);

batt_I_filt = (batt_I_filt >> 1) + (stab_fuzzy_B.batt_I_raw >> 1);

ucap_I_filt = (ucap_I_filt >> 1) + (stab_fuzzy_B.ucap_I_raw >> 1);

ovd_I_filt = (ovd_I_filt >> 1) + (stab_fuzzy_B.ovd_I_raw >> 1);


// *** convert raw ADC readings ********************************

busVE = BUSV_NOM - (real32_T)bus_V_filt * stab_fuzzy_P.volts_per_bit_200Vmax_Value;

battV = (real32_T)batt_V_filt * stab_fuzzy_P.volts_per_bit_63V_max_Value;

ucapV = (real32_T)ucap_V_filt * stab_fuzzy_P.volts_per_bit_63V_max_Value;


// *** update bus voltage error integrator and saturate ********
```

```
bverr_int = bverr_int + busVE * INTBVERR_GAIN;

if (bverr_int > 1) bverr_int = 1;

if (bverr_int < -1) bverr_int = -1;


// ======================================================================


// ======================================================================

// COMPUTE RESULTS OF FUZZY INPUT MEMBERSHIP FUNCTIONS


// The general approach for computing membership functions is to calculate

// the actual value given the membership function. Out-of-range inputs to

// the membership function are handled by clipping the value at [0,1]. This

// is possible because each membership function has a value of 1 beyond one

// bound and 0 beyond the other, with a straight line connecting the two

// points.


// *** fuzzy input - battery voltage - low *********************

// battv_L = 1 - (battV - BATTV_MIN)/(BATTV_MID_L - BATTV_MIN);

// battv_L = 1 - (battV - 47.2)/(1.08);

battv_L = 44.7037 - 0.9259*battV;           // calculate

if (battv_L < 0) battv_L = 0;               // clip

else if (battv_L > 1) battv_L = 1;


// *** fuzzy input - battery voltage - high ********************

// battv_H = (battV - BATTV_MID_R)/(BATTV_MAX - BATTV_MID_R);

// battv_H = (battV - 49.72)/(1.08);

battv_H = 0.9259*battV - 46.037;            // calculate

if (battv_H < 0) battv_H = 0;               // clip

else if (battv_H > 1) battv_H = 1;


// *** fuzzy input - ultracapacitor voltage - low **************

// ucapv_L = 1 - (ucapV - UCAPV_MIN)/(UCAPV_MID_L - UCAPV_MIN);

// ucapv_L = 1 - (ucapV - 20)/(6);
```

```
ucapv_L = 4.333 - 0.1667*ucapV;            // calculate
if (ucapv_L < 0) ucapv_L = 0;              // clip
else if (ucapv_L > 1) ucapv_L = 1;

// *** fuzzy input - ultracapacitor voltage - high *************
// ucapv_H = (ucapV - UCAPV_MID_R)/(UCAPV_MAX - UCAPV_MID_R);
// ucapv_H = (ucapV - 44)/(6);
ucapv_H = 0.1667*ucapV - 7.333;            // calculate
if (ucapv_H < 0) ucapv_H = 0;              // clip
else if (ucapv_H > 1) ucapv_H = 1;

// *** fuzzy input - bus voltage error - positive **************
// busverr_P = (busV + BVERR_OFF ) / (BVERR_FULL + BVERR_OFF);
// busverr_P = (busV + 0.6)/(1.4);
busverr_P = 0.7143*busVE + 0.4286;         // calculate
if (busverr_P < 0) busverr_P = 0;          // clip
else if (busverr_P > 1) busverr_P = 1;

// *** fuzzy input - bus voltage error - negative **************
// busverr_N = 1 - (busV + BVERR_FULL) / (BVERR_FULL + BVERR_OFF);
// busverr_N = 1 - (busV + 0.8)/(1.4);
busverr_N = 0.4286 - 0.7143*busVE;         // calculate
if (busverr_N < 0) busverr_N = 0;          // clip
else if (busverr_N > 1) busverr_N = 1;

// *** fuzzy input - integrated bus voltage error - positive ***
// intbusverr_P = (bverr_int + INTBVERR_OFF)/(INTBVERR_FULL + INTBVERR_OFF);
// intbusverr_P = (bverr_int + 0.4)/(1.2);
intbusverr_P = 0.8333*bverr_int + 0.3333;  // calculate
if (intbusverr_P < 0) intbusverr_P = 0;    // clip
else if (intbusverr_P > 1) intbusverr_P = 1;

// *** fuzzy input - integrated bus voltage error - negative ***
// intbusverr_N = 1 - (bverr_int + INTBVERR_FULL)/(INTBVERR_FULL + INTBVERR_OFF);

// intbusverr_N = 1 - (bverr_int + 0.8)/(1.2);

intbusverr_N = 0.3333 - bverr_int*0.8333;   // calculate

if (intbusverr_N < 0) intbusverr_N = 0;     // clip

else if (intbusverr_N > 1) intbusverr_N = 1;


// ======================================================================



// ======================================================================

// FUZZY RULES


// *** battery stabilization rules *****************************

batti_VN += (1-battv_H)*(ucapv_H)*(busverr_N)*(intbusverr_N);

batti_N += (1-battv_H)*(1-ucapv_L)*(busverr_N);

batti_P += (1-battv_L)*(1-ucapv_H)*(busverr_P);

batti_VP += (1-battv_L)*(ucapv_L)*(busverr_P)*(intbusverr_P);

batti_AZ += (1-ucapv_L)*(busverr_P);

batti_AZ += (1-ucapv_H)*(busverr_N);


// *** ultracapacitor stabilization rules **********************

ucapi_VN += (1-ucapv_H)*(busverr_N)*(intbusverr_N);

ucapi_N += (1-ucapv_H)*(busverr_N);

ucapi_P += (1-ucapv_L)*(busverr_P);

ucapi_VP += (1-ucapv_L)*(busverr_P)*(intbusverr_P);


// *** overvoltage discharge rules *****************************

ovdi_L += (battv_H)*(ucapv_H)*(busverr_N);

ovdi_H += (ovdi_L)*(intbusverr_N);


// *** energy transfer rules ***********************************

batti_P += (battv_H)*(ucapv_L);

ucapi_N += (battv_H)*(ucapv_L);
```

```
batti_N += (battv_L)*(ucapv_H);

ucapi_P += (battv_L)*(ucapv_H);


// =======================================================================




// =======================================================================

// DEFUZZIFICATION


// *** battery current membership functions ********************

// max membership current is 9A, for approximate max output current of 5A


if (batti_VP > THRESH)  // very positive

{

    temp = yBar(batti_VP);

    batti_wt += temp;

    // batti_pos += temp * ( 1.8 + (BATTI_VX_B / 2) * (1 + temp) );

    // batti_pos += temp * ( 1.8 + 3.6 * (1 + temp) );

    batti_pos += temp * (5.4 + 3.6 * temp);

}


if (batti_P > THRESH)   // positive

{

    temp = yBar(batti_P);

    batti_wt += temp;

    batti_pos += temp * 2.7;    // symmetrical triangle

}


// about zero

if (batti_AZ > THRESH) batti_wt += yBar(batti_AZ);     // symmetrical about 0


if (batti_N > THRESH)   // negative

{
    temp = yBar(batti_N);

    batti_wt += temp;

    batti_pos += temp * -2.7;    // symmetrical triangle

}


if (batti_VN > THRESH)  // very negative

{

    temp = yBar(batti_VN);

    batti_wt += temp;

    // batti_pos = temp * ( -9 + (BATTI_VX_B / 2) * (1 - temp) );

    // batti_pos = temp * ( -9 + 3.6 * (1 - temp) );

    batti_pos += temp * (-5.4 - (3.6 * temp));

}


// compute output reference and saturate

batti_ref = batti_pos / batti_wt;

if (batti_ref > 5) batti_ref = 5;

if (batti_ref < -5) batti_ref = -5;


// *** ultracapacitor current membership functions *************

// max membership current is 20A, for approximate max output current of 11A


if (ucapi_VP > THRESH)  // very positive

{

    temp = yBar(ucapi_VP);

    ucapi_wt += temp;

    // ucapi_pos += temp * ( 8 + (UCAPI_VX_B / 2) * (1 + temp) );

    // ucapi_pos += temp * ( 8 + 6*(1+temp) );

    ucapi_pos += temp * (14 + 6*temp);

}


if (ucapi_P > THRESH)   // positive

{
    temp = yBar(ucapi_P);

    ucapi_wt += temp;

    // ucapi_pos = temp * ( (UCAPI_X_BASE / 2) + (temp * -1 * UCAPI_DIAG_F / (2 * UCAPI_X_BASE) ) );

    // ucapi_pos = temp * (8 + (temp * -1 * 64 / (2 * 16) ) );

    ucapi_pos += temp * (8 - 2 * temp);

}


if (ucapi_N > THRESH)   // negative

{

    temp = yBar(ucapi_N);

    ucapi_wt += temp;

    // ucapi_pos = temp * ( -16 + (UCAPI_X_BASE / 2) + ( temp * UCAPI_DIAG_F / (2 * UCAPI_X_BASE) ) );

    // ucapi_pos = temp * (-16 + 8 + temp * (64 / (2 * 16) ) );

    ucapi_pos += temp * (temp * 2 - 8);

}


if (ucapi_VN > THRESH)  // very negative

{

    temp = yBar(ucapi_VN);

    ucapi_wt += temp;

    // ucapi_pos += temp * ( -20 + (UCAPI_VX_B / 2) * (1 - temp) );

    // ucapi_pos += temp * ( -20 + 6*(1-temp) );

    ucapi_pos += temp * (-14 - (6 * temp));

}


// compute current reference and saturate

ucapi_ref = ucapi_pos / ucapi_wt;

if (ucapi_ref > 11) ucapi_ref = 11;

if (ucapi_ref < -11) ucapi_ref = -11;


// *** overvoltage discharge current membership functions ******

// max membership current is 12A, for approximate max output current of 6A
```

```
if (ovdi_L > THRESH)    // low
{
    temp = yBar(ovdi_L);
    ovdi_wt += temp;
    // ovdi_pos += temp * ( (OVDI_L_B / 2) * (1 - temp) );
    // ovdi_pos += temp * ( (6 / 2) * (1 - temp) );
    ovdi_pos += temp * (3 - 3*temp);
}

if (ovdi_H > THRESH)    // high
{
    temp = yBar(ovdi_H);
    ovdi_wt += temp;
    // ovdi_pos += temp * ( (OVDI_H_B / 2) * (1 + temp) );
    // ovdi_pos += temp * ( (12 / 2) * (1 + temp) );
    ovdi_pos += temp * (6 + 6*temp);
}

// compute current reference and saturate
ovdi_ref = -1 * ovdi_pos / ovdi_wt;
if (ovdi_ref > 0) ovdi_ref = 0;
if (ovdi_ref < -6) ovdi_ref = -6;

// ========================================================================



/* RelationalOperator: '<Root>/Relational Operator' incorporates:
 *  Constant: '<Root>/batt_low'
 *  Constant: '<Root>/volts_per_bit_63V_max'
 *  DataTypeConversion: '<Root>/Data Type Conversion1'
 *  Product: '<Root>/Product1'
 */
// MODIFIED
```

```
/* S-Function (c280xgpio_do): '<Root>/IO49' */

{

    GpioDataRegs.GPBSET.bit.GPIO49 = (battV <= stab_fuzzy_P.batt_low_Value);

    GpioDataRegs.GPBCLEAR.bit.GPIO49 = !(battV <= stab_fuzzy_P.batt_low_Value);

}


/* RelationalOperator: '<Root>/Relational Operator1' incorporates:

 *  Constant: '<Root>/ucap_low'

 *  Constant: '<Root>/volts_per_bit_63V_max'

 *  DataTypeConversion: '<Root>/Data Type Conversion2'

 *  Product: '<Root>/Product2'

 */

// MODIFIED

/* S-Function (c280xgpio_do): '<Root>/IO61' */

{

    GpioDataRegs.GPBSET.bit.GPIO61 = (ucapV <= stab_fuzzy_P.ucap_low_Value);

    GpioDataRegs.GPBCLEAR.bit.GPIO61 = !(ucapV <= stab_fuzzy_P.ucap_low_Value);

}



// ======================================================================

// REMOVED S1/S7 PI CONTROL SYSTEM (BusV_Batt_Ctrl)

// ======================================================================


// <><><><><><> TOP OF PhaseI_Batt_Ctrl PI CONTROL <><><><><><>


/* Product: '<Root>/Product3' incorporates:

 *  Constant: '<Root>/1_66_V'

 *  Constant: '<Root>/amps_per_bit_14Amax'

 *  Sum: '<Root>/Subtract4'

 */

rtb_Product7 = ((real32_T)batt_I_filt - stab_fuzzy_P._66_V_Value) *

    stab_fuzzy_P.amps_per_bit_14Amax_Value;
```

```
/* Sum: '<S4>/Sum2' */

// MODIFIED - INSERTED FUZZY OUTPUT FOR BATTERY CURRENT

stab_fuzzy_B.Sum1_a = batti_ref - rtb_Product7;


/* Sum: '<S10>/Sum1' incorporates:

 *  Constant: '<S10>/Constant'

 *  Constant: '<S10>/Constant1'

 *  Product: '<S10>/Product'

 *  Product: '<S10>/Product1'

 *  UnitDelay: '<S4>/Unit Delay3'

 *  UnitDelay: '<S4>/Unit Delay4'

 */

rtb_Sum1_l3 = (stab_fuzzy_P.Constant_Value_d * stab_fuzzy_B.Sum1_a +

               stab_fuzzy_DW.UnitDelay3_DSTATE_n) +

    stab_fuzzy_P.Constant1_Value_j * stab_fuzzy_DW.UnitDelay4_DSTATE_e;


/* Sum: '<S4>/Sum1' incorporates:

 *  Constant: '<S4>/prop_gain'

 *  Product: '<S4>/Product'

 */

stab_fuzzy_B.Sum1_a = stab_fuzzy_B.Sum1_a * stab_fuzzy_P.prop_gain_Value_k +

    rtb_Sum1_l3;


/* Saturate: '<S4>/Saturation' */

if (stab_fuzzy_B.Sum1_a > stab_fuzzy_P.Saturation_UpperSat_d) {

    rtb_Saturation_b = stab_fuzzy_P.Saturation_UpperSat_d;

} else if (stab_fuzzy_B.Sum1_a < stab_fuzzy_P.Saturation_LowerSat_a) {

    rtb_Saturation_b = stab_fuzzy_P.Saturation_LowerSat_a;

} else {

    rtb_Saturation_b = stab_fuzzy_B.Sum1_a;

}
```

```
/* End of Saturate: '<S4>/Saturation' */

/* S-Function (c280xpwm): '<Root>/ePWM1' */

/*-- Update CMPA value for ePWM1 --*/
{
    EPwm1Regs.CMPA.half.CMPA = (uint16_T)(rtb_Saturation_b);
}


// ======================================================================
// REMOVED S3/S9 PI CONTROL SYSTEM (BusV_UCap_Ctrl)
// ======================================================================

// <><><><><><> TOP OF PhaseI_UCap_Ctrl PI CONTROL <><><><><><>

/* Product: '<Root>/Product4' incorporates:
 *  Constant: '<Root>/1_66_V'
 *  Constant: '<Root>/amps_per_bit_14Amax'
 *  Sum: '<Root>/Subtract2'
 */
rtb_Product9 = ((real32_T)ucap_I_filt - stab_fuzzy_P._66_V_Value) *
    stab_fuzzy_P.amps_per_bit_14Amax_Value;

/* Sum: '<S6>/Sum2' */
// MODIFIED - INSERTED FUZZY OUTPUT FOR ULTRACAP CURRENT
rtb_Sum1_ih = ucapi_ref - rtb_Product9;

/* Sum: '<S12>/Sum1' incorporates:
 *  Constant: '<S12>/Constant'
 *  Constant: '<S12>/Constant1'
 *  Product: '<S12>/Product'
 *  Product: '<S12>/Product1'
 *  UnitDelay: '<S6>/Unit Delay3'
 *  UnitDelay: '<S6>/Unit Delay4'
 */
rtb_Sum1_c = (stab_fuzzy_P.Constant_Value_m * rtb_Sum1_ih +
              stab_fuzzy_DW.UnitDelay3_DSTATE_p) +
    stab_fuzzy_P.Constant1_Value_n * stab_fuzzy_DW.UnitDelay4_DSTATE_d;

/* Sum: '<S6>/Sum1' incorporates:
 *  Constant: '<S6>/prop_gain'
 *  Product: '<S6>/Product'
 */
rtb_Sum1_ih = rtb_Sum1_ih * stab_fuzzy_P.prop_gain_Value_d + rtb_Sum1_c;

/* Saturate: '<S6>/Saturation' */
if (rtb_Sum1_ih > stab_fuzzy_P.Saturation_UpperSat_dv) {
    rtb_Saturation_a = stab_fuzzy_P.Saturation_UpperSat_dv;
} else if (rtb_Sum1_ih < stab_fuzzy_P.Saturation_LowerSat_e) {
    rtb_Saturation_a = stab_fuzzy_P.Saturation_LowerSat_e;
} else {
    rtb_Saturation_a = rtb_Sum1_ih;
}

/* End of Saturate: '<S6>/Saturation' */

/* S-Function (c280xpwm): '<Root>/ePWM2' */

/*-- Update CMPA value for ePWM2 --*/
{
    EPwm2Regs.CMPA.half.CMPA = (uint16_T)(rtb_Saturation_a);
}


// ======================================================================
// REMOVED S2/S8 PI CONTROL SYSTEM (BusV_OVD_Ctrl)

// ========================================================================


// <><><><><><><> TOP OF PhaseI_OVD_Ctrl PI CONTROL <><><><><><><>


/* Product: '<Root>/Product5' incorporates:

 *  Constant: '<Root>/1_66_V'

 *  Constant: '<Root>/amps_per_bit_14Amax'

 *  DataTypeConversion: '<Root>/Data Type Conversion5'

 *  Sum: '<Root>/Subtract'

 */

rtb_Sum1_n = ((real32_T)ovd_I_filt - stab_fuzzy_P._66_V_Value) *

    stab_fuzzy_P.amps_per_bit_14Amax_Value;


/* Saturate: '<Root>/Saturation' */

if (rtb_Sum1_n > stab_fuzzy_P.Saturation_UpperSat_l2) {

    rtb_Sum1_n = stab_fuzzy_P.Saturation_UpperSat_l2;

} else {

    if (rtb_Sum1_n < stab_fuzzy_P.Saturation_LowerSat_m) {

        rtb_Sum1_n = stab_fuzzy_P.Saturation_LowerSat_m;

    }

}


/* Sum: '<S5>/Sum2' incorporates:

 *  Saturate: '<Root>/Saturation'

 */

// MODIFIED - INSERTED FUZZY OUTPUT FOR OVD CURRENT

rtb_Sum1_n = ovdi_ref - rtb_Sum1_n;


/* Sum: '<S11>/Sum1' incorporates:

 *  Constant: '<S11>/Constant'

 *  Constant: '<S11>/Constant1'

 *  Product: '<S11>/Product'
 *  Product: '<S11>/Product1'

 *  UnitDelay: '<S5>/Unit Delay3'

 *  UnitDelay: '<S5>/Unit Delay4'

 */

rtb_Sum1_i = (stab_fuzzy_P.Constant_Value_h * rtb_Sum1_n +

              stab_fuzzy_DW.UnitDelay3_DSTATE_g) +

    stab_fuzzy_P.Constant1_Value_ne * stab_fuzzy_DW.UnitDelay4_DSTATE_m;


/* Sum: '<S5>/Sum1' incorporates:

 *  Constant: '<S5>/prop_gain'

 *  Product: '<S5>/Product'

 */

rtb_Sum1_n = rtb_Sum1_n * stab_fuzzy_P.prop_gain_Value_kj + rtb_Sum1_i;


/* Saturate: '<S5>/Saturation' */

if (rtb_Sum1_n > stab_fuzzy_P.Saturation_UpperSat_c) {

    rtb_Saturation_m = stab_fuzzy_P.Saturation_UpperSat_c;

} else if (rtb_Sum1_n < stab_fuzzy_P.Saturation_LowerSat_o) {

    rtb_Saturation_m = stab_fuzzy_P.Saturation_LowerSat_o;

} else {

    rtb_Saturation_m = rtb_Sum1_n;

}


/* End of Saturate: '<S5>/Saturation' */


/* S-Function (c280xpwm): '<Root>/ePWM3' */


/*-- Update CMPA value for ePWM3 --*/

{

    EPwm3Regs.CMPA.half.CMPA = (uint16_T)(rtb_Saturation_m);

}


// <><><><><><><> TOP OF AUX PWM INTERFACES <><><><><><><>
```

```
/* Product: '<Root>/Product6' incorporates:
 *  Constant: '<Root>/offset_batt'
 *  Constant: '<Root>/scale_batt'
 *  Sum: '<Root>/Subtract1'
 */
// MODIFIED - INSERTED FUZZY OUTPUT FOR BATTERY CURRENT
rtb_Product8 = (stab_fuzzy_P.offset_batt_Value + batti_ref) *
    stab_fuzzy_P.scale_batt_Value;

/* Product: '<Root>/Product7' incorporates:
 *  Constant: '<Root>/offset_batt'
 *  Constant: '<Root>/scale_batt'
 *  Sum: '<Root>/Subtract3'
 */
rtb_Product7 = (stab_fuzzy_P.offset_batt_Value + rtb_Product7) *
    stab_fuzzy_P.scale_batt_Value;

/* S-Function (c280xpwm): '<Root>/ePWM5' */

/*-- Update CMPA value for ePWM5 --*/
{
    EPwm5Regs.CMPA.half.CMPA = (uint16_T)(rtb_Product8);
}

/*-- Update CMPB value for ePWM5 --*/
{
    EPwm5Regs.CMPB = (uint16_T)(rtb_Product7);
}

/* Product: '<Root>/Product8' incorporates:
 *  Constant: '<Root>/offset_ucap'
 *  Constant: '<Root>/scale_ucap'
 *  Sum: '<Root>/Subtract5'

 */

rtb_Product8 = (ucapi_ref + stab_fuzzy_P.offset_ucap_Value) *

    stab_fuzzy_P.scale_ucap_Value;


/* Product: '<Root>/Product9' incorporates:

 *  Constant: '<Root>/offset_ucap'

 *  Constant: '<Root>/scale_ucap'

 *  Sum: '<Root>/Subtract6'

 */

rtb_Product9 = (rtb_Product9 + stab_fuzzy_P.offset_ucap_Value) *

    stab_fuzzy_P.scale_ucap_Value;


/* S-Function (c280xpwm): '<Root>/ePWM6' */


/*-- Update CMPA value for ePWM6 --*/

{

    EPwm6Regs.CMPA.half.CMPA = (uint16_T)(rtb_Product8);

}


/*-- Update CMPB value for ePWM6 --*/

{

    EPwm6Regs.CMPB = (uint16_T)(rtb_Product9);

}


// ======================================================================

// CUSTOM HEARTBEAT CODE


if (tog > 1) tog = 1;

tog = 1 - tog;

GpioDataRegs.GPBTOGGLE.bit.GPIO48 = tog;


// ======================================================================
/* S-Function (c280xgpio_do): '<Root>/IO24_VFltDis' */
{
    GpioDataRegs.GPASET.bit.GPIO24 = (stab_fuzzy_P.reset_disable_Value != 0);
    GpioDataRegs.GPACLEAR.bit.GPIO24 = !(stab_fuzzy_P.reset_disable_Value !=
        0);
}

/* S-Function (c280xgpio_do): '<Root>/IO30_DCFltDis' */
{
    GpioDataRegs.GPCSET.bit.GPIO84 = (stab_fuzzy_P.reset_disable_Value != 0);
    GpioDataRegs.GPCCLEAR.bit.GPIO84 = !(stab_fuzzy_P.reset_disable_Value !=
        0);
}

/* S-Function (c280xgpio_do): '<Root>/IO84_WFltDis' */
{
    GpioDataRegs.GPCSET.bit.GPIO84 = (stab_fuzzy_P.reset_disable_Value != 0);
    GpioDataRegs.GPCCLEAR.bit.GPIO84 = !(stab_fuzzy_P.reset_disable_Value !=
        0);
}

/* S-Function (c280xgpio_do): '<Root>/IO87_UFltDis' */
{
    GpioDataRegs.GPCSET.bit.GPIO87 = (stab_fuzzy_P.reset_disable_Value != 0);
    GpioDataRegs.GPCCLEAR.bit.GPIO87 = !(stab_fuzzy_P.reset_disable_Value !=
        0);
}

/* S-Function (c280xgpio_di): '<Root>/IO26_!VFlt' */
{
    stab_fuzzy_B.VFlt = GpioDataRegs.GPADAT.bit.GPIO26;
}
```

```
/* S-Function (c280xgpio_di): '<Root>/IO28_!UFlt' */
{
    stab_fuzzy_B.UFlt = GpioDataRegs.GPADAT.bit.GPIO28;
}

/* S-Function (c280xgpio_di): '<Root>/IO34_!DCFlt' */
{
    stab_fuzzy_B.DCFlt = GpioDataRegs.GPBDAT.bit.GPIO34;
}

/* S-Function (c280xgpio_do): '<Root>/IO59' */
{
    GpioDataRegs.GPBSET.bit.GPIO59 = (stab_fuzzy_P.heartbeat_Value != 0);
    GpioDataRegs.GPBCLEAR.bit.GPIO59 = !(stab_fuzzy_P.heartbeat_Value != 0);
}

/* S-Function (c280xgpio_do): '<Root>/IO86_!WFlt' */
{
    GpioDataRegs.GPCSET.bit.GPIO86 = (stab_fuzzy_P.hack_Value != 0);
    GpioDataRegs.GPCCLEAR.bit.GPIO86 = !(stab_fuzzy_P.hack_Value != 0);
}

/* Update for UnitDelay: '<S4>/Unit Delay3' */
stab_fuzzy_DW.UnitDelay3_DSTATE_n = rtb_Sum1_l3;

/* Update for UnitDelay: '<S4>/Unit Delay4' incorporates:
 *  Sum: '<S4>/Sum3'
 */
stab_fuzzy_DW.UnitDelay4_DSTATE_e = rtb_Saturation_b - stab_fuzzy_B.Sum1_a;

/* Update for UnitDelay: '<S6>/Unit Delay3' */
stab_fuzzy_DW.UnitDelay3_DSTATE_p = rtb_Sum1_c;
```

```
    /* Update for UnitDelay: '<S6>/Unit Delay4' incorporates:
     *  Sum: '<S6>/Sum3'
     */
    stab_fuzzy_DW.UnitDelay4_DSTATE_d = rtb_Saturation_a - rtb_Sum1_ih;

    /* Update for UnitDelay: '<S5>/Unit Delay3' */
    stab_fuzzy_DW.UnitDelay3_DSTATE_g = rtb_Sum1_i;

    /* Update for UnitDelay: '<S5>/Unit Delay4' incorporates:
     *  Sum: '<S5>/Sum3'
     */
    stab_fuzzy_DW.UnitDelay4_DSTATE_m = rtb_Saturation_m - rtb_Sum1_n;
}

/* Model initialize function */
void stab_fuzzy_initialize(void)
{
    /* Registration code */

    /* initialize error status */
    rtmSetErrorStatus(stab_fuzzy_M, (NULL));

    /* block I/O */
    (void) memset(((void *) &stab_fuzzy_B), 0,
                  sizeof(B_stab_fuzzy_T));

    /* states (dwork) */
    (void) memset((void *)&stab_fuzzy_DW, 0,
                  sizeof(DW_stab_fuzzy_T));

    /* Start for S-Function (c280xadc): '<Root>/TheAlmightyADC' */
    InitAdc();
```

```
config_ADC_A (5U, 53512U, 254U, 0U, 0U);


/* Start for S-Function (c280xgpio_do): '<Root>/IO49' */

EALLOW;

GpioCtrlRegs.GPBMUX2.all &= 4294967283U;

GpioCtrlRegs.GPBDIR.all |= 131072U;

EDIS;


/* Start for S-Function (c280xgpio_do): '<Root>/IO61' */

EALLOW;

GpioCtrlRegs.GPBMUX2.all &= 4093640703U;

GpioCtrlRegs.GPBDIR.all |= 536870912U;

EDIS;


/* Start for S-Function (c280xpwm): '<Root>/ePWM1' */


/*** Initialize ePWM1 modules ***/

{

    /*-- Setup Time-Base (TB) Submodule --*/

    EPwm1Regs.TBPRD = 11000;


    /* // Time-Base Control Register

       EPwm1Regs.TBCTL.bit.CTRMODE    = 0;          // Counter Mode

       EPwm1Regs.TBCTL.bit.SYNCOSEL   = 3;          // Sync output select

       EPwm1Regs.TBCTL.bit.PRDLD      = 0;          // Shadow select

       EPwm1Regs.TBCTL.bit.PHSEN      = 0;          // Phase load enable

       EPwm1Regs.TBCTL.bit.PHSDIR     = 0;          // Phase Direction

       EPwm1Regs.TBCTL.bit.HSPCLKDIV  = 0;          // High speed time pre-scale

       EPwm1Regs.TBCTL.bit.CLKDIV     = 0;          // Timebase clock pre-scale

     */

    EPwm1Regs.TBCTL.all = (EPwm1Regs.TBCTL.all & ~0x3FBF) | 0x30;


    /* // Time-Base Phase Register
```

```
   EPwm1Regs.TBPHS.half.TBPHS     = 0;          // Phase offset register
 */
EPwm1Regs.TBPHS.all = (EPwm1Regs.TBPHS.all & ~0xFFFF0000) | 0x0;
EPwm1Regs.TBCTR = 0x0000;      /* Clear counter*/

/*-- Setup Counter_Compare (CC) Submodule --*/
/* // Counter-Compare Control Register
   EPwm1Regs.CMPCTL.bit.SHDWAMODE = 0;  // Compare A block operating mode.
   EPwm1Regs.CMPCTL.bit.SHDWBMODE = 0;  // Compare B block operating mode.
   EPwm1Regs.CMPCTL.bit.LOADAMODE = 0;          // Active compare A
   EPwm1Regs.CMPCTL.bit.LOADBMODE = 0;          // Active compare A
 */
EPwm1Regs.CMPCTL.all = (EPwm1Regs.CMPCTL.all & ~0x5F) | 0x0;
EPwm1Regs.CMPA.half.CMPA = 5500;
EPwm1Regs.CMPB = 10000;

/*-- Setup Action-Qualifier (AQ) Submodule --*/
EPwm1Regs.AQCTLA.all = 18;
EPwm1Regs.AQCTLB.all = 33;

/* // Action-Qualifier Software Force Register
   EPwm1Regs.AQSFRC.bit.RLDCSF    = 0;          // Reload from Shadow options
 */
EPwm1Regs.AQSFRC.all = (EPwm1Regs.AQSFRC.all & ~0xC0) | 0x0;

/* // Action-Qualifier Continuous S/W Force Register Set
   EPwm1Regs.AQCSFRC.bit.CSFA     = 0;          // Continuous Software Force on output A
   EPwm1Regs.AQCSFRC.bit.CSFB     = 0;          // Continuous Software Force on output B
 */
EPwm1Regs.AQCSFRC.all = (EPwm1Regs.AQCSFRC.all & ~0xF) | 0x0;

/*-- Setup Dead-Band Generator (DB) Submodule --*/
/* // Dead-Band Generator Control Register
```

```
   EPwm1Regs.DBCTL.bit.OUT_MODE   = 3;          // Dead Band Output Mode Control
   EPwm1Regs.DBCTL.bit.IN_MODE    = 0;          // Dead Band Input Select Mode Control
   EPwm1Regs.DBCTL.bit.POLSEL     = 2;          // Polarity Select Control
 */
EPwm1Regs.DBCTL.all = (EPwm1Regs.DBCTL.all & ~0x3F) | 0xB;
EPwm1Regs.DBRED = 300;
EPwm1Regs.DBFED = 300;

/*-- Setup Event-Trigger (ET) Submodule --*/
/* // Event-Trigger Selection and Event-Trigger Pre-Scale Register
   EPwm1Regs.ETSEL.bit.SOCAEN     = 0;          // Start of conversion A Enable
   EPwm1Regs.ETSEL.bit.SOCASEL    = 1;          // Start of conversion A Select
   EPwm1Regs.ETPS.bit.SOCAPRD     = 1;          // EPWM1SOCA Period Select
   EPwm1Regs.ETSEL.bit.SOCBEN     = 0;          // Start of conversion B Enable
   EPwm1Regs.ETSEL.bit.SOCBSEL    = 1;          // Start of conversion B Select
   EPwm1Regs.ETPS.bit.SOCBPRD     = 1;          // EPWM1SOCB Period Select
   EPwm1Regs.ETSEL.bit.INTEN      = 0;          // EPWM1INTn Enable
   EPwm1Regs.ETSEL.bit.INTSEL     = 1;          // EPWM1INTn Select
   EPwm1Regs.ETPS.bit.INTPRD      = 1;          // EPWM1INTn Period Select
 */
EPwm1Regs.ETSEL.all = (EPwm1Regs.ETSEL.all & ~0xFF0F) | 0x1101;
EPwm1Regs.ETPS.all = (EPwm1Regs.ETPS.all & ~0x3303) | 0x1101;

/*-- Setup PWM-Chopper (PC) Submodule --*/
/* // PWM-Chopper Control Register
   EPwm1Regs.PCCTL.bit.CHPEN      = 0;          // PWM chopping enable
   EPwm1Regs.PCCTL.bit.CHPFREQ    = 0;          // Chopping clock frequency
   EPwm1Regs.PCCTL.bit.OSHTWTH    = 0;          // One-shot pulse width
   EPwm1Regs.PCCTL.bit.CHPDUTY    = 0;          // Chopping clock Duty cycle
 */
EPwm1Regs.PCCTL.all = (EPwm1Regs.PCCTL.all & ~0x7FF) | 0x0;

/*-- Set up Trip-Zone (TZ) Submodule --*/
```

```
    EALLOW;
    EPwm1Regs.TZSEL.all = 0;

    /* // Trip-Zone Control Register
       EPwm1Regs.TZCTL.bit.TZA        = 2;          // TZ1 to TZ6 Trip Action On EPWM1A
       EPwm1Regs.TZCTL.bit.TZB        = 2;          // TZ1 to TZ6 Trip Action On EPWM1B
     */
    EPwm1Regs.TZCTL.all = (EPwm1Regs.TZCTL.all & ~0xF) | 0xA;

    /* // Trip-Zone Enable Interrupt Register
       EPwm1Regs.TZEINT.bit.OST       = 0;          // Trip Zones One Shot Int Enable
       EPwm1Regs.TZEINT.bit.CBC       = 0;          // Trip Zones Cycle By Cycle Int Enable
     */
    EPwm1Regs.TZEINT.all = (EPwm1Regs.TZEINT.all & ~0x6) | 0x0;
    EDIS;
}

/* Start for S-Function (c280xpwm): '<Root>/ePWM2' */

/*** Initialize ePWM2 modules ***/
{
    /*-- Setup Time-Base (TB) Submodule --*/
    EPwm2Regs.TBPRD = 12000;

    /* // Time-Base Control Register
       EPwm2Regs.TBCTL.bit.CTRMODE    = 1;          // Counter Mode
       EPwm2Regs.TBCTL.bit.SYNCOSEL   = 3;          // Sync output select
       EPwm2Regs.TBCTL.bit.PRDLD      = 0;          // Shadow select
       EPwm2Regs.TBCTL.bit.PHSEN      = 0;          // Phase load enable
       EPwm2Regs.TBCTL.bit.PHSDIR     = 0;          // Phase Direction
       EPwm2Regs.TBCTL.bit.HSPCLKDIV  = 0;          // High speed time pre-scale
       EPwm2Regs.TBCTL.bit.CLKDIV     = 0;          // Timebase clock pre-scale
     */
```

```
EPwm2Regs.TBCTL.all = (EPwm2Regs.TBCTL.all & ~0x3FBF) | 0x31;


/* // Time-Base Phase Register

   EPwm2Regs.TBPHS.half.TBPHS     = 0;          // Phase offset register

 */

EPwm2Regs.TBPHS.all = (EPwm2Regs.TBPHS.all & ~0xFFFF0000) | 0x0;

EPwm2Regs.TBCTR = 0x0000;      /* Clear counter*/


/*-- Setup Counter_Compare (CC) Submodule --*/

/* // Counter-Compare Control Register

   EPwm2Regs.CMPCTL.bit.SHDWAMODE = 0;  // Compare A block operating mode.

   EPwm2Regs.CMPCTL.bit.SHDWBMODE = 0;  // Compare B block operating mode.

   EPwm2Regs.CMPCTL.bit.LOADAMODE = 0;          // Active compare A

   EPwm2Regs.CMPCTL.bit.LOADBMODE = 0;          // Active compare A

 */

EPwm2Regs.CMPCTL.all = (EPwm2Regs.CMPCTL.all & ~0x5F) | 0x0;

EPwm2Regs.CMPA.half.CMPA = 6000;

EPwm2Regs.CMPB = 11000;


/*-- Setup Action-Qualifier (AQ) Submodule --*/

EPwm2Regs.AQCTLA.all = 132;

EPwm2Regs.AQCTLB.all = 72;


/* // Action-Qualifier Software Force Register

   EPwm2Regs.AQSFRC.bit.RLDCSF    = 0;          // Reload from Shadow options

 */

EPwm2Regs.AQSFRC.all = (EPwm2Regs.AQSFRC.all & ~0xC0) | 0x0;


/* // Action-Qualifier Continuous S/W Force Register Set

   EPwm2Regs.AQCSFRC.bit.CSFA     = 0;          // Continuous Software Force on output A

   EPwm2Regs.AQCSFRC.bit.CSFB     = 0;          // Continuous Software Force on output B

 */

EPwm2Regs.AQCSFRC.all = (EPwm2Regs.AQCSFRC.all & ~0xF) | 0x0;
```

```
/*-- Setup Dead-Band Generator (DB) Submodule --*/
/* // Dead-Band Generator Control Register
   EPwm2Regs.DBCTL.bit.OUT_MODE   = 3;          // Dead Band Output Mode Control
   EPwm2Regs.DBCTL.bit.IN_MODE    = 0;          // Dead Band Input Select Mode Control
   EPwm2Regs.DBCTL.bit.POLSEL     = 2;          // Polarity Select Control
 */
EPwm2Regs.DBCTL.all = (EPwm2Regs.DBCTL.all & ~0x3F) | 0xB;
EPwm2Regs.DBRED = 300;
EPwm2Regs.DBFED = 300;

/*-- Setup Event-Trigger (ET) Submodule --*/
/* // Event-Trigger Selection and Event-Trigger Pre-Scale Register
   EPwm2Regs.ETSEL.bit.SOCAEN     = 0;          // Start of conversion A Enable
   EPwm2Regs.ETSEL.bit.SOCASEL    = 1;          // Start of conversion A Select
   EPwm2Regs.ETPS.bit.SOCAPRD     = 1;          // EPWM2SOCA Period Select
   EPwm2Regs.ETSEL.bit.SOCBEN     = 0;          // Start of conversion B Enable
   EPwm2Regs.ETSEL.bit.SOCBSEL    = 1;          // Start of conversion B Select
   EPwm2Regs.ETPS.bit.SOCBPRD     = 1;          // EPWM2SOCB Period Select
   EPwm2Regs.ETSEL.bit.INTEN      = 0;          // EPWM2INTn Enable
   EPwm2Regs.ETSEL.bit.INTSEL     = 1;          // EPWM2INTn Select
   EPwm2Regs.ETPS.bit.INTPRD      = 1;          // EPWM2INTn Period Select
 */
EPwm2Regs.ETSEL.all = (EPwm2Regs.ETSEL.all & ~0xFF0F) | 0x1101;
EPwm2Regs.ETPS.all = (EPwm2Regs.ETPS.all & ~0x3303) | 0x1101;

/*-- Setup PWM-Chopper (PC) Submodule --*/
/* // PWM-Chopper Control Register
   EPwm2Regs.PCCTL.bit.CHPEN      = 0;          // PWM chopping enable
   EPwm2Regs.PCCTL.bit.CHPFREQ    = 0;          // Chopping clock frequency
   EPwm2Regs.PCCTL.bit.OSHTWTH    = 0;          // One-shot pulse width
   EPwm2Regs.PCCTL.bit.CHPDUTY    = 0;          // Chopping clock Duty cycle
 */
```

```
    EPwm2Regs.PCCTL.all = (EPwm2Regs.PCCTL.all & ~0x7FF) | 0x0;

    /*-- Set up Trip-Zone (TZ) Submodule --*/
    EALLOW;
    EPwm2Regs.TZSEL.all = 0;

    /* // Trip-Zone Control Register
       EPwm2Regs.TZCTL.bit.TZA        = 2;          // TZ1 to TZ6 Trip Action On EPWM2A
       EPwm2Regs.TZCTL.bit.TZB        = 2;          // TZ1 to TZ6 Trip Action On EPWM2B
     */
    EPwm2Regs.TZCTL.all = (EPwm2Regs.TZCTL.all & ~0xF) | 0xA;

    /* // Trip-Zone Enable Interrupt Register
       EPwm2Regs.TZEINT.bit.OST       = 0;          // Trip Zones One Shot Int Enable
       EPwm2Regs.TZEINT.bit.CBC       = 0;          // Trip Zones Cycle By Cycle Int Enable
     */
    EPwm2Regs.TZEINT.all = (EPwm2Regs.TZEINT.all & ~0x6) | 0x0;
    EDIS;
}

/* Start for S-Function (c280xpwm): '<Root>/ePWM3' */

/*** Initialize ePWM3 modules ***/
{
    /*-- Setup Time-Base (TB) Submodule --*/
    EPwm3Regs.TBPRD = 15000;

    /* // Time-Base Control Register
       EPwm3Regs.TBCTL.bit.CTRMODE    = 0;          // Counter Mode
       EPwm3Regs.TBCTL.bit.SYNCOSEL   = 3;          // Sync output select
       EPwm3Regs.TBCTL.bit.PRDLD      = 0;          // Shadow select
       EPwm3Regs.TBCTL.bit.PHSEN      = 0;          // Phase load enable
       EPwm3Regs.TBCTL.bit.PHSDIR     = 0;          // Phase Direction
```

```
   EPwm3Regs.TBCTL.bit.HSPCLKDIV  = 0;          // High speed time pre-scale

   EPwm3Regs.TBCTL.bit.CLKDIV     = 0;          // Timebase clock pre-scale

 */

EPwm3Regs.TBCTL.all = (EPwm3Regs.TBCTL.all & ~0x3FBF) | 0x30;


/* // Time-Base Phase Register

   EPwm3Regs.TBPHS.half.TBPHS     = 0;          // Phase offset register

 */

EPwm3Regs.TBPHS.all = (EPwm3Regs.TBPHS.all & ~0xFFFF0000) | 0x0;

EPwm3Regs.TBCTR = 0x0000;      /* Clear counter*/


/*-- Setup Counter_Compare (CC) Submodule --*/

/* // Counter-Compare Control Register

   EPwm3Regs.CMPCTL.bit.SHDWAMODE = 0;  // Compare A block operating mode.

   EPwm3Regs.CMPCTL.bit.SHDWBMODE = 0;  // Compare B block operating mode.

   EPwm3Regs.CMPCTL.bit.LOADAMODE = 0;          // Active compare A

   EPwm3Regs.CMPCTL.bit.LOADBMODE = 0;          // Active compare A

 */

EPwm3Regs.CMPCTL.all = (EPwm3Regs.CMPCTL.all & ~0x5F) | 0x0;

EPwm3Regs.CMPA.half.CMPA = 15000;

EPwm3Regs.CMPB = 0;


/*-- Setup Action-Qualifier (AQ) Submodule --*/

EPwm3Regs.AQCTLA.all = 36;

EPwm3Regs.AQCTLB.all = 33;


/* // Action-Qualifier Software Force Register

   EPwm3Regs.AQSFRC.bit.RLDCSF    = 0;          // Reload from Shadow options

 */

EPwm3Regs.AQSFRC.all = (EPwm3Regs.AQSFRC.all & ~0xC0) | 0x0;


/* // Action-Qualifier Continuous S/W Force Register Set

   EPwm3Regs.AQCSFRC.bit.CSFA     = 0;          // Continuous Software Force on output A
   EPwm3Regs.AQCSFRC.bit.CSFB     = 0;          // Continuous Software Force on output B
 */
EPwm3Regs.AQCSFRC.all = (EPwm3Regs.AQCSFRC.all & ~0xF) | 0x0;

/*-- Setup Dead-Band Generator (DB) Submodule --*/
/* // Dead-Band Generator Control Register
   EPwm3Regs.DBCTL.bit.OUT_MODE   = 0;          // Dead Band Output Mode Control
   EPwm3Regs.DBCTL.bit.IN_MODE    = 0;          // Dead Band Input Select Mode Control
   EPwm3Regs.DBCTL.bit.POLSEL     = 0;          // Polarity Select Control
 */
EPwm3Regs.DBCTL.all = (EPwm3Regs.DBCTL.all & ~0x3F) | 0x0;
EPwm3Regs.DBRED = 0;
EPwm3Regs.DBFED = 0;

/*-- Setup Event-Trigger (ET) Submodule --*/
/* // Event-Trigger Selection and Event-Trigger Pre-Scale Register
   EPwm3Regs.ETSEL.bit.SOCAEN     = 0;          // Start of conversion A Enable
   EPwm3Regs.ETSEL.bit.SOCASEL    = 1;          // Start of conversion A Select
   EPwm3Regs.ETPS.bit.SOCAPRD     = 1;          // EPWM3SOCA Period Select
   EPwm3Regs.ETSEL.bit.SOCBEN     = 0;          // Start of conversion B Enable
   EPwm3Regs.ETSEL.bit.SOCBSEL    = 1;          // Start of conversion B Select
   EPwm3Regs.ETPS.bit.SOCBPRD     = 1;          // EPWM3SOCB Period Select
   EPwm3Regs.ETSEL.bit.INTEN      = 0;          // EPWM3INTn Enable
   EPwm3Regs.ETSEL.bit.INTSEL     = 1;          // EPWM3INTn Select
   EPwm3Regs.ETPS.bit.INTPRD      = 1;          // EPWM3INTn Period Select
 */
EPwm3Regs.ETSEL.all = (EPwm3Regs.ETSEL.all & ~0xFF0F) | 0x1101;
EPwm3Regs.ETPS.all = (EPwm3Regs.ETPS.all & ~0x3303) | 0x1101;

/*-- Setup PWM-Chopper (PC) Submodule --*/
/* // PWM-Chopper Control Register
   EPwm3Regs.PCCTL.bit.CHPEN      = 0;          // PWM chopping enable
   EPwm3Regs.PCCTL.bit.CHPFREQ    = 0;          // Chopping clock frequency
```

```
      EPwm3Regs.PCCTL.bit.OSHTWTH    = 0;          // One-shot pulse width
      EPwm3Regs.PCCTL.bit.CHPDUTY    = 0;          // Chopping clock Duty cycle
    */
   EPwm3Regs.PCCTL.all = (EPwm3Regs.PCCTL.all & ~0x7FF) | 0x0;

   /*-- Set up Trip-Zone (TZ) Submodule --*/
   EALLOW;
   EPwm3Regs.TZSEL.all = 0;

   /* // Trip-Zone Control Register
      EPwm3Regs.TZCTL.bit.TZA        = 2;          // TZ1 to TZ6 Trip Action On EPWM3A
      EPwm3Regs.TZCTL.bit.TZB        = 2;          // TZ1 to TZ6 Trip Action On EPWM3B
    */
   EPwm3Regs.TZCTL.all = (EPwm3Regs.TZCTL.all & ~0xF) | 0xA;

   /* // Trip-Zone Enable Interrupt Register
      EPwm3Regs.TZEINT.bit.OST       = 0;          // Trip Zones One Shot Int Enable
      EPwm3Regs.TZEINT.bit.CBC       = 0;          // Trip Zones Cycle By Cycle Int Enable
    */
   EPwm3Regs.TZEINT.all = (EPwm3Regs.TZEINT.all & ~0x6) | 0x0;
   EDIS;
}

/* Start for S-Function (c280xpwm): '<Root>/ePWM5' */

/*** Initialize ePWM5 modules ***/
{
   /*-- Setup Time-Base (TB) Submodule --*/
   EPwm5Regs.TBPRD = 1000;

   /* // Time-Base Control Register
      EPwm5Regs.TBCTL.bit.CTRMODE    = 0;          // Counter Mode
      EPwm5Regs.TBCTL.bit.SYNCOSEL   = 3;          // Sync output select
      EPwm5Regs.TBCTL.bit.PRDLD      = 0;          // Shadow select

      EPwm5Regs.TBCTL.bit.PHSEN      = 0;          // Phase load enable

      EPwm5Regs.TBCTL.bit.PHSDIR     = 0;          // Phase Direction

      EPwm5Regs.TBCTL.bit.HSPCLKDIV  = 0;          // High speed time pre-scale

      EPwm5Regs.TBCTL.bit.CLKDIV     = 0;          // Timebase clock pre-scale

    */

   EPwm5Regs.TBCTL.all = (EPwm5Regs.TBCTL.all & ~0x3FBF) | 0x30;


   /* // Time-Base Phase Register

      EPwm5Regs.TBPHS.half.TBPHS     = 0;          // Phase offset register

    */

   EPwm5Regs.TBPHS.all = (EPwm5Regs.TBPHS.all & ~0xFFFF0000) | 0x0;

   EPwm5Regs.TBCTR = 0x0000;      /* Clear counter*/


   /*-- Setup Counter_Compare (CC) Submodule --*/

   /* // Counter-Compare Control Register

      EPwm5Regs.CMPCTL.bit.SHDWAMODE = 0;  // Compare A block operating mode.

      EPwm5Regs.CMPCTL.bit.SHDWBMODE = 0;  // Compare B block operating mode.

      EPwm5Regs.CMPCTL.bit.LOADAMODE = 0;          // Active compare A

      EPwm5Regs.CMPCTL.bit.LOADBMODE = 0;          // Active compare A

    */

   EPwm5Regs.CMPCTL.all = (EPwm5Regs.CMPCTL.all & ~0x5F) | 0x0;

   EPwm5Regs.CMPA.half.CMPA = 50;

   EPwm5Regs.CMPB = 50;


   /*-- Setup Action-Qualifier (AQ) Submodule --*/

   EPwm5Regs.AQCTLA.all = 18;

   EPwm5Regs.AQCTLB.all = 258;


   /* // Action-Qualifier Software Force Register

      EPwm5Regs.AQSFRC.bit.RLDCSF    = 0;          // Reload from Shadow options

    */

   EPwm5Regs.AQSFRC.all = (EPwm5Regs.AQSFRC.all & ~0xC0) | 0x0;
```

```
/* // Action-Qualifier Continuous S/W Force Register Set
   EPwm5Regs.AQCSFRC.bit.CSFA     = 0;          // Continuous Software Force on output A
   EPwm5Regs.AQCSFRC.bit.CSFB     = 0;          // Continuous Software Force on output B
 */
EPwm5Regs.AQCSFRC.all = (EPwm5Regs.AQCSFRC.all & ~0xF) | 0x0;

/*-- Setup Dead-Band Generator (DB) Submodule --*/
/* // Dead-Band Generator Control Register
   EPwm5Regs.DBCTL.bit.OUT_MODE   = 0;          // Dead Band Output Mode Control
   EPwm5Regs.DBCTL.bit.IN_MODE    = 0;          // Dead Band Input Select Mode Control
   EPwm5Regs.DBCTL.bit.POLSEL     = 0;          // Polarity Select Control
 */
EPwm5Regs.DBCTL.all = (EPwm5Regs.DBCTL.all & ~0x3F) | 0x0;
EPwm5Regs.DBRED = 0;
EPwm5Regs.DBFED = 0;

/*-- Setup Event-Trigger (ET) Submodule --*/
/* // Event-Trigger Selection and Event-Trigger Pre-Scale Register
   EPwm5Regs.ETSEL.bit.SOCAEN     = 0;          // Start of conversion A Enable
   EPwm5Regs.ETSEL.bit.SOCASEL    = 1;          // Start of conversion A Select
   EPwm5Regs.ETPS.bit.SOCAPRD     = 1;          // EPWM5SOCA Period Select
   EPwm5Regs.ETSEL.bit.SOCBEN     = 0;          // Start of conversion B Enable
   EPwm5Regs.ETSEL.bit.SOCBSEL    = 1;          // Start of conversion B Select
   EPwm5Regs.ETPS.bit.SOCBPRD     = 1;          // EPWM5SOCB Period Select
   EPwm5Regs.ETSEL.bit.INTEN      = 0;          // EPWM5INTn Enable
   EPwm5Regs.ETSEL.bit.INTSEL     = 1;          // EPWM5INTn Select
   EPwm5Regs.ETPS.bit.INTPRD      = 1;          // EPWM5INTn Period Select
 */
EPwm5Regs.ETSEL.all = (EPwm5Regs.ETSEL.all & ~0xFF0F) | 0x1101;
EPwm5Regs.ETPS.all = (EPwm5Regs.ETPS.all & ~0x3303) | 0x1101;

/*-- Setup PWM-Chopper (PC) Submodule --*/
```

```
    /* // PWM-Chopper Control Register

       EPwm5Regs.PCCTL.bit.CHPEN      = 0;          // PWM chopping enable

       EPwm5Regs.PCCTL.bit.CHPFREQ    = 0;          // Chopping clock frequency

       EPwm5Regs.PCCTL.bit.OSHTWTH    = 0;          // One-shot pulse width

       EPwm5Regs.PCCTL.bit.CHPDUTY    = 0;          // Chopping clock Duty cycle

     */

    EPwm5Regs.PCCTL.all = (EPwm5Regs.PCCTL.all & ~0x7FF) | 0x0;


    /*-- Set up Trip-Zone (TZ) Submodule --*/

    EALLOW;

    EPwm5Regs.TZSEL.all = 0;


    /* // Trip-Zone Control Register

       EPwm5Regs.TZCTL.bit.TZA        = 2;          // TZ1 to TZ6 Trip Action On EPWM5A

       EPwm5Regs.TZCTL.bit.TZB        = 2;          // TZ1 to TZ6 Trip Action On EPWM5B

     */

    EPwm5Regs.TZCTL.all = (EPwm5Regs.TZCTL.all & ~0xF) | 0xA;


    /* // Trip-Zone Enable Interrupt Register

       EPwm5Regs.TZEINT.bit.OST       = 0;          // Trip Zones One Shot Int Enable

       EPwm5Regs.TZEINT.bit.CBC       = 0;          // Trip Zones Cycle By Cycle Int Enable

     */

    EPwm5Regs.TZEINT.all = (EPwm5Regs.TZEINT.all & ~0x6) | 0x0;

    EDIS;

}


/* Start for S-Function (c280xpwm): '<Root>/ePWM6' */


/*** Initialize ePWM6 modules ***/

{

    /*-- Setup Time-Base (TB) Submodule --*/

    EPwm6Regs.TBPRD = 1000;
```

```
/* // Time-Base Control Register

   EPwm6Regs.TBCTL.bit.CTRMODE    = 0;          // Counter Mode

   EPwm6Regs.TBCTL.bit.SYNCOSEL   = 3;          // Sync output select

   EPwm6Regs.TBCTL.bit.PRDLD      = 0;          // Shadow select

   EPwm6Regs.TBCTL.bit.PHSEN      = 0;          // Phase load enable

   EPwm6Regs.TBCTL.bit.PHSDIR     = 0;          // Phase Direction

   EPwm6Regs.TBCTL.bit.HSPCLKDIV  = 0;          // High speed time pre-scale

   EPwm6Regs.TBCTL.bit.CLKDIV     = 0;          // Timebase clock pre-scale

 */

EPwm6Regs.TBCTL.all = (EPwm6Regs.TBCTL.all & ~0x3FBF) | 0x30;


/* // Time-Base Phase Register

   EPwm6Regs.TBPHS.half.TBPHS     = 0;          // Phase offset register

 */

EPwm6Regs.TBPHS.all = (EPwm6Regs.TBPHS.all & ~0xFFFF0000) | 0x0;

EPwm6Regs.TBCTR = 0x0000;      /* Clear counter*/


/*-- Setup Counter_Compare (CC) Submodule --*/

/* // Counter-Compare Control Register

   EPwm6Regs.CMPCTL.bit.SHDWAMODE = 0;  // Compare A block operating mode.

   EPwm6Regs.CMPCTL.bit.SHDWBMODE = 0;  // Compare B block operating mode.

   EPwm6Regs.CMPCTL.bit.LOADAMODE = 0;          // Active compare A

   EPwm6Regs.CMPCTL.bit.LOADBMODE = 0;          // Active compare A

 */

EPwm6Regs.CMPCTL.all = (EPwm6Regs.CMPCTL.all & ~0x5F) | 0x0;

EPwm6Regs.CMPA.half.CMPA = 50;

EPwm6Regs.CMPB = 50;


/*-- Setup Action-Qualifier (AQ) Submodule --*/

EPwm6Regs.AQCTLA.all = 18;

EPwm6Regs.AQCTLB.all = 258;


/* // Action-Qualifier Software Force Register
```

```
   EPwm6Regs.AQSFRC.bit.RLDCSF    = 0;          // Reload from Shadow options

 */

EPwm6Regs.AQSFRC.all = (EPwm6Regs.AQSFRC.all & ~0xC0) | 0x0;


/* // Action-Qualifier Continuous S/W Force Register Set

   EPwm6Regs.AQCSFRC.bit.CSFA     = 0;          // Continuous Software Force on output A

   EPwm6Regs.AQCSFRC.bit.CSFB     = 0;          // Continuous Software Force on output B

 */

EPwm6Regs.AQCSFRC.all = (EPwm6Regs.AQCSFRC.all & ~0xF) | 0x0;


/*-- Setup Dead-Band Generator (DB) Submodule --*/

/* // Dead-Band Generator Control Register

   EPwm6Regs.DBCTL.bit.OUT_MODE   = 0;          // Dead Band Output Mode Control

   EPwm6Regs.DBCTL.bit.IN_MODE    = 0;          // Dead Band Input Select Mode Control

   EPwm6Regs.DBCTL.bit.POLSEL     = 0;          // Polarity Select Control

 */

EPwm6Regs.DBCTL.all = (EPwm6Regs.DBCTL.all & ~0x3F) | 0x0;

EPwm6Regs.DBRED = 0;

EPwm6Regs.DBFED = 0;


/*-- Setup Event-Trigger (ET) Submodule --*/

/* // Event-Trigger Selection and Event-Trigger Pre-Scale Register

   EPwm6Regs.ETSEL.bit.SOCAEN     = 0;          // Start of conversion A Enable

   EPwm6Regs.ETSEL.bit.SOCASEL    = 1;          // Start of conversion A Select

   EPwm6Regs.ETPS.bit.SOCAPRD     = 1;          // EPWM6SOCA Period Select

   EPwm6Regs.ETSEL.bit.SOCBEN     = 0;          // Start of conversion B Enable

   EPwm6Regs.ETSEL.bit.SOCBSEL    = 1;          // Start of conversion B Select

   EPwm6Regs.ETPS.bit.SOCBPRD     = 1;          // EPWM6SOCB Period Select

   EPwm6Regs.ETSEL.bit.INTEN      = 0;          // EPWM6INTn Enable

   EPwm6Regs.ETSEL.bit.INTSEL     = 1;          // EPWM6INTn Select

   EPwm6Regs.ETPS.bit.INTPRD      = 1;          // EPWM6INTn Period Select

 */

EPwm6Regs.ETSEL.all = (EPwm6Regs.ETSEL.all & ~0xFF0F) | 0x1101;
```

```
    EPwm6Regs.ETPS.all = (EPwm6Regs.ETPS.all & ~0x3303) | 0x1101;


    /*-- Setup PWM-Chopper (PC) Submodule --*/
    /* // PWM-Chopper Control Register
       EPwm6Regs.PCCTL.bit.CHPEN      = 0;          // PWM chopping enable
       EPwm6Regs.PCCTL.bit.CHPFREQ    = 0;          // Chopping clock frequency
       EPwm6Regs.PCCTL.bit.OSHTWTH    = 0;          // One-shot pulse width
       EPwm6Regs.PCCTL.bit.CHPDUTY    = 0;          // Chopping clock Duty cycle
     */
    EPwm6Regs.PCCTL.all = (EPwm6Regs.PCCTL.all & ~0x7FF) | 0x0;


    /*-- Set up Trip-Zone (TZ) Submodule --*/
    EALLOW;
    EPwm6Regs.TZSEL.all = 0;


    /* // Trip-Zone Control Register
       EPwm6Regs.TZCTL.bit.TZA        = 2;          // TZ1 to TZ6 Trip Action On EPWM6A
       EPwm6Regs.TZCTL.bit.TZB        = 2;          // TZ1 to TZ6 Trip Action On EPWM6B
     */
    EPwm6Regs.TZCTL.all = (EPwm6Regs.TZCTL.all & ~0xF) | 0xA;


    /* // Trip-Zone Enable Interrupt Register
       EPwm6Regs.TZEINT.bit.OST       = 0;          // Trip Zones One Shot Int Enable
       EPwm6Regs.TZEINT.bit.CBC       = 0;          // Trip Zones Cycle By Cycle Int Enable
     */
    EPwm6Regs.TZEINT.all = (EPwm6Regs.TZEINT.all & ~0x6) | 0x0;
    EDIS;
}

// ======================================================================
// CUSTOM CONFIGURATION FOR HEARTBEAT SIGNAL


EALLOW;
```

```
GpioCtrlRegs.GPBDIR.bit.GPIO48 = 1;

EDIS;


// ======================================================================


/* Start for S-Function (c280xgpio_do): '<Root>/IO24_VFltDis' */

EALLOW;

GpioCtrlRegs.GPAMUX2.all &= 4294770687U;

GpioCtrlRegs.GPADIR.all |= 16777216U;

EDIS;


/* Start for S-Function (c280xgpio_do): '<Root>/IO30_DCFltDis' */

EALLOW;

GpioCtrlRegs.GPCMUX2.all &= 4294966527U;

GpioCtrlRegs.GPCDIR.all |= 1048576U;

EDIS;


/* Start for S-Function (c280xgpio_do): '<Root>/IO84_WFltDis' */

EALLOW;

GpioCtrlRegs.GPCMUX2.all &= 4294966527U;

GpioCtrlRegs.GPCDIR.all |= 1048576U;

EDIS;


/* Start for S-Function (c280xgpio_do): '<Root>/IO87_UFltDis' */

EALLOW;

GpioCtrlRegs.GPCMUX2.all &= 4294918143U;

GpioCtrlRegs.GPCDIR.all |= 8388608U;

EDIS;


/* Start for S-Function (c280xgpio_di): '<Root>/IO26_!VFlt' */

EALLOW;

GpioCtrlRegs.GPAMUX2.all &= 4291821567U;

GpioCtrlRegs.GPADIR.all &= 4227858431U;
```

```
EDIS;


/* Start for S-Function (c280xgpio_di): '<Root>/IO28_!UFlt' */

EALLOW;

GpioCtrlRegs.GPAMUX2.all &= 4244635647U;

GpioCtrlRegs.GPADIR.all &= 4026531839U;

EDIS;


/* Start for S-Function (c280xgpio_di): '<Root>/IO34_!DCFlt' */

EALLOW;

GpioCtrlRegs.GPBMUX1.all &= 4294967247U;

GpioCtrlRegs.GPBDIR.all &= 4294967291U;

EDIS;


/* Start for S-Function (c280xgpio_do): '<Root>/IO59' */

EALLOW;

GpioCtrlRegs.GPBMUX2.all &= 4282384383U;

GpioCtrlRegs.GPBDIR.all |= 134217728U;

EDIS;


/* Start for S-Function (c280xgpio_do): '<Root>/IO86_!WFlt' */

EALLOW;

GpioCtrlRegs.GPCMUX2.all &= 4294955007U;

GpioCtrlRegs.GPCDIR.all |= 4194304U;

EDIS;


/* InitializeConditions for UnitDelay: '<S1>/Unit Delay3' */

stab_fuzzy_DW.UnitDelay3_DSTATE = stab_fuzzy_P.UnitDelay3_InitialCondition;


/* InitializeConditions for UnitDelay: '<S1>/Unit Delay4' */

stab_fuzzy_DW.UnitDelay4_DSTATE = stab_fuzzy_P.UnitDelay4_InitialCondition;


/* InitializeConditions for UnitDelay: '<S4>/Unit Delay3' */
```

```
stab_fuzzy_DW.UnitDelay3_DSTATE_n =
    stab_fuzzy_P.UnitDelay3_InitialCondition_m;

/* InitializeConditions for UnitDelay: '<S4>/Unit Delay4' */
stab_fuzzy_DW.UnitDelay4_DSTATE_e =
    stab_fuzzy_P.UnitDelay4_InitialCondition_k;

/* InitializeConditions for UnitDelay: '<S3>/Unit Delay3' */
stab_fuzzy_DW.UnitDelay3_DSTATE_l =
    stab_fuzzy_P.UnitDelay3_InitialCondition_g;

/* InitializeConditions for UnitDelay: '<S3>/Unit Delay4' */
stab_fuzzy_DW.UnitDelay4_DSTATE_n =
    stab_fuzzy_P.UnitDelay4_InitialCondition_j;

/* InitializeConditions for UnitDelay: '<S6>/Unit Delay3' */
stab_fuzzy_DW.UnitDelay3_DSTATE_p =
    stab_fuzzy_P.UnitDelay3_InitialCondition_b;

/* InitializeConditions for UnitDelay: '<S6>/Unit Delay4' */
stab_fuzzy_DW.UnitDelay4_DSTATE_d =
    stab_fuzzy_P.UnitDelay4_InitialCondition_o;

/* InitializeConditions for UnitDelay: '<S2>/Unit Delay3' */
stab_fuzzy_DW.UnitDelay3_DSTATE_i =
    stab_fuzzy_P.UnitDelay3_InitialCondition_i;

/* InitializeConditions for UnitDelay: '<S2>/Unit Delay4' */
stab_fuzzy_DW.UnitDelay4_DSTATE_i =
    stab_fuzzy_P.UnitDelay4_InitialCondition_o5;

/* InitializeConditions for UnitDelay: '<S5>/Unit Delay3' */
stab_fuzzy_DW.UnitDelay3_DSTATE_g =
        stab_fuzzy_P.UnitDelay3_InitialCondition_n;


    /* InitializeConditions for UnitDelay: '<S5>/Unit Delay4' */
    stab_fuzzy_DW.UnitDelay4_DSTATE_m =
        stab_fuzzy_P.UnitDelay4_InitialCondition_ja;
}


/* Model terminate function */
void stab_fuzzy_terminate(void)
{
    /* (no terminate code required) */
}


/*
 * File trailer for generated code.
 *
 * [EOF]
 */
```