ABSTRACT

DOSHI, JIMIT. Framework for the Integration of Distributed Energy Storage Device with the Future Renewable Electric Energy Delivery and Management System. (Under the direction of Dr. Srdjan Lukic.)

The Future Renewable Electric Energy Delivery and Management (FREEDM) System is a revolutionary concept in revamping the grid infrastructure and making the move towards Smart Grids. It envisions a distributed electric energy system that supports bi-directional power flow with the grid, coupled with the intelligence to make optimal use of such distributed resources from an economic and electric standpoint.

The FREEDM System consists of several localized Energy Cells made of one or more renewable energy generation and storage devices which are governed by distributed and intelligent energy management algorithms. Energy storage devices, based on batteries and super capacitors, shall play a vital role in such a distributed system to act as local reservoirs of energy that can be utilized for applications such as peak shaving, frequency regulation, load levelling etc. The operation and control of such a Distributed Energy Storage Device (DESD) thus depends on its control and communication interface with other devices within the Energy Cell.

This thesis presents a comprehensive framework to integrate a DESD within the FREEDM Energy Cell. It involves defining a standardized command and status interface, called the 'Device Profile', which it can share with any other device on the network. We implement an end to end communication network between the DSPs (that directly interact with the hardware of the DESD) and the external entity that seeks to control the DESD. We make judicious choice of various communication protocols such as MQTT, MODBUS and IEEE 802.15.4 based on RF that make this network, considering not only system requirements but also their scalability, portability and acceptance in the industry. In this regard, the DESD is used as a template device to not only define the communication framework but also the overall system architecture from
an embedded systems perspective. We arrive at a two-tier solution consisting of the DSPs to directly control the power electronics and a central single board computer such as the Beagle Bone Black (BBB) that acts as a communication gateway and also an application development platform. This scheme is generic and suitable enough for all the power converter devices within the FREEDM Energy Cell, and not just the DESD.

Using the BBB platform and a battery data acquisition system, we implement a State of Charge (SoC) estimation algorithm for the DESD which helps us track its energy reserve capacity in real time. Based on this we develop other application such as SoC Balancing between the batteries of DESD and also Integration with a Smart Home system. We also characterize the performance of the DESD by measuring parameters such as communication latencies, reference tracking accuracy etc.
Framework for the Integration of Distributed Energy Storage Device with the Future Renewable Electric Energy Delivery and Management System

by
Jimit Doshi

A thesis submitted to the Graduate Faculty of North Carolina State University in partial fulfillment of the requirements for the Degree of Master of Science

Electrical Engineering

Raleigh, North Carolina
2015

APPROVED BY:

Dr. David Lubkeman
Dr. Iqbal Husain
Dr. Srdjan Lukic
Chair of Advisory Committee
DEDICATION

To my parents

Parul and Dhiren Doshi
BIOGRAPHY

Jimit Doshi was born on October 2, 1987 in the city of Bhavnagar (Gujarat), India. He received the Bachelor of Engineering (B.E.) degree in Electronics Engineering from the K. J. Somaiya College of Engineering, University of Mumbai, in 2009. Subsequently he worked as a Senior Embedded Design Engineer at Sprylogic Technologies Ltd. (Thane, 2009 - 2012) and as a Senior Software Engineer at Robert Bosch Engineering and Business Solutions India Ltd. (Coimbatore, 2012 - 2013).

Jimit joined the Electrical and Computer Engineering Department of North Carolina State University in August 2013 to pursue the Master of Science (M.S.) degree in Electrical Engineering. He has been working with the Future Renewable Electric Energy Delivery and Management (FREEDM) Systems Center under the guidance of Dr. Srdjan Lukic since January 2014. He worked at Makani Power (Google) as a Hardware Engineering Intern from June to August 2014.

His research interests include Embedded Systems and Power Electronics. After his graduation he plans to pursue a career in the industry in an engineering design capacity.
ACKNOWLEDGEMENTS

I thank Dr. Srdjan Lukic for giving me the opportunity to work on this Thesis. Your complete support and faith in me since I joined the FREEDM Center has given me the confidence to pursue this Thesis to the best of my abilities. I shall always remember the numerous insightful discussions we have had over the past year as they have always been a great source of learning. I sincerely appreciate your patience with me when things wouldn’t necessarily go as planned.

I thank Dr. David Lubkeman for guiding me as a part of the Software Team responsible for the Green Energy Hub (GEH) Demonstration. It wouldn’t have been possible to make all the appropriate design choices for architecting the communication network if not for your helpful suggestions and deep understanding of the system requirements.

I express my gratitude to Dr. Iqbal Husain for agreeing to be a part of my committee and efficiently leading the FREEDM Center over the past year.

Thank you Siddhartha for handling the MQTT backend and a lot many other things. It was great fun working with you. Thanks Habib for often taking out the time in spite of your hectic schedule for guiding me with the implementation of SoC algorithm. I thank Youxi Shao for introducing me to the project and Chi and Xinyu for assisting me with the design and testing of hardware. Thanks Aakash and Heinrich for assisting me with data processing. A special thanks to Karen, Hulgize and Greg: you guys leave no stone unturned to make FREEDM a resourceful, safe and friendly place to work at.

Nirmal and Bhumi, I wouldn’t be here without you. Thanks for your unwavering confidence in me and for always ensuring my well being.

Saurabh and Rohan, thanks for always being there. We have come a long way from where we began.

Bhavin and Manasi, I never missed my family when I was around you guys. Thanks for being with me through all the ups and downs. Soumya, thanks for all your guidance time and again.
Neha, this thesis wouldn’t be the same without you. Thanks for keeping me motivated.

To all my friends and colleagues: I have learnt something from each one of you and am glad that our paths crossed at some point of time.

I would like to express my sincere appreciation to my extended family: my aunts, uncles, cousins, nephews and nieces. I cannot thank you enough for your love and blessings.

Mom and Dad, I shall be eternally grateful for all your love and sacrifices for me. I thank you for your patience and understanding. I will always be indebted to you for your unconditional faith in me. Parth, even the shortest chat with you would lighten up my day and I feel blessed to have you as my brother.
# TABLE OF CONTENTS

**LIST OF TABLES** .......................................................... viii

**LIST OF FIGURES** .......................................................... ix

**Chapter 1 Introduction** .................................................. 1
  1.1 The FREEDM System .................................................... 2

**Chapter 2 Design and Operation of Distributed Energy Storage Device** ..... 6
  2.1 System Design .......................................................... 7
    2.1.1 Cascaded H-Bridge Series Inverter System ....................... 9
    2.1.2 Battery Management System ..................................... 11
    2.1.3 Communication and Application Development Platform .......... 14
  2.2 System Operation ..................................................... 21

**Chapter 3 Communication Architecture for DESD** .......................... 24
  3.1 DESD Architectural Configurations .................................. 25
  3.2 Evaluation of M2M and IoT Messaging Protocols ...................... 28
  3.3 Architecture and Implementation of the MQTT based Communication Network .................................................. 31
    3.3.1 Architecture of the FREEDM MQTT Network .................... 31
    3.3.2 Implementation of the MQTT based Communication Network .... 33
  3.4 Performance Test of the Communication Network .................... 40
    3.4.1 Test A: Network of Two Devices ................................ 40
    3.4.2 Test B: Network of Three Devices ................................ 43

**Chapter 4 Applications Developed for DESD** ................................ 46
  4.1 State of Charge Estimation .......................................... 46
    4.1.1 Determining the \( A_{b_{\text{ batt max}}} \) .................................. 48
    4.1.2 SoC Estimation Algorithm ....................................... 48
    4.1.3 Validation ...................................................... 54
  4.2 State of Charge Balancing Algorithm ................................ 54
  4.3 Integration with a Home Energy Management System .................. 59

**Chapter 5 Performance Benchmarking Tests for the DESD** .................... 61
  5.1 Reference Signal Tracking .......................................... 61
  5.2 Response Time and Ramp Rate ....................................... 62

**Chapter 6 Conclusion** .................................................... 64
  6.1 Accomplishments ..................................................... 64
  6.2 Future Work .......................................................... 65

**BIBLIOGRAPHY** .............................................................. 66

**APPENDICES** .................................................................. 70
Appendix A  Serial Communication Commands ........................................ 71
  A.1 DESD Turn ON Sequence ...................................................... 72
  A.2 DESD Turn OFF Sequence .................................................. 72
  A.3 Power Dispatch Commands .................................................. 72
  A.4 System Parameter Polling Commands .................................... 74
  A.5 Modbus Polling for BMS ..................................................... 75
Appendix B  Xbee Configuration for Master - Slave Operation ................... 76
  B.1 Xbee Master Configuration .................................................. 77
  B.2 Xbee Slave Configuration : Rx and Tx Enabled ....................... 77
  B.3 Xbee Slave Configuration : Only Rx Enabled ......................... 78
Appendix C  High Level Flow Chart of Software on Beagle Bone Black ........ 80
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Test Results for Communication Latency between two Clients representative of a DESD and an SST</td>
</tr>
<tr>
<td>3.2</td>
<td>Test results for communication latency between three clients representative of a DESD, SST and a Data Concentrator</td>
</tr>
<tr>
<td>5.1</td>
<td>Reference Tracking Performance of DESD</td>
</tr>
<tr>
<td>A.1</td>
<td>DESD Turn ON Sequence</td>
</tr>
<tr>
<td>A.2</td>
<td>DESD Turn OFF Sequence</td>
</tr>
<tr>
<td>A.3</td>
<td>Charge / Discharge Direction Command for DESD</td>
</tr>
<tr>
<td>A.4</td>
<td>Sample Power Command Dispatch Frames</td>
</tr>
<tr>
<td>A.5</td>
<td>Polling Commands for System Parameters</td>
</tr>
<tr>
<td>A.6</td>
<td>Range of Modbus Data Points for each Battery</td>
</tr>
<tr>
<td>A.7</td>
<td>Input Register Address of Modbus Data Points for Battery 1</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Figure 1.1</td>
<td>Electric grid with key elements of the FREEDM System. Adapted from [1]</td>
</tr>
<tr>
<td>Figure 1.2</td>
<td>Functional Diagram of an SST. Adapted from [1]</td>
</tr>
<tr>
<td>Figure 2.1</td>
<td>System Block Diagram of DESD</td>
</tr>
<tr>
<td>Figure 2.2</td>
<td>Topology of Cascaded H-Bridge DESD. Adapted from [2]</td>
</tr>
<tr>
<td>Figure 2.3</td>
<td>Power Stage System Configuration of Cascaded H-Bridge DESD. Adapted from [2]</td>
</tr>
<tr>
<td>Figure 2.4</td>
<td>Cell Composition Structure of the Battery</td>
</tr>
<tr>
<td>Figure 2.5</td>
<td>Functional Block Diagram of LTC6803-2. Adapted from [3]</td>
</tr>
<tr>
<td>Figure 2.6</td>
<td>(a) RC Filter Circuit for Cell Module Voltage Detection and (b) RC Filter Circuit for Battery Current Detection</td>
</tr>
<tr>
<td>Figure 2.7</td>
<td>Complete setup of BMS system</td>
</tr>
<tr>
<td>Figure 2.8</td>
<td>Single SPI Master communicating with multiple Slaves over an isolated network. Note the need for a dedicated $C_{S}$ signal for each Slave</td>
</tr>
<tr>
<td>Figure 2.9</td>
<td>Single I2C Master communicating with multiple Slaves over an isolated network. SDA (Data) line is bidirectional requiring sophisticated isolation provisioning</td>
</tr>
<tr>
<td>Figure 2.10</td>
<td>Multi node isolated CAN bus</td>
</tr>
<tr>
<td>Figure 2.11</td>
<td>Modbus Communication over Serial Line (UART) using XBee</td>
</tr>
<tr>
<td>Figure 2.12</td>
<td>(a) Beagle Bone Black. Adapted from [4] and (b) XB24-AWI-001. Adapted from [5]</td>
</tr>
<tr>
<td>Figure 2.13</td>
<td>System operation with a two-tier controller scheme</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>SSTs directly controls their respective DESDs as a means to match demand-supply constraints. The DGI software manages IEM algorithms on SST to control the power flow operation</td>
</tr>
<tr>
<td>Figure 3.2</td>
<td>DESD as a part of a smart home, controlled by an intelligent HEMS</td>
</tr>
<tr>
<td>Figure 3.3</td>
<td>Example of an MQTT Network using hierarchical topics to organize messages</td>
</tr>
<tr>
<td>Figure 3.4</td>
<td>Multi-Tiered Network Architecture of the FREEDM System</td>
</tr>
<tr>
<td>Figure 3.5</td>
<td>Device Profile of a DESD</td>
</tr>
<tr>
<td>Figure 3.6</td>
<td>Data flow path between an SST and DESD over the MQTT Network</td>
</tr>
<tr>
<td>Figure 3.7</td>
<td>Hardware - software platforms for different test cases in a two device network</td>
</tr>
<tr>
<td>Figure 3.8</td>
<td>Test setup for a three device network</td>
</tr>
<tr>
<td>Figure 4.1</td>
<td>Calculation of the residual energy capacity in the DESD based on the SoCs of cell modules that make a battery</td>
</tr>
<tr>
<td>Figure 4.2</td>
<td>Charge characteristics for a fully discharged battery cell module to calculate its Ah capacity. Blue lines demarcate the integration limits for charging current against time</td>
</tr>
<tr>
<td>Figure 4.3</td>
<td>Battery Model with internal resistance, relaxation effect and Voc - SoC relationship</td>
</tr>
<tr>
<td>Figure 4.4</td>
<td>The experimental curve for Voc - SoC</td>
</tr>
</tbody>
</table>
Figure 4.5  SoC, Charging current and Voltage for a cell module approaching full charged state. The intermittent fall in current readings can be attributed to measurement noise .................. 53
Figure 4.6  Block Diagram for SoC Balancing Algorithm .......................... 57
Figure 4.7  Convergence of Battery Pack SoCs during a discharge operation .... 58
Figure 4.8  Integration of DESD with the HEMS System .......................... 60

Figure 5.1  Response Time and Ramp Rate Calculation. Adapted from [6] ....... 63
Figure 5.2  Response Time observed for the DESD ............................... 63

Figure C.1  Flowchart for application software on BBB ......................... 81
Chapter 1

Introduction

The existing electric grid infrastructure, deployed several decades ago, has essentially remained the same over the years in terms of its architecture and design. The fundamental law governing its design was to transfer power from a centralized generation system to the end user through a series of distribution transformers. While this infrastructure has served us well over the years, it is not suitable to support a number of recent developments in the power generation and distribution domain. Firstly, the world is making a slow transition to utilizing non-conventional energy resources, such as wind and solar power, thanks to increasing environmental consciousness and the realization that conventional energy resources such as coal and gas are fast depleting. However, the intermittent nature of such renewable energy resources can cause serious stability issues with the grid [7, 8, 9], unless we have a better forecast data acquisition mechanism in place: something that’s absent from the existing grid infrastructure. Secondly, these non-conventional energy resources need not necessarily be deployed in a centralized fashion with large capacities, but rather they can be distributed in space, closer to the end user and in smaller capacities. Such a distributed power generation architecture goes against the design philosophy of existing grid which was meant to support only a unidirectional power flow. Thirdly, the power grid infrastructure has largely remained insulated from the tremendous progress made in recent years in the Information and Communication Technology (ICT) space.
We need to integrate these ICT technologies with the grid infrastructure to reap the benefits of real time data gathering, big data analytics and robust control.

This recognition that the needs of the 21st century are vastly different from the ones that the existing grid infrastructure was designed to serve, has led to extensive interest in the concept of Smart Grids (SG). The National Institute of Standards and Technology defines Smart Grid as “a modernized grid that enables bidirectional flows of energy and uses two-way communication and control capabilities that will lead to an array of new functionalities and applications”[10]. So SG shall be an enabling technology to allow bi-directional flow of both power and communication. It involves redesigning the hardware at each stage of transmission and distribution network to support a bidirectional power flow and redefining the terms of engagement between various devices that form this network which connects the end user to the grid. An SG shall allow easy integration of renewables to the electric grid by being adaptive enough to their dynamic nature while at the same time better managing the supply-demand constraints in real time.

1.1 The FREEDM System

The Future Renewable Electric Energy Delivery and Management (FREEDM) System [11] as being developed at the North Carolina State University and its partner universities, essentially envisions such an Energy Internet that is architected to allow automated and flexible electric power distribution. Power converters such as the Solid State Transformer (SST), Distributed Renewable Energy Resource (DRER) and Distributed Energy Storage Device(DESD) form the building blocks of FREEDM Systems power flow scheme. Figure 1.1 from [1] shows a possible layout of a FREEDM system. The DRER, DESD and loads are connected to an SST that also hosts an Intelligent Energy Management (IEM ) subsystem. The SST is a three port energy router that consists of a high voltage AC port and low voltage AC and DC ports (Figure 1.2). It not only performs the voltage step down function but also serves to route the energy bidirectionally. The IEM subsystem, consisting of the Distributed Grid Intelligence
(DGI) [12] software and communication interfaces uses the SST as the underlying hardware to serve as a power exchanger and an energy router. There can be several small and modular DRERs (powered by solar panels or wind turbines) and DESDs (based on batteries of any chemistry or even those of plug in hybrid vehicles or supercapacitors) that are connected to an SST and governed by the IEM algorithm(s) running on that SST. Such a cluster of power converters whose operation is governed by a local intelligence system, be it IEM or even a Smart Home Management System (for residential users) can be called an Energy Cell. The FREEDM grid is then composed of several such Energy Cells, each of whom try to achieve an optimal electric and economic performance at any given point of time. Considerable work has been done in the area of designing power electronics and control strategies for and amongst each of these converters, but the communication interface between them has been rather loosely defined. While a Distributed Grid Intelligence (DGI) Operating System that would allow peer to peer communication amongst different DGI nodes is under development [13, 12], it is the
This thesis identifies the communication needs specifically for a device that is meant to operate within an Energy Cell. We use a 1 kW / 2.7 kWh DESD as a template device while proposing a comprehensive framework to connect the different devices within an Energy Cell. While all the testing and validation for the proposed communication design has been performed with the said DESD, it is portable and generic enough to be adopted by any of the other devices (such as a DRER) in the energy cell. Additionally, this thesis also demonstrates the development of multiple applications that can be deployed on the DESD to support its various possible use cases with the FREEDM System. Finally, we also derive certain performance parameters for characterizing the DESD in order to provide a complete picture of its performance to any entity that seeks to control it.

The remainder of this thesis is organized as follows: Chapter 2 describes the DESD design and operation in detail. This DESD hardware was first developed by Sanzhong Bai [2]. That setup has since been enhanced to a higher power rating and energy capacity. It also describes the design of a battery data acquisition system which collects the raw data to estimate the State of Charge (SoC) of the batteries. A two-tier embedded controller solution consisting of DSPs and a single board computer, the Beagle Bone Black (BBB) is described as a means to
meeting all the computational and communication needs of the DESD. The choice of Modbus
protocol for communication within the DESD (between the DSPs and the BBB) is also justified
here. Chapter 3 details the role of the DESD in a FREEDM system and the different use cases
it might need to support. Requirements for a suitable external communication interface are
identified and addressed by assessing the relevant industry standard protocols. Message Queue
Telemetry Transport (MQTT), a messaging layer communication protocol based on TCP/IP is
finally chosen and implemented after a careful consideration of system requirements. Relevant
implementation details and test results are discussed. Chapter 4 elaborates on the algorithm
and implementation details for three different applications viz. State of Charge Estimation,
State of Charge Balancing and Integration with a Home Energy Management System (HEMS),
that we have demonstrated to work with the DESD. In Chapter 5 we present the results of some
characterization tests performed on the DESD to benchmark its performance. Finally Chapter
6 concludes the thesis with a summary of the accomplishments and suggesting a road map for
future work.
Chapter 2

Design and Operation of Distributed Energy Storage Device

Conventional energy storage installations have used mechanisms such as pumped hydrostorage, compressed air storage, spinning flywheel and lead acid or sodium sulphur based batteries[14]. While these are suitable for a large scale, centralized or utility controlled installations, not all of them scale down well to support a distributed architecture with smaller capacities. Battery based solutions, albeit of varying battery chemistries, dominate this space due to their ubiquitous availability and easy scaling. While lead acid has typically been the dominant battery type in all kinds of industrial applications, a lot of recent efforts have been focused on exploring other battery chemistries for energy storage applications to achieve higher energy density without compromising on safety and reliability[15, 16, 17, 18].

This chapter gives an overview of the design and features of the DESD used for this work. The modular control scheme of the DESD as described in Section 2.1 lends itself well to having low voltage batteries of different chemistries and capacities in the same unit. Currently the unit is powered with Li-ion batteries reclaimed from Hymotion L5 conversion module for a Toyota Prius hybrid electric vehicle. This DESD is designed to exchange power with the 120 V AC
port of any power system. So as a part of the FREEDM Energy Cell, it can be interfaced to the 120 V AC port of an SST (at a community level) or it may also be used directly by the residential end user as a part of a Smart Home. It also incorporates a Battery Management System (BMS) that provides real time battery cell module voltage, current and temperature data to estimate the State of Charge (SoC) of the batteries. The relevant architecture and implementation details are described in the sections below.

2.1 System Design

With reference to Figure 2.1 below, the DESD broadly consists of 3 sub-systems that work in tandem to deliver the overall functionality:


2. Battery Management System: Consists of analog front end and a DSP to acquire raw data of cell voltage and current from each battery in the system.

3. Communication and Application Development Platform: Consists of an ARM based single board computer, the Beagle Bone Black (BBB), to run native algorithms and also act as a communication gateway.
Figure 2.1 System Block Diagram of DESD
2.1.1 Cascaded H-Bridge Series Inverter System

The AC DESD device uses a cascaded H-bridge topology as described in [2, 19]. In Figure 2.2 each block represents a modular and bi-directional H-bridge inverter interfaced to it’s battery. In the actual setup elaborated in Figure 2.3, four such converters are connected in series so that they share the AC current while their respective AC voltages add up to form the interface voltage with the grid (via a 1:2.5 transformer). The main advantages of such a topology are as below:

1. Modular design with independent control of each module.
2. Batteries with dissimilar chemistries and potentials can be combined in a single unit.
3. No need of a centralized controller, which leads to a scalable architecture.
4. No need for communication between modules as each of them independently follow their own control law.

![Figure 2.2 Topology of Cascaded H-Bridge DESD. Adapted from [2]](image-url)
Figure 2.3 Power Stage System Configuration of Cascaded H-Bridge DESD. Adapted from [2]

The overall control strategy is as below:

1. Module 1 solely controls the AC current that flows through the series string of H-bridge converters. The reference signal for this AC current is derived from the power dispatch command and the sensed grid interface voltage.

2. Modules 2, 3 and 4 control their respective additive contributions in sharing the grid interface voltage. The overall grid interface voltage is thus the sum of those generated by modules 1 to 4, but directly controlled by only modules 2, 3 and 4 (since module 1 operates with AC current as the control parameter). The voltage contribution of any of modules from 2, 3 and 4 is directly proportional to it’s net power contribution to the total power command. Thus, in the simplest case when all the modules are sharing the total power equally amongst themselves, the voltage reference for each of the modules 2, 3 and 4 is also equal.

All the modules receive the isolated signal for grid interface voltage. They natively apply zero crossing detection to achieve phase synchronization of their respective sinusoidal PWM generation signals. The digital control of this system is performed by a DSP (TMS320F28335, Texas Instruments [20]) for each module in the cascade string.
The above work was accomplished by Sanzhong Bai as a part of his PhD dissertation. A complete formal description of the control strategy with additional implementation details can be found in [2, 19].

2.1.2 Battery Management System

Architecture

The DESD consists of four H-bridge converters, each supported by a stack of two Lithium ion batteries in parallel. Thus the system consists of a total of eight batteries. We have two batteries in parallel for each H-bridge module to raise the system capacity and provide better transient response, considering that the batteries being used are reclaimed ones.

Each battery consists of a series string of eight cell modules, each rated for 3.3 V nominal. Each of these cell modules physically consists of eleven ANR26650M1A cells (A123 Systems [21]) in parallel. The cell composition structure that makes up a battery is thus shown below in Figure 2.4 The analog front end to acquire raw data of module voltages and current for each battery is based on the LTC 6803-2 IC (Linear Technologies [3]. It is designed to perform analog to digital conversion of sensed battery parameters from a stack of battery cells. A PCB board consisting of the LTC 6803-2 IC and it’s associated circuitry for communication and isolation

![Cell Composition Structure of the Battery](image)

Figure 2.4 Cell Composition Structure of the Battery
is mounted on each of the batteries. A single DSP controller communicates with all of these 
BMS boards to acquire raw data of cell voltages and battery currents. The LTC 6803-2 IC 
supports Serial Peripheral Interface (SPI) communication protocol, so all the BMS boards are 
connected over a shared SPI bus with the DSP. Since the LTC 6803-2 IC also has provision for 
4 bit addressing, it’s easy to individually address each of the BMS boards by assigning them 
a unique 4 bit address in hardware. The DSP then polls each of the BMS boards in a cyclic 
manner to gather the raw data associated with it’s battery.

Implementation

This section describes the hardware implementation details for the BMS system.

The BMS board is capable of measuring all the eight cell module voltages, the net battery 
current and also ambient temperature. As seen in the functional diagram of the LTC 6803-2 IC 
[3] in Figure 2.5, it has provision to measure cell voltages of upto 12 stacked cells in any battery. 
We use it for monitoring the 8 cell modules of the battery used by this DESD. A simple low 
pass RC filter with cut-off frequency of 16 kHz (R = 100 Ω, C = 0.1 µF) recommended in [3] 
as optimal, is used for cell voltage measurement (Figure 2.6a).

The LTC 6803-2 also has provision for interfacing two external temperature sensors at 
terminals V_{TEMP1} and V_{TEMP2}. We use one of those terminals for actually interfacing a Hall 
Effect current sensor, ACS759LCB-050B (Allegro MicroSystems LLC, [22]) , to measure the 
battery current. However, the battery current is actually not a clean DC signal, but rather a 
rectified sinusoidal signal with 60 Hz as the dominating fundamental frequency. This is because 
the battery is directly interfaced with the H-bridge converter. Since we are interested in the 
average value of the signal for calculating the SoC of the batteries, we feed the signal to a 
low pass RC filter with a cut-off frequency of less than 2 Hz (R = 22 kΩ, C = 4.7 µF) before 
applying it to the V_{TEMP1} pin of LTC6803-2 (Figure 2.6b). The output of the current sensor is 
biased to half of it’s supply voltage, thereby enabling bi-directional sensing of battery current.
Figure 2.5 Functional Block Diagram of LTC6803-2. Adapted from [3]

Figure 2.6 (a) RC Filter Circuit for Cell Module Voltage Detection and (b) RC Filter Circuit for Battery Current Detection
While the board has provision for implementing an NTC type temperature sensor, we do not use one currently as it’s not required for any of the implemented algorithms. However, we have evaluated MF52C1104F4150 (Cantherm [23]) to be suitable for measuring ambient temperature.

In order for all the boards to share the same SPI communication bus with a DSP, ISO7220A and ISO7221A (Texas Instruments [24]), high speed digital isolators from TI, are used. They provide an isolated communication interface for the LTC6803-2 ICs which are powered from their respective batteries. Figure 2.7 shows the actual setup of BMS system with all the eight batteries with their respective BMS boards and the DSP that controls all of them.

2.1.3 Communication and Application Development Platform

While the DSPs are powerful enough to handle all the power electronics related sensing and control operations, a dedicated central controller is required to act as an external communication interface for the DESD. This central controller not only forms a TCP/IP based network with
other devices in the FREEDM System (described in Chapter 3), but also serves as a platform for implementing native algorithms and applications for the DESD (described in Chapter 4). We chose the Beagle Bone Black (Figure 2.12a), a low cost embedded Linux based single board computer for this purpose. It is powered by the AM3358BZCZ100 processor from TI, based on the ARM Cortex A8 architecture. It runs a Debian distribution of Linux out of the box. On the hardware side, it features 512 MB of RAM, 4 GB of Flash and a 1 GHz clock.

**Internal Communication between the BBB and DSPs**

The BBB handles internal communication with the DSPs to issue them power commands and poll system parameters at run time. It also communicates with the BMS DSP to fetch raw values of sensed cell module voltages and current for each of the batteries. Several board level embedded communication options such as Inter Integrated Circuit (I2C) [25], Serial Peripheral Interface (SPI) [26], Controller Area Network (CAN) [27] and Modbus over Serial line [28, 29] were evaluated to chose a suitable mechanism to handle this internal communication between the BBB and DSPs. The most significant factors to consider in this context were:

1. The communication protocol should have minimal processing and computational requirements as the DSPs could be running power control loops at frequencies in the order of tens of kHz, which implies a system interrupt atleast every 100 µs. The digital control loop’s execution time typically runs in tens of microseconds thereby leaving only so much more time for the DSP to handle other housekeeping and communication tasks.

2. Since the communication between the BBB and DSPs shall be performed in an electrically noisy environment as the power converter(s) would be operational, it should have built-in defence mechanisms for error identification, else we run the risk of undetected data corruption which could lead to unsafe operation.

3. It should be a modular and scalable mechanism as a single BBB would need to communicate with multiple DSPs. Adding additional DSPs to the system for any kind of
future enhancements should be smooth.

4. Topology wise, it would essentially be a Master-Slave mechanism with the BBB as Master and all the DSPs as slaves. So the BBB should have the flexibility to either broadcast it’s commands to all the slave DSPs or individually address them if required.

5. The DSPs would be electrically isolated from each other as well as from the BBB as they all could be powered from different sources. The underlying physical layer for communication should support such a scheme.

**SPI**

We begin our evaluation with SPI, it being the simplest protocol amongst all possible options. However it does not meet most of the above requirements. It’s not scalable (requires a dedicated chip select pin $\text{CS}$ for each slave), doesn’t have any inherent error checking, cannot be used for broadcast communication and would lead to complicated wiring for isolated communication as each node would require at least a four channel digital isolator.
Figure 2.8 Single SPI Master communicating with multiple Slaves over an isolated network. Note the need for a dedicated $CS$ signal for each Slave.

**I2C**

I2C was originally designed for communication between an embedded controller and a peripheral IC like memory chips or external ADCs. While it does support a well defined multi-slave networking scheme, driving the bus for slaves that are physically not on the same board is often challenging because the bus lines are passively driven with pull up resistors. External drivers can be used but then isolation is an issue because the data line is bi-directional.
Figure 2.9 Single I2C Master communicating with multiple Slaves over an isolated network. SDA (Data) line is bidirectional requiring sophisticated isolation provisioning.

**CAN**

CAN bus has been designed to provide robust performance in noisy environments with features such as differential bus signalling, multi-node communication, sophisticated mechanism for error detection and fault handling and high speed communication throughput (a baud rate of up to 1 Mbps is possible). It makes for a viable choice but given it’s flexibility and features, it is considerably more complicated to implement and troubleshoot. Moreover, as each board would end up requiring an isolator and a CAN transceiver, it would lead to non-trivial wiring.

Figure 2.10 Multi node isolated CAN bus
Modbus over Serial Line

Finally, we consider the Modbus over UART scheme. Modbus is an open standard protocol quite popular in industrial control applications. It defines a Client-Server model where multiple servers could be connected to the same Client, which maps perfectly with the desired Master-Slave scheme. It follows a query-response architecture that is supported by an acknowledgement and a 16 bit Cyclic Redundancy Check (CRC) mechanism for fault identification. Modbus, fundamentally being an application layer protocol in terms of the OSI model, is supported by several physical layer options ranging from RS-232 to Ethernet. For typical embedded microcontroller multi-slave applications Modbus is implemented over an RS-485 physical layer bound by the UART channels of controllers, which could be isolated if required. We instead opt to use XBee radios with the controllers to greatly simplify isolation and assembly. XBee radios are manufactured by Digi International and they communicate over an RF network based on the IEEE 802.15.4 standard [30]. We use the XB24-AWI-001 modules [5] (Figure 2.12b) due to their reasonable power consumption (50 mA) and easy to implement UART interface with the controllers. So all the DSPs and the BBB have a XBee radio mounted to their respective UART ports such that they form a multi-drop network. As far as the controllers are concerned, they just work with their respective UART ports (where the XBee radios are interfaced) without really bothering about the management of the wireless network, as that’s handled completely by the XBee radios themselves. Using XBee radios also gets us around the grounding and isolation issues that would otherwise come up for communicating across several DSPs.
In the current implementation for the DESD, not all the DSPs communicate with the BBB over Modbus though. This is due to legacy reasons as the DSPs controlling power electronics already had their communication interface defined previously. Thus only the BMS DSP communicates with the BBB over Modbus. However, since the underlying physical layer is still UART for either of the cases, its indeed possible to have proprietary and Modbus implementations working on the same network with appropriate safeguards in place for preventing misinterpretation of data. (Appendix A).

**Implementation**

Modbus needs to be implemented on both Master as well as Slave sides. While the master side code runs on the Linux platform, the slave side code runs on the DSPs. Accordingly, dedicated software meant for each of the platforms needs to be used.

We used the *FreeModbus* library [31] as the base code for the slave side. FreeModbus provides an open-source implementation of Modbus slave functionality for the slave devices with ready
Figure 2.12 (a) Beagle Bone Black. Adapted from [4] and (b) XB24-AWI-001. Adapted from [5].

ports available for various platforms such as for controllers from Atmel, Freescale, TI etc. However, it wasn’t ported for the DSP (TMS320F28335) used in the DESD. Therefore the FreeModbus library was ported for the DSP being used in this project.

The software development on BBB can be done in a variety of languages such as C, C++, Python etc. We chose Python and accordingly the `modbus-tk` package [32] for Python was used to implement the Modbus Master routines. It’s an open source implementation of Modbus master developed primarily by Luc Jean.

### 2.2 System Operation

The BBB board controls the operation of DESD by issuing appropriate commands to the DSPs to actuate the hardware. As previously stated, this communication between the BBB and DSPs is conducted using XBee radios over the wireless RF network. The XBee radios are appropriately configured such that the one connected to the BBB acts as a Master and all the others connected to the DSPs act as slaves (Appendix B). All the communication between the
Master - Slave modules is then handled in a command/query-response fashion.

At run time, the BBB is expected to receive a power dispatch command for the DESD from an external device as a result of an intelligent power or energy management algorithm running on that external device. Once the BBB receives this dispatch command, which could be either a charge or discharge command, it formulates the corresponding data frames for each of the DSPs (Appendix A) and transmits them via the XBee network.

The BBB also periodically polls the raw battery cell module voltage, current and temperature (V,I,T) data from the BMS DSP and processes it at a sampling rate of 1 second to run the SoC estimation algorithm. Further details about SoC estimation algorithm are provided in Section 4.2. As shown in Figure 2.13, we have a two tier controller system in place consisting of the DSPs that directly control the hardware and the BBB board that acts as a communication gateway for the DESD and also as a platform for developing native applications. This kind of a layered architecture of controllers suits even a DRER or an SST kind of power converter as they both need to perform communication and control operations just like the DESD. It is noteworthy that the BBB neither directly interacts with the hardware nor handles any sensing or control operation for the power electronics. This sort of decoupling leads to a modular design as any of the sub components can be later changed or upgraded with minimal changes in rest of the system. Not only that, the capability of the BBB to communicate with multiple slave DSPs also makes the design reasonably scalable as additional DSPs can be added in any converter (for enhanced functionality).
Figure 2.13 System operation with a two-tier controller scheme
Chapter 3

Communication Architecture for DESD

The DESD is designed to work in conjunction with other smart power converters that form the FREEDM Energy Cell so that it can play its role in acting as a reservoir of energy that’s charged or discharged strategically. Considering this need for close coordination amongst the different power converters within the Energy Cell, it’s essential that a standard communication interface be designed to simplify system integration. We begin our analysis in Section 3.1 by investigating the various possible architectural configurations for the DESD in the FREEDM System which would allow us to identify the communication requirements for it. In Section 3.2 we perform a comparative analysis of various standard protocols and their suitability for usage with the DESD. Section 3.3 describes the implementation details of the communication architecture arrived based on the issues identified so far and Section 3.4 discusses the performance results for communication latency tests and analyses the factors affecting it.
3.1 DESD Architectural Configurations

A DESD could potentially be deployed at any level of hierarchy within the power distribution network. Two representative cases could be:

1. Community Energy Storage Device: A community of end users which is served by an SST type of Energy Router could be supported by one or more DESDs to deliver functionalities such as peak shaving, load levelling, frequency regulation or voltage control. In such use cases, the DESD is used as a captive energy source to mitigate intermittent fluctuations in demand and supply of power thereby ensuring a smooth and reliable provision of ‘grid’ power to the end user. The SST shall be directly governing the operation of the DESD based on the IEM algorithms running on it.

2. Residential Energy Resource: The DESD could also be located closer to the end user by serving a smart house or a smart building system directly. Here it is more likely to be used in conjunction with a DRER based on solar or wind power. Typical use cases would include load shifting during peak demand hours, smoothening of intermittent power

Figure 3.1 SSTs directly controls their respective DESDs as a means to match demand-supply constraints. The DGI software manages IEM algorithms on SST to control the power flow operation.

2. Residential Energy Resource: The DESD could also be located closer to the end user by serving a smart house or a smart building system directly. Here it is more likely to be used in conjunction with a DRER based on solar or wind power. Typical use cases would include load shifting during peak demand hours, smoothening of intermittent power.
generation by DRER or as a back up for uninterrupted power supply (UPS) schemes. The DESD is essentially used to make economically optimal use of the grid power. In such a case, the DESD could be controlled by a Home Energy Management System (HEMS) or a Building Automation System (BAS). It is noteworthy that in such cases, the HEMS or BAS could in turn be coordinating with an SST like device to negotiate the power flow direction and quantum in real time.

Irrespective of the way a DESD is deployed, it needs to be intelligently controlled by an external entity. We therefore need a reliable Machine to Machine (M2M) messaging mechanism to interface with the DESD. Since the ethernet based TCP/IP network is pervasive and growing, thanks to the Internet of Things (IoT) phenomenon, it’s but natural to look for a solution from this domain to meet our communication needs. A BBB like single board computer serves this functionality adequately as the underlying operating system provides the necessary resources to deploy any such messaging infrastructure.

Another observation that one can make considering the above usage scenarios is that the DESD and other devices within the FREEDM Energy Cell would need to form a local network to
identify and interact with each other. The characteristics of this network would be data transfer of small packet size but high frequency, low communication latency (for effective control) and support for devices across the power/ performance spectrum: from wireless sensor networks and embedded single board computers to sophisticated supervisory platforms and servers.

Based on the use cases described above and the operational details of the DESD within the Energy Cell, we can identify the following requirements for the M2M messaging protocol:

1. To control the DESD, short but frequent data transfer is needed. Status information (such as state of charge, residual energy capacity etc.) shall be frequently polled whereas commands to start/stop or dispatch power flow (direction and quantity) would be delivered to the DESD. So while the payload size would typically be only a few bytes, it needs to be exchanged periodically (typically in the order of seconds). So the messaging protocol should be efficient and have low overheads such that it can work even on low bandwidth networks.

2. The DESD might also need to communicate its status information to a SCADA like supervisory system. There could also be independent ‘observers’ on the network that could be maintaining a database or logging usage statistics for any applications. In such a scenario, the messaging protocol should have in place a mechanism to allow the DESD to communicate its state to more than one entity on the network. However, since such a supervisory mechanism could also be retrofitted as it does not directly affect the functionality of the DESD, we would need the messaging protocol to handle the situation with minimal changes on DESD’s end.

3. The chosen messaging protocol has to be application agnostic and scalable enough to become the glueing logic for not just the DESD but also DRERs and other devices within the Energy Cell. End users might also want to avail the real-time status of their devices using battery-powered devices such as phones or tablets. Some of these devices could
be having resource constrained hardware as their communication gateway and hence the messaging protocol has to feature a light weight implementation with minimal overheads.

4. Compliance with the IEC 61850 standard [33] is being actively pursued in substation automation domain. It defines a hierarchical data model to map a device’s physical attributes to the data points that the device can exchange with an external entity [34]. While devices such as the DESD do not necessarily fall within the scope of IEC 61850, it would be prudent to chose a messaging protocol that would support hierarchical object model so that compliance in future to similar or derived protocols is ensured.

3.2 Evaluation of M2M and IoT Messaging Protocols

Given the analysis presented in Section 3.1, we considered several M2M messaging protocols to evaluate their suitability for our application.

HTTP

The Hypertext Transfer Protocol (HTTP) [35] was the first protocol to be considered as it has been the de facto standard for information transfer over the Web. HTTP is designed to work on a Client-Server model as a request-response protocol. At the simplest level, the client essentially sends a request to the server to fetch or set data. It’s intended usage is for transferring sizeable chunks of data over the World Wide Web. Such an architecture is not suitable for our application as the underlying complexity and verbosity of HTTP would work against our design needs for a light weight protocol. Moreover, the Client-Server model for point to point communications doesn’t scale up well in a distributed environment where multiple nodes could be tracking the status of any given entity on the network.
CoAP

We next consider the Constrained Application Protocol (CoAP)[36], which is a simpler alternative to HTTP. It is optimized for platforms with constrained power and processing capabilities. It has a simplified header mechanism as compared to HTTP while still providing mapping methods to HTTP via defined proxies. It also supports multicast communication with resource discovery. However unlike HTTP which uses TCP, CoAP uses UDP to manage the Transport Layer. This comes with its own drawbacks as retries and re-ordering need to be implemented in the application stack. CoAP also provides options for ensuring reliability (two qualities of service: ‘conformable’ and ‘nonconformable’) and resource discovery. We considered CoAP to be a viable option while continuing to explore other alternatives.

AMQP

Advanced Message Queuing Protocol (AMQP) [37] is another application layer protocol initially developed by the financial industry. It’s an enterprise level feature rich protocol designed for message passing between servers. It follows a publish - subscribe (pub-sub) model for message passing. In a pub-sub design, the sender does not transmit message to any specific recipient, rather the message is published to a particular ‘topic’ which could be subscribed to by one or more peers on the network. This requires the use of a ‘broker’ on the network to manage message routing. Such an architecture is easy to scale as the message transmitter no longer has to bother about intended recipients because that is completely managed by the broker. Moreover, a topic based messaging mechanism easily renders itself to hierarchical messaging as hierarchical topics can be formed which can then be selectively subscribed to by interested recipients. AMQP is also quite comprehensive in the sense that it supports multiple messaging patterns like round-robin, message queuing, store-and-forward etc. However, such a rich feature set comes at the cost of communication overhead (the smallest packet size is 60 bytes) and complexity.
MQTT

We finally turn to MQTT (Message Queuing Telemetry Transport) which was originally developed by IBM and recently (November 2014) was formalized as an open standard managed by OASIS (Organization for the Advancement of Structured Information Standards). MQTT provides a simple pub-sub model designed for resource constrained devices, typically embedded systems. While it’s simple to implement, it still supports three qualities of service: fire-and-forget (unreliable), at least once (to ensure a message is sent a minimum of one time) and exactly once. It also provides a hierarchical topic based message passing.

[38] provides a comprehensive overview of the underlying mechanism for working, reliability and security of some of the protocols discussed above. Based on these observations with regard to the various protocols considered, HTTP and AMQP were ruled out because of their processing overheads and high data bandwidth requirements. They wouldn’t be a suitable choice for transferring short but frequent messages across multiple nodes on a network. We eventually narrowed down the choice between CoAP and MQTT.

According to [39] MQTT experiences lower delays than CoAP for lower packet loss and higher delays than CoAP for higher packet loss. MQTT’s underlying layers being TCP as compared to CoAP’s UDP leads to simpler application layer as message reordering need not be explicitly handled. By providing a hierarchical topic based message transmission mechanism, MQTT can be suitably adapted in future to support IEC 61850 like protocols, though it would involve a fairly complex mapping from real world parameters to virtual data objects and eventually to topics for the messaging layer. MQTT also gels perfectly well with an allied protocol called MQTT-SN (MQTT for Sensor Networks) [40] which is a pub-sub protocol for Wireless Sensor Networks. It can be thought of as a version of MQTT which is designed for wireless communication environments. Such a provisioning could lead to easy integration, especially in the Smart Home context where the HEMS could possibly network with all the devices in the
home over MQTT-SN and with other devices in the Energy Cell using MQTT. Considering these issues, we selected MQTT for communications amongst the devices within an Energy Cell.

### 3.3 Architecture and Implementation of the MQTT based Communication Network

#### 3.3.1 Architecture of the FREEDM MQTT Network

**Overview of the MQTT Protocol**

MQTT is described as a pub-sub messaging protocol that uses TCP/IP under the hood. Any given MQTT network consists of one or more clients and a single broker. All the clients on a network communicate with each other via a broker. As such, all the clients are decoupled from each other and are connected only to the broker. An MQTT client could be any device, varying in computational power from a microcontroller to a server, that can run any implementation of the MQTT library on it. The client could be a publisher, subscriber or both. Publishers tag their messages with topics and transmit them to the broker. Subscribers let the broker know about the message topics they are interested in. It’s the broker that receives all the messages, filters them based on their respective topics and then forwards them to appropriate subscribers for each message. A powerful implementation of broker can handle subscription of thousands of clients. The broker is also responsible for handling authentication and security over the network. [41] is the complete specification of the protocol and [42] provides a comprehensive overview of the protocol from an implementation perspective. Moreover, one can also use TLS/SSL (Transport Layer Security/ Secure Sockets Layer) protocol for encryption purposes if required. MQTT also provides username/password based authentication mechanism for the clients to connect over the network.
Figure 3.3 Example of an MQTT Network using hierarchical topics to organize messages

As an example, consider a simple thermostat system for home that uses an MQTT network as shown in Figure 3.3. Clients A, B, C and D are connected to a central broker, and are agnostic about each other’s presence. Client A is a temperature sensor that publishes the real time temperature value to the network using the topic `Home/Sensors/Temperature`. Client B is subscribed to it because it manages the cooling system and uses the temperature sensor’s value for control purposes. Client B is also a publisher for the fan speed parameter which being acquired by a sensor, is published under the topic `Home/Sensors/FanSpeed`. Client C is basically a data logger and is therefore interested in all the sensor readings available over the network. As such, it subscribes to the broker using a wild card topic using an `*` as `Home/Sensors/*`. This basically tells the broker that Client C is interested in all the data published under the topic of 'Sensors'. Client D being a remote display is interested in only displaying the temperature data and therefore subscribes specifically to the topic `Home/Sensors/Temperature`. This makes the topic based pub-sub communication not only scalable by decoupling clients but is also organized and hierarchical.
MQTT Network in FREEDM Energy Cell

In the context of the FREEDM Energy Cell, we can think of all the possible devices within the Energy Cell as MQTT clients which network via a broker as shown in Figure 3.1. The broker, essentially being software driven, could be actually running on any of the client devices as well to reduce the device count on the network. So a possible configuration could be that all the DRERs, DESDs, FIDs etc. connected to a given SST can have an instance of MQTT client running on them. Now since all of them need to communicate at least with the SST, even the SST shall have an instance of MQTT client running on it. Moreover, the broker could be running from SST itself as a background process.

Another possible configuration for the Energy Cell could be as shown on the right hand side of Figure 3.4 where an SST directly networks with a Smart Home in a small community. Here we show an architecture that further exploits the choice of MQTT by having a tiered networking structure. Internal to the home, all the smart sensors and appliances could be communicating to the HEMS via MQTT-SN or MQTT as applicable. We assume the presence of an appropriate Gateway device to couple the two networks. And then the HEMS as a single entity could network with the SST to arrive at an optimal power usage operation.

Looking at a possible overall FREEDM communication architecture in Figure 3.4, the communication across SSTs is handled using TCP/IP sockets as managed by DGI. The SSTs could also need to communicate to a SCADA server or any central command centre. Since this falls within the ambit of a distribution network, a protocol like DNP3 or IEC 61850 is ideal for this purpose.

3.3.2 Implementation of the MQTT based Communication Network

We have used open source tools for our implementation of MQTT. The broker services were provided using Mosquitto [43], whereas Client services were built on top of the Python
Device Profile

MQTT doesn’t specify any data model for messaging as a part of it’s standard. It’s completely up to the application to specify the message format and it is assumed that the receiver is aware of this message format. We devised a simple data model for prototyping purposes which can be used by any of the small scale distributed entities within the Energy Cell.

We start with listing all the static and dynamic attributes of interest for the device in a spreadsheet in a predefined format. This spreadsheet is called as the Device Profile of the device. The device attributes could be either static (such as Device Name, Model, Version etc.) or dynamic (Active Power , Temperature, Discharge Capacity etc.). Static attributes do not influence the run-time behavior of the device whereas dynamic attributes either control or report the run-time behavior of the device. Furthermore, the dynamic attributes could either be analog or digital in nature. And they could as well be a status report (OUT) of the device
or a command (IN) to the device by one of its peers in the network. Thus we come up with further classification of device attributes as AIN, AOUT, DIN and DOUT. Static attributes are correspondingly tagged by a field called DEV.CHAR (device characteristics). Different attributes belonging to the same tag are identified by their indices (starting with 0). The said representation borrows heavily from the DNP3 representation of data points. This spreadsheet also lists the default, minimum and maximum values for all the attributes which are used at start up (for initialization) and run time (for boundary check). A sample device profile made for the DESD is shown in Figure 3.5. The device profile is stored locally on the device’s BBB or equivalent board.

**Data Flow Mechanism**

While the Device Profile provides an easy to manage interface for the programmer to add or remove data points from the device's object model, it is not used directly during run time as working with spreadsheets at run time could be computationally intensive. Instead, we use a data format called JSON (JavaScript Object Notation) to manage the data points at run time. JSON defines a format for specifying an order collection of name/value pairs which is lightweight (in terms of parsing) and easy to read/write (in terms of implementation as well as form human visibility). Such a file interface to manage data points is required because at run
time the MQTT client as well as any native applications on the device could read/write into the data points. The JSON file then serves as a consistent means of storing the state of the device across all the applications running on it. A set of API (Application Program Interface) routines were developed to read/write specific data attributes from the JSON file with inbuilt checks for atomic access (using file lock mechanisms), write protection and boundary checking. Any application, including the MQTT Client, would be using these routines to access the data attributes thereby avoiding any race conditions.
Figure 3.6 Data flow path between an SST and DESD over the MQTT Network
Figure 3.6 describes the data flow path across devices and within their respective controllers. For any given device (like for the SST and DESD shown), there could be multiple applications, such as the MQTT Client, the Modbus implementation or any other native application (App) that could be the producer or consumer of data points stored in the JSON file (device attributes). While the Modbus code deals with translation and transfer of data points within the device, MQTT Client handles the transfer of data points external to the device. As such, with regard to the two-tier controller architecture described in Section 2.2, we can generically call the DSPs as controllers for power electronics and BBB as an Application Platform (which could possibly be any device with an operating system and a networking communication stack) for the purpose of this discussion.

Consider the device communication between just the DESD and an SST for simplicity. To begin with the DESD, at the simplest level, the DESD expects to receive commands for start/stop, power dispatch etc. from the SST and the SST receives the residual energy left in the DESD as a status update from it. All of these data points are listed in the Device Profile for DESD. For example, start/stop being a binary value is considered a DIN because it is received as an external Digital Input. Similarly, power command to the DESD is an AIN data point (Analog Input) whereas the residual energy calculated by the DESD based on it’s BMS system is quantified as an AOUT (Analog Output) data point. At the beginning of program execution, the DESD’s application platform (BBB in our case) translates the Device Profile (which is a spreadsheet) into an equivalent JSON format. This JSON file is also published over the network so that the devices that are interested in controlling or monitoring the DESD can do so based on the data points published therein. Any native application that wants to read the AIN/DIN data points or write to AOUT/DOUT data points would be using the previously mentioned API routines developed to handle the JSON file in order to do so. The Modbus application would need to periodically poll for any updates in the data points that need to be transferred to the DSPs and as well write to those data points which need to be updated based
on the data received from the DSPs. The MQTT Client would be tracking any changes that happen on the JSON file and if any of the AOUT/DOUT data points is updated by Modbus or any other native application, then the MQTT Client shall push it over the network to the broker with full hierarchy of the data point as the message topic. For example, if it wants to publish the quantity of residual energy left in the DESD over the network, it would publish the data point called ‘Discharge_Capacity’ which is enumerated as an AOUT with index 0 in the Device Profile, as a message with topic ‘DESD/1/AOUT/0’. The actual name of the quantity is not required as a part of the message topic because the type (AOUT) and enumeration index (0) are enough to uniquely identify the data point and it’s source (DESD_1).

From the perspective of the application platform on SST, as soon as the DESD application platform joins the MQTT network, it receives DESD’s Device Profile in the form of JSON file because it would have subscribed for the same with the broker. The SST’s application platform would also have another JSON file which it would have generated by itself based on its own unique Device Profile. However, that JSON file is not relevant to the discussion here because it plays no role in communicating with the DESD. As such, any device that controls another device, shall have its JSON file in addition to its own JSON file. The MQTT Client on SST has a two fold task: one, to subscribe to the relevant AOUT/DOUT topics of DESD so that anything that the DESD publishes for those data points is received by the SST and second, to publish to DESD’s AIN/DIN data points in order to control it. The static characteristics of the DESD, under the category of DEV_CHAR are not subscribed to because they are not updated/published during run time, instead they are read directly from the JSON file received at the beginning of the communication.

In this way, MQTT Clients on all the devices connected to the network essentially seek to synchronize the data points for each device via updates in their respective JSON files. Applications running natively on the devices can then read or write to the data points by just updating the JSON file’s corresponding data field. They are agnostic to the underlying
communication network.

Thanks to Siddhartha Kumar for developing the APIs to read/write JSON files concurrently while using MQTT as the underlying communication network.

### 3.4 Performance Test of the Communication Network

We performed a set of tests to characterize the performance of our implementation of the MQTT network and identify the critical factors affecting it. The various test scenarios, their associated results and conclusions are as presented below.

All the hardware boards used during the tests were initially time synchronized to NCSU’s NTP servers so that they can all be trusted to follow the same clock. Moreover, they are all connected to NCSU’s ethernet network. Specifically, the Android phone and Ubuntu laptop are connected via Wi-Fi whereas the BBB boards were connected physically to an ethernet switch (TE-100 S8, Trendnet). the MQTT communication for all the tests was handled with QoS value of 1, which ensures message delivery of ‘atleast once’.

#### 3.4.1 Test A: Network of Two Devices

To model a communication network between just two devices, we used a variety of hardware and software platforms as described in Figure 3.7. We performed tests to calculate the average communication latency to transfer a data point from Client A to Client B. To represent analog and digital data points, we performed the tests with message sizes of 10 bytes and 3 bytes respectively. This is because, the minimum header size for MQTT is 2 bytes and therefore considering an additional byte to represent digital data, the message size for it is 3 bytes. For the analog data, the actual message size depends on the range and resolution of the number, however, 10 bytes is considered representative of a typical data point. Table 3.4.1 lists the average communication latency in milliseconds to transfer a message from from Client A to Client B.
Figure 3.7 Hardware - software platforms for different test cases in a two device network
Table 3.1 Test Results for Communication Latency between two Clients representative of a DESD and an SST

<table>
<thead>
<tr>
<th>Platform</th>
<th>Test 1</th>
<th>Test 2</th>
<th>Test 3</th>
<th>Test 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client A</td>
<td>Intel Core i3-2370M 2.4 GHz Ubuntu LTS 14.04</td>
<td>Intel Core i3-2370M 2.4 GHz Ubuntu LTS 14.04</td>
<td>BBB ARMv7 Processor 300MHz Debian 7.8</td>
<td>BBB ARMv7 Processor 300MHz Debian 7.8</td>
</tr>
<tr>
<td>Broker</td>
<td>Qualcomm Snapdragon S4 Pro (ARMv7) @ 1.5GHz Android v5.0 (Lollipop) MQTT Broker Pro</td>
<td>Intel Core i3-2370M 2.4 GHz Ubuntu LTS 14.04</td>
<td>Intel Core i3-2370M 2.4 GHz Ubuntu LTS 14.04 Mosquitto Broker</td>
<td>Qualcomm Snapdragon S4 Pro (ARMv7) @ 1.5GHz Android v5.0 (Lollipop) MQTT Broker Pro</td>
</tr>
<tr>
<td>Client B</td>
<td>BBB ARMv7 Processor 300MHz Debian 7.8</td>
<td>BBB ARMv7 Processor 300MHz Debian 7.8</td>
<td>BBB ARMv7 Processor 300MHz Debian 7.8</td>
<td>BBB ARMv7 Processor 300MHz Debian 7.8</td>
</tr>
<tr>
<td>3 Bytes (ms)</td>
<td>192.985</td>
<td>8.455</td>
<td>15.542</td>
<td>335.132</td>
</tr>
<tr>
<td>10 Bytes (ms)</td>
<td>239.129</td>
<td>9.720</td>
<td>16.761</td>
<td>359.987</td>
</tr>
</tbody>
</table>
Consider Test 1 that has a laptop powered by an Intel Core -i3 Processor and a BBB communicating via a broker that is running on an Android mobile phone (LG Nexus 4). The test results here are similar to Test 4 where the only difference is that the laptop is replaced by another BBB device. This implies that the relatively high communication latency is not due to the processing power constraints on the client platforms but rather due to the bottlenecks in the broker’s performance. For Tests 2 and 3, we have lower communication latency as the broker is actually running on the laptop which leads to superior performance. It should also be noted that the broker implementations could also be considerably different since the one running on laptops is the open source Mosquitto Broker whereas the one running on the Android device is an ‘Android App’ called MQTT Broker Pro. Test 2 gives the lowest latency, as expected, because one of the clients and the broker share the same powerful platform of the laptop. Test 3 reaffirms our observation that even with both the devices being BBB boards, the communication latencies are low which implies that the performance of the broker holds the key to improving network latencies.

3.4.2 Test B: Network of Three Devices

In this test, we added a ‘Listener’ device to the network. This ‘Listener’ device is representative of any device that could be present on the network that wishes to snoop on the data transactions occurring on the network in order to capture the states of the devices at different time instances. Figure 3.8 shows the setup for the test case whereas Table 3.2 lists the test results. For this test, the aim was to assess the impact on communication latency due to

Table 3.2 Test results for communication latency between three clients representative of a DESD, SST and a Data Concentrator

<table>
<thead>
<tr>
<th>Client A</th>
<th>Client B / Broker</th>
<th>Client C</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBB ARMv7 Processor 300 MHz Debian 7.8</td>
<td>Intel Core i3 - 2370M 2.4 GHz Ubuntu LTS 14.04</td>
<td>BBB ARMv7 Processor 300 MHz Debian 7.8</td>
</tr>
<tr>
<td>3 Bytes(ms)</td>
<td>5.777</td>
<td>8.876</td>
</tr>
<tr>
<td>10 Bytes(ms)</td>
<td>11.555</td>
<td>14.997</td>
</tr>
</tbody>
</table>
Figure 3.8 Test setup for a three device network
the presence of a third device. As such, we opted for the best client-broker combination from Test A above. The results are comparable to those of Test Case 2 in Test A above. Firstly, the latency from client A to client B is the lowest because in both cases one of the clients and Broker share the same physical platform. Secondly, the trend is of an increase in latency as the broker is loaded to manage 3 clients in this case.

The tests above show that a powerful Broker needs to be in place to form a low latency, scalable communication network. Several commercial implementations of broker are available that can be evaluated for end usage. [45] provides a comprehensive list of open source and enterprise MQTT brokers with a comparison of their prominent features.

Latencies in the order of tens of milliseconds imply that fault handling at hardware level is best managed by the DSPs due to their real time performance. MQTT based communication can be used for networking of devices within the Energy Cell as the time steps for IEM algorithms shall be in the order of seconds.
Chapter 4

Applications Developed for DESD

Considering the different design choices made with regard to the embedded control and communication network for the DESD, we seek to demonstrate their effectiveness in supporting various FREEDM use cases for the same. We describe below the various applications that we have developed with the DESD and how they contribute towards easy and efficient integration of the DESD with the rest of FREEDM devices.

4.1 State of Charge Estimation

For any external device seeking to control the DESD as a part of any of the intelligent power or energy management algorithms, it needs to know the instantaneous energy capacity of the DESD which is a function of the SoCs of each of it’s batteries. This information is a vital input to any of the high level algorithms operating at the scale of a FREEDM Energy Cell as it directly determines, along with other system inputs, whether the DESD should be charged or discharged (and by what amount) at that instant. Thus to get a reasonably accurate estimate of the energy capacity of the DESD in real time, we must have have an accurate SoC estimation algorithm in place. This not only enables us to make intelligent decisions to operate the DESD but also ensures safe operation of the batteries by preventing situations such as over charge / deep discharge, over heating etc.
Our approach to calculate the energy capacity of the system is as below:

1. With reference to Figure 2.4, we start with the smallest observable unit of the battery being used in the DESD i.e. the cell modules that form a series string to make up a battery. Using the BMS system described in Section 2.1.2, we periodically collect the readings of cell module voltage and the average current that flows through that cell module for all the batteries. Modbus communication protocol works over the RF network formed by XBees to transfer all of this raw data from the BMS DSP to the BBB.

2. The BBB runs an SoC estimation algorithm at a time step of 1 second on each of these cell modules to come up with a real time estimate of SoC for all of them.

3. We average the SoCs for all the cell modules that belong to the same battery and multiply it by the maximum ampere-hour (Ah) capacity of a fully charged battery, $Ah_{battmax}$. $Ah_{battmax}$ is calculated offline based on a complete charge-discharge cycle test. While a theoretical $Ah_{battmax}$ is easy to calculate since the nominal capacities of all the cells is known from their datasheet [21], however, its not reliable as those numbers hold true for a new cell. The batteries being used in this DESD are actually reclaimed for secondary usage and hence it is rather required to experimentally identify their maximum capacities in the present state offline.

4. We add all the cell module voltages of the battery to arrive at the battery terminal voltage. This is multiplied by the quantity calculated in Step 2 above to arrive at the Watt-hour (Wh) capacity of the battery. The Wh capacities of all the batteries in the system are added to calculate the energy capacity of the DESD as a whole, at that time step. This process is shown in Figure 4.1.

Considering the above, we first discuss the process for determining $Ah_{battmax}$ and then detail the SoC estimation algorithm implemented for cell modules.
4.1.1 Determining the $\text{Ah}_{\text{battmax}}$

To determine the actual maximum capacity of the battery, we begin with a fully discharged battery and then charge it at a constant steady rate until the voltage across its cell modules’ terminal rises sharply to 3.6 V, which is equivalent to more than 95 percent of its's [21]. Based on the logged charge characteristics and the rate of charging current, it is easy to identify the Ah capacity of each cell module. Integration of current against time is then representative of the Ah capacity of the cell module. A representative characteristic is shown in Figure 4.2. For this plot, the Ah capacity after integration is equivalent to 19.9 Ah which is about 20 percent lesser than the theoretical capacity one could calculate from the datasheet of the cell modules.

4.1.2 SoC Estimation Algorithm

The SoC estimation algorithm implemented here is a simplified version of the one proposed in [46]. Instead of an adaptive online algorithm, in our implementation we determine the system constants offline based on data acquired through pulsed discharge test and then assume them to remain constant throughout the lifetime of the DESD and for all of its batteries. While this simplicity of implementation comes at the cost of accuracy offered by the adaptive approach, it is a reasonable trade off considering that the SoC algorithm is actually run for a total of 64 cell modules at every time step (8 cell modules in each of 8 batteries) and therefore processing
bandwidth is also a constraint. Moreover, this application doesn’t require an accuracy as high as the one demanded by other applications such as electric vehicles or plug in hybrid vehicles and hence such an implementation makes for an acceptable first iteration.

While a complete description of all the steps involved in parameter identification and tuning the gains for SoC observer is out of the scope of this Thesis, [46], [47] and [48] serve as useful references. We instead present below the details of the SoC estimation algorithm from an implementation perspective.

Model of a Battery Cell Module

We refer to the battery model used by Rahimi-Eichi and Chow [46] as shown in Figure 4.3. It consists of the following:

1. A large capacitor $Q_R$ to denote internal storage capacity of the battery.

2. A small internal resistance $R_O$ to represent the inter phase resistance involved in the
electrolyte based chemical charge/discharge process.

3. An RC circuit that represents the slow convergence of the battery Voc to its equilibrium point after hours of relaxation following charging/discharging [49].

4. A controlled voltage source to represent non-linear relationship between $V_{oc}$ and SoC.

In order to map this model to the cell modules of the batteries used in the DESD, we first had to identify the Voc- SoC characteristics of the cell modules. We subjected one such battery to a pulsed discharge test with constant load after it was fully charged using the Constant Current Constant Voltage (CCCV) technique. The discharge rate was about 12.5 A which corresponds to the nominal C/2 rate for the battery (where ‘C’ is the nominal maximum Ah capacity of the battery) and the discharge pattern was 10 minutes of discharge followed by 10 minutes of rest until the battery drained out completely. Based on the data collected from this one time offline test, a Voc - SoC curve is derived for the cell modules based on the method described in [50]. A representative plot for the cell modules as shown in Figure 4.4 was thus derived for the system. Given the characteristics in Figure 4.4, the $V_{oc}$ - SoC curve can be mapped with a sliding line with a varying slope, $b_1$, and $V_{oc}$ intersection, $b_0$.

$$V_{oc} = f(SoC) = b_1 SoC + b_0$$ (4.1)
Figure 4.4 The experimental curve for Voc - SoC

State Space Model

Based on the physical model described in Figure 4.3, the following state-space equations for the model can be derived.

\[
\begin{bmatrix}
SoC \\
V_{RC}
\end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & -\frac{1}{RC} \end{bmatrix} \begin{bmatrix} SoC \\
V_{RC}
\end{bmatrix} + \begin{bmatrix} \frac{1}{QR} \\ \frac{1}{C} \end{bmatrix} i_L
\]

\[v_T = b_1 \begin{bmatrix} SoC \\
V_{RC}
\end{bmatrix} + R_0 i_L + b_0 \]

Discretization and Parameter Identification

Considering Eq. 4.2, the terminal current \((i_L)\) and voltage \((v_T)\) are the only two quantities accessible for measurement. As such, rest of the parameters are identified using system parameter identification methods and state estimation techniques.

To begin with, the system transfer function is derived from Eq. 4.2. The moving window Least Squares method is used to identify battery parameters \((b_0, R, C, R_0, b_1, QR)\) from the data collected with the pulsed discharge test described in Section 4.1.2. These parameters are then used to complete the discrete model of the system by discretizing the system transfer.
function (Eq. 4.3) for a time step of 1 second.

\[
\frac{Y(s) - b_0}{U(s)} = \frac{R_0 s^2 + \left( \frac{b_1}{Q_R} + \frac{1}{C} + \frac{R_0}{R_C} \right) s + \frac{b_1}{R_C Q_R}}{s(s + \frac{1}{R_C})}
\]  

(4.3)

**Observer design for SoC**

Given the state-space model of Eq. 4.2, an observer is designed to estimate the SoC which is one of the states of the model. The observer can be designed as a system with Eq. 4.4.

\[
\begin{align*}
\dot{x} &= A \dot{x} + Bu + L(y - \hat{y}) \\
\hat{y} &= C \hat{x} + Du + b_0
\end{align*}
\]  

(4.4)

where

\[
\begin{align*}
x &= \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, x_1 &= SoC, x_2 &= V_{RC} \\
A &= \begin{bmatrix} 0 & 0 \\ 0 & -\frac{1}{R_C} \end{bmatrix}, B = \begin{bmatrix} \frac{1}{Q_R} \\ \frac{1}{C} \end{bmatrix}, C = \begin{bmatrix} b_1 & 1 \end{bmatrix} \\
D &= R_0, u = i_L, y = v_T
\end{align*}
\]  

(4.5)

and L is the observer gain vector. Linear Quadratic (LQ) is used to design the optimal observer.

**Run Time Execution**

All the above steps are completed offline. Once all the system parameters and also the observer gain are identified, real time values of \(v_T\) and \(i_L\) are all that’s needed for online SoC estimation. We begin with an initial reasonable estimate of SoC at the beginning of program execution and then feed real time values to the observer to update this estimate. Thus effectively, at every time step, a new estimate of SoC is calculated based on the previous estimate of SoC and the sensed terminal voltage and current that is used to estimate \(V_{oc}\). The time for convergence depends upon the difference between actual SoC and the initial estimate.
Figure 4.5 SoC, Charging current and Voltage for a cell module approaching full charged state. The intermittent fall in current readings can be attributed to measurement noise.
4.1.3 Validation

A useful indicator to validate the performance of SoC estimation algorithm could be its behavior when the cell module is approaching a fully charged condition. In this state, it’s terminal voltage rises sharply and if it’s SoC at this point approaches unity, then that could be considered an indication of it’s accuracy as well. Figure 4.5 shows how the SoC estimate closely tracks the charging of the cell module and approaches unit as soon as the terminal voltage of the cell module shoots up.

Thanks to Dr. Habiballah Rahimi Eichi for helping us with extracting the battery parameters and observer gains for this algorithm from raw data for pulsed discharge test.

4.2 State of Charge Balancing Algorithm

As described in Section 2.1, each of the H-Bridge circuits that constitutes the series string of inverters in this DESD is connected to two Li-ion batteries in parallel. Over a period of sustained usage, the batteries connected to different inverter modules might have different SoCs. In fact, since the DESD is designed to support batteries of possibly different chemistries and can even use reclaimed batteries from traction applications for secondary usage, it’s only natural that their respective SoCs might vary significantly by default. Such a variation in SoCs, if left unchecked, could lead to over charge or over discharge of some battery cells. This could lead to issues such as battery degradation, over heating and eventually under-utilization of the batteries [51].

In this section we seek to exploit the modular control scheme of the DESD to balance the average SoCs of each of the battery packs connected to the inverters. The objective is to make the average SoCs of all the battery packs converge to a common value, irrespective of whether the DESD is under a charge or a discharge operation. The central idea of our scheme is that given a total power command to the DESD, it can be distributed amongst the inverter modules in
correlation with their respective SoCs (instead of just an equal distribution) so as to eventually make all modules’ batteries converge on an SoC value. This algorithm is implemented on the BBB in a closed loop form by periodically redistributing the power references for each of the inverter blocks based on the instantaneous average SoC values of the battery packs connected to those inverter modules. While the algorithm is based on the scheme proposed by Huang and Abu Qahouq [52] for a series string of DC -DC converters, our implementation is for the series string of AC-DC converters and differs in the following ways:

1. Since we can only control power flow for each inverter i.e. for the two batteries connected in parallel to that inverter, we use the average of the SoCs of all the cell modules in those two batteries as representative of the SoC corresponding to that particular inverter. We can call this number as the battery pack SoC, \( \text{SoC}_{BP} \).

2. Since the battery pack SoCs might vary widely at run time, we would not want the corresponding power distribution amongst inverter modules to be disproportionate beyond a range which could cause the modulation index of any of the inverter modules to be beyond or below a certain limit. This would lead to distortion in the AC current waveform that is controlled by the DESD. This is prevented by setting maximum and minimum limits for the power quantum assigned to each inverter module based on the maximum and minimum modulation indices defined at system level for all the inverters. The modulation index \( m \) is defined as:

\[
m = \frac{V_{ac-pk}}{V_{dc}} = \frac{V_{ac-pk}}{V_{batt}}
\]

(4.6)

where

\[
V_{ac-pk} = \frac{P_{ac}}{I_{ac}} \sqrt{2}
\]

(4.7)

Based on Eq. 4.6 and Eq. 4.7, the maximum and minimum power limits for a given inverter
module can be defined as:

\[
\begin{align*}
P_{ac\text{-}\text{max}} &= \frac{m_{\text{max}}V_{\text{batt}}I_{ac}}{\sqrt{2}} \\
P_{ac\text{-}\text{min}} &= \frac{m_{\text{min}}V_{\text{batt}}I_{ac}}{\sqrt{2}}
\end{align*}
\] (4.8)

If for any given SoC variance and corresponding power command the distributed power commands are seen to breach these maximum or minimum limits, then the SoC balancing algorithm is reiterated with a slightly lower gain \(K_{\text{imbalance}}\) until the power distribution commands for all the modules fall within the valid range. This online tuning of \(K_{\text{imbalance}}\), which essentially serves as a proportional gain to the feedback loop, helps faster convergence of SoC while ensuring system safety.

3. The SoC balancing algorithm is run to distribute power commands amongst the inverter modules every one minute or when a new power command is received by the BBB board, whichever is earlier. The update rate of one minute has been experimentally verified to provide a good trade off between the time for SoC convergence, computational overhead and communication traffic over the XBee wireless network.

Figure 4.6 provides a block diagram of the closed loop control scheme for balancing battery pack SoCs. For any given time step, the battery pack SoC \(SoC_{BPi}\) for each of the four battery packs is calculated, based on which \(SoC_{avg}\) is determined. A \(\Delta P_{BPi}\) is calculated for each of the battery packs which is proportional to the difference between that battery pack’s SoC to \(SoC_{avg}\) and also the average power \(P_{avg}\) distribution for a given total power command \(P_{TOTAL}\). This \(\Delta P_{BPi}\) is then added to \(P_{avg}\) to arrive at the net power dispatch command for each of the battery packs. Such a scheme ensures that even though the distribution of power commands to each of the inverter modules is uneven, the sum total of the power generated/absorbed by the DESD is still the same as the received \(P_{TOTAL}\) command. At this point, if for any of the battery packs it’s power dispatch command is found to violate the limits set by Eq. 4.8, then a new \(\Delta P_{BPi}\) is calculated for all the battery packs again with a slightly lesser \(K_{\text{imbalance}}\) until a solution that respects the limits set by Eq. 4.8 is arrived for all the battery packs for the same value of
Figure 4.6 Block Diagram for SoC Balancing Algorithm

- $K_{CD} = 1$ if Discharge
  - $-1$ if Charge

- $SoC_{avg} = \sum_{i=1}^{4} \frac{SoC_{BPi}}{4}$

- $P_{avg} = \frac{P_{Total}}{4}$
Whether the net power dispatch command should be directly or inversely correlated to the \( SoC_{BP_i} \) is dependent on whether the power transfer operation under consideration is charging or discharging. The constant \( K_{C/D} \) is used in the algorithm to arrive at the appropriate correlation in this regard. Also note, that while \( K_{INITIAL} \) just serves to initialize \( K_{imbalance} \) for every time step, but by making it a function of the maximum variance of battery pack SoCs for that time step, faster convergence can be achieved.

Figure 4.7 provides the experimental results for convergence of battery packs SoCs during a discharge operation. It is noteworthy that the battery pack SoCs do not necessarily converge in the order of their relative SoCs. This is because the batteries are in dissimilar State of Health (SoH) and degradation as well, due to which their relative characteristics of discharge differ considerably. However the algorithm ensures that all of them eventually do converge.

![Convergence of SoCs of Battery Packs during Discharge](image)

Figure 4.7 Convergence of Battery Pack SoCs during a discharge operation
4.3 Integration with a Home Energy Management System

In order to demonstrate the interconnectivity of the DESD with a Home Energy Management System (HEMS), we connected them via the MQTT network. The HEMS could be serving various objective functions such as: one, to regulate a power cap for the amount of power drawn from the grid by using the DESD to meet any extra load demand or two, to use the DESD based on time of usage (TOU) utility rates so as to charge the DESD when the cost of electricity is low during the day and then later discharge it at peak demand times (when the cost of electricity is typically high). To perform these operations effectively, HEMS periodically needs an updated value of the residual energy inside the DESD and also a mechanism to dispatch power commands to the DESD in order to charge or discharge it at a particular power level. The HEMS’ core algorithm works on the Matlab platform on a PC and so we also had an MQTT Client running on the same PC. For the sake of simplicity, the HEMS and MQTT Client exchanged data via a shared file interface. At every time step, HEMS reads the residual energy in the DESD, run it’s algorithm to achieve optimal power flow in to the house from sources such as the grid and a simulated PV panel, and comes up with a dispatch power command for the DESD. The MQTT Client then forwards this command to the DESD and reads back an updated value of the residual energy capacity from the DESD. The HEMS reads this value of residual capacity in the next time step. Figure 4.8 shows the integration of DESD with the HEMS.

Thanks to Jiahong Yan for helping with the integration of DESD with the HEMS system developed by him.
Figure 4.8 Integration of DESD with the HEMS System
Chapter 5

Performance Benchmarking Tests for the DESD

[6] has recently published a standardized and exhaustive set of tests for measuring and expressing the performance benchmarks of energy storage systems. In this chapter, we present the results for some of the tests mentioned in it. With this activity, we aim to provide certain performance benchmarks for the DESD.

5.1 Reference Signal Tracking

This test is required for validating the usage of an energy storage device for applications such as frequency regulation and in operation with islanded microgrids. It involves measurement of the actual generated power \( P_{ess} \) against the reference command \( P_{signal} \) provided to the device. For a representative test, we give the device a power command in steps of 100 W covering a range of 600 W for both charging and discharging. The results are tabulated in Table 5.1. Error is defined as

\[
Error = \left| \frac{(P_{signal} - P_{ess})}{P_{ess}} \right|
\]  

(5.1)
Table 5.1 Reference Tracking Performance of DESD

<table>
<thead>
<tr>
<th>Psignal</th>
<th>Pess</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>-600</td>
<td>-596</td>
<td>0.01</td>
</tr>
<tr>
<td>-500</td>
<td>-500</td>
<td>0.00</td>
</tr>
<tr>
<td>-400</td>
<td>-400</td>
<td>0.00</td>
</tr>
<tr>
<td>-300</td>
<td>-300</td>
<td>0.00</td>
</tr>
<tr>
<td>-200</td>
<td>-199</td>
<td>0.01</td>
</tr>
<tr>
<td>-100</td>
<td>-102</td>
<td>0.02</td>
</tr>
<tr>
<td>100</td>
<td>95</td>
<td>0.05</td>
</tr>
<tr>
<td>200</td>
<td>197</td>
<td>0.01</td>
</tr>
<tr>
<td>300</td>
<td>294</td>
<td>0.02</td>
</tr>
<tr>
<td>400</td>
<td>402</td>
<td>0.01</td>
</tr>
<tr>
<td>500</td>
<td>499</td>
<td>0.00</td>
</tr>
<tr>
<td>600</td>
<td>602</td>
<td>0.00</td>
</tr>
</tbody>
</table>

If the error is observed to be less than 0.02, then the storage device is considered to track the reference. As can be seen from Table 5.1, the error for DESD is within limits for all but one data point.

5.2 Response Time and Ramp Rate

Response time of the storage device is a parameter that indicates the amount of time it requires to ramp up from no discharge to full rated discharge power. Figure 5.1 explains how the response time is to be be measured. Time instant T1 defines the boundary of the storage device when it receives the rated power command from an external entity. Time instance T2 is for the instant when the ESS reaches it’s rated power limit. The difference between T2 and T1 is the response time.

We performed the above test with the DESD where it was given a command to discharge 1000 W while it was operating at 0 W. The result can be seen in Figure 5.2 The rising edge of yellow curve on Channel 1 marks the instance when the DESD received the reference power command from an external entity. The pink curve on Channel 3 shows the ramp up of the discharge current against time. The instance where the ramp reaches within 2 percent of 1000
is marked. The response time is thus observed to be 60 seconds. Based on its definition, the effective ramp rate is then given as:

\[
\text{Ramp Rate} = \frac{\text{Rated Power}}{\text{Response Time}} = \frac{1000}{60} = 16.66 \text{ Watts/sec} \quad (5.2)
\]

Figure 5.1 Response Time and Ramp Rate Calculation. Adapted from [6]

Figure 5.2 Response Time observed for the DESD
Chapter 6

Conclusion

6.1 Accomplishments

In this Thesis we explored the role of the DESD as a distributed energy resource within the FREEDM Energy Cell and the different possible use cases for such a device from an architectural perspective. This allowed us to formulate the requirements for such a device in terms of it’s interface, communication and controller system design.

We constructed a scalable and modular Battery Management System and integrated it with the rest of the power electronics because the SoC of the DESD’s batteries is a vital parameter for any algorithm that governs the DESD. In the process we came up with a two tiered controller architecture consisting of DSPs and an embedded single board computer like the Beagle Bone Black. Such an approach leads to a modular design where the operation of underlying power electronics is decoupled from higher level communication and application development tasks. We evaluated several embedded communication protocols and chose Modbus using UART as the best option in terms of simplicity, reliability and scalability. XBee based RF radios were used at the hardware layer which leads to almost no wiring to handle internal communication within the device.
The networking needs of devices that form the FREEDM Energy Cell were also analyzed. After exploring the design space based on the existing open messaging protocols, we chose MQTT as a light weight TCP/IP based networking protocol for these distributed devices to form a local network within the Energy Cell. A standardized data model using JSON as the underlying data interface format is proposed and implemented. The implementation supports a scheme where multiple applications can concurrently access and control the devices on the network without any concurrency issues.

Applications such SoC Estimation, SoC Balancing and Interfacing of the DESD with a Smart Home are demonstrated to validate the framework. Test results for

6.2 Future Work

To further ease the interfacing of FREEDM devices, the next step would be to integrate all the existing work within the DGI architecture. This would also provide a consistent and vertically integrated platform for future development. The concept of Device Profile could be refined to standardize the nomenclature and quantification of common data points. The usage of TLS/SSL protocol for network layer security and further authentication using username/password approach for MQTT also needs to be explored to make the solution resilient and robust.

For the DESD, a more accurate approach to estimating SoC should be considered using online parameter identification methods. This would allow for SoC estimation to consider effects of temperature, ageing, degradation etc. Further automated test platforms can be developed for the DESD to characterize its performance parameters such as round trip energy efficiency, reference tracking etc. Additional enhancements to the system can be made to raise it’s power and energy capacity.
BIBLIOGRAPHY


67


Appendix A

Serial Communication Commands

This Appendix documents the commands to control the power electronics as well as the BMS system. The system consists of four DSPs to control the power electronics and a fifth DSP to gather raw data for BMS. Only the BMS DSP follows MODBUS protocol whereas the rest of the DSPs follow a system specific command structure as stated below. Since all the commands are broadcast by the Master Xbee interfaced to the BBB to the Slave Xbees connected to the DSPs, it is important to prevent the Slave DSPs controlling the power electronics to not accept or respond to the Modbus commands meant for the BMS DSP. This is ensured by having a dedicated header and footer for all the commands meant for power DSPs. The header is chosen as '0x5555 0xAAAA' and the footer is chosen as '0xFFFF'. Since the Modbus commands being used in the system won’t have these same sequence of commands for any of the messages (verified experimentally), it is guaranteed that there would be no misinterpretation of commands between the DSPs. As such, Modbus as well as system specific command structure can coexist without any interference. Note that such a differentiated command structure is being followed due to legacy reasons and in subsequent design iterations all the commands should be completely Modbus based.

Note that all the frames that are not Modbus based are all dispatched in Hexadecimal (Hex) format unless otherwise state explicitly.
The BMS DSP is powered independently from the rest of the system in current implementation. So it can always remain ON and therefore control the power supply for the rest of the hardware. Thus, the relay to control the power supply for rest of the hardware is in turn controlled by the BMS DSP.

A.1 DESD Turn ON Sequence

Table A.1 lists the commands to be issued by the BBB to the DSPs to turn ON the DESD.

Table A.1 DESD Turn ON Sequence

<table>
<thead>
<tr>
<th>Operation</th>
<th>Hex Code or Modbus Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Supply Relay ON</td>
<td>Modbus: Holding Reg. Addr = 0, Data = 1</td>
</tr>
<tr>
<td>Module 1 Battery Relay ON</td>
<td>5555 AAAA A011 FFFF</td>
</tr>
<tr>
<td>Module 2 Battery Relay ON</td>
<td>5555 AAAA A021 FFFF</td>
</tr>
<tr>
<td>Module 3 Battery Relay ON</td>
<td>5555 AAAA A031 FFFF</td>
</tr>
<tr>
<td>Module 4 Battery Relay ON</td>
<td>5555 AAAA A041 FFFF</td>
</tr>
<tr>
<td>Grid Relay ON</td>
<td>5555 AAAA A001 FFFF</td>
</tr>
<tr>
<td>Module 4 Turn ON</td>
<td>5555 AAAA 9001 FFFF</td>
</tr>
<tr>
<td>Module 3 Turn ON</td>
<td>5555 AAAA 7001 FFFF</td>
</tr>
<tr>
<td>Module 2 Turn ON</td>
<td>5555 AAAA 5001 FFFF</td>
</tr>
<tr>
<td>Module 1 Turn ON</td>
<td>5555 AAAA 2001 FFFF</td>
</tr>
</tbody>
</table>

A.2 DESD Turn OFF Sequence

Table A.2 lists the commands to be issued by the BBB to the DSPs to turn OFF the DESD.

A.3 Power Dispatch Commands

Once the DESD is turned ON, it’s operation is controlled by issuing power dispatch commands to it’s DSPs. Considering the modular control scheme followed by the system, all the power DSPs need to be communicated their respective power reference commands and also the
### Table A.2 DESD Turn OFF Sequence

<table>
<thead>
<tr>
<th>Operation</th>
<th>Hex Code or Modbus Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module 1 Turn OFF</td>
<td>5555 AAAA 2000 FFFF</td>
</tr>
<tr>
<td>Module 2 Turn OFF</td>
<td>5555 AAAA 5000 FFFF</td>
</tr>
<tr>
<td>Module 3 Turn OFF</td>
<td>5555 AAAA 7000 FFFF</td>
</tr>
<tr>
<td>Module 4 Turn OFF</td>
<td>5555 AAAA 9000 FFFF</td>
</tr>
<tr>
<td>Grid Relay OFF</td>
<td>5555 AAAA A000 FFFF</td>
</tr>
<tr>
<td>Module 1 Battery Relay OFF</td>
<td>5555 AAAA A010 FFFF</td>
</tr>
<tr>
<td>Module 2 Battery Relay OFF</td>
<td>5555 AAAA A020 FFFF</td>
</tr>
<tr>
<td>Module 3 Battery Relay OFF</td>
<td>5555 AAAA A030 FFFF</td>
</tr>
<tr>
<td>Module 4 Battery Relay OFF</td>
<td>5555 AAAA A040 FFFF</td>
</tr>
<tr>
<td>Control Supply Relay OFF</td>
<td>Modbus : Holding Reg. Addr = 0, Data = 0</td>
</tr>
</tbody>
</table>

The message frame for dispatching a power command is slightly more involved as it consists of communicating the Total Power to all the modules and then each module also needs to know its own contribution to that total power. For the sake of simplicity, here we assume that all modules contribute equally. The general format of the message frame is:

```
Header TotalPwr Header Mod1Pwr Mod2Pwr Mod3Pwr Mod4Pwr Footer
```

where,

- **Header** = 5555 AAAA
- **TotalPower** = Total power to be dispatched + 0x0000
- **Mod1Pwr** = Power to be dispatched by Module 1 + 0x1000
- **Mod2Pwr** = Power to be dispatched by Module 2 + 0x4000

Total power command. Also the direction of power flow needs to be communicated to control charge/discharge operation.

Table A.3 lists the commands for Charge/Discharge Operation.

### Table A.3 Charge / Discharge Direction Command for DESD

<table>
<thead>
<tr>
<th>Operation</th>
<th>Hex Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Charge</td>
<td>5555 AAAA D000 FFFF</td>
</tr>
<tr>
<td>Discharge</td>
<td>5555 AAAA D001 FFFF</td>
</tr>
</tbody>
</table>
Mod3Pwr = Power to be dispatched by Module 3 + 0x6000

Mod4Pwr = Power to be dispatched by Module 4 + 0x8000

Footer = FFFF

Some sample frames for typical power commands are shown in Table A.4.

<table>
<thead>
<tr>
<th>Dispatch Power</th>
<th>Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>5555 AAAA 0064 5555 AAAA 1019 4019 6019 8019 FFFF</td>
</tr>
<tr>
<td>200</td>
<td>5555 AAAA 00C8 5555 AAAA 1032 4032 6032 8032 FFFF</td>
</tr>
<tr>
<td>400</td>
<td>5555 AAAA 0190 5555 AAAA 1064 4064 6064 8064 FFFF</td>
</tr>
</tbody>
</table>

A.4 System Parameter Polling Commands

The below table gives polling commands for system parameters. AC Voltage, AC Current, Time Period (inverse of AC Frequency) and power flow Direction can be polled in real time from the system.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Polling Command</th>
<th>Offset</th>
<th>Scaling (Decimal)</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC Voltage</td>
<td>B001 FFFF</td>
<td>B000</td>
<td>50</td>
<td>V</td>
</tr>
<tr>
<td>AC Current</td>
<td>B002 FFFF</td>
<td>C000</td>
<td>100</td>
<td>A</td>
</tr>
<tr>
<td>Time Period</td>
<td>B003 FFFF</td>
<td>F000</td>
<td>100</td>
<td>ms</td>
</tr>
<tr>
<td>Direction</td>
<td>B004 FFFF</td>
<td>D000</td>
<td>1</td>
<td>-</td>
</tr>
</tbody>
</table>

From the response received, the physical value can be determined using:

\[
Physical Value = (Response - Offset)/Scaling
\]

For example, if the response received for polling AC Voltage is 0xB992, then the physical
value is calculated as:

\[
AC\ Voltage\ (Physical) = (B992 - B000)_{16}/50_{10} = 992_{16}/50_{10} = 2450_{10}/50_{10} = 49\ V
\]

A response of '1' for Direction should be interpreted as 'Charging' and '0' as 'Discharging'

A.5 Modbus Polling for BMS

The system consists a total of 8 batteries. Each of the batteries generates 11 data points: 8 cell module voltages, 2 temperature (unused, dummy values for now) and 1 current. The range of data points for each battery is described in Table A.6 and the position for a specific data point within the range for Battery 1 is shown in Table A.7. All the batteries follow similar positioning of data points.

<table>
<thead>
<tr>
<th>Modbus Input Reg. Address</th>
<th>Battery</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0....10]</td>
<td>Battery 1</td>
</tr>
<tr>
<td>[11....21]</td>
<td>Battery 2</td>
</tr>
<tr>
<td>[22....32]</td>
<td>Battery 3</td>
</tr>
<tr>
<td>[33....43]</td>
<td>Battery 4</td>
</tr>
<tr>
<td>[44....54]</td>
<td>Battery 5</td>
</tr>
<tr>
<td>[55....65]</td>
<td>Battery 6</td>
</tr>
<tr>
<td>[66....76]</td>
<td>Battery 7</td>
</tr>
<tr>
<td>[77....87]</td>
<td>Battery 8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Modbus Input Reg. Address</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0....7]</td>
<td>Cell Module 1...Cell Module 8 (Voltage)</td>
</tr>
<tr>
<td>[8, 9]</td>
<td>Temperature (Unused)</td>
</tr>
<tr>
<td>[10]</td>
<td>Current</td>
</tr>
</tbody>
</table>
Appendix B

Xbee Configuration for Master - Slave Operation

This Appendix describes the configuration of Xbee modules for different use cases. We use the Xbee module XB24-AWI-001 for our project as it is 3.3 V compatible, works with the UART port of microcontrollers and has reasonable power consumption (50 mA). One can as well use XBP24-AWI-001 which has the exact same features as XB24, but a higher range and power consumption as well.

While the IEEE 802.15.4 protocol, on which the Xbees are based, does not specify any Master-Slave kind of configuration for the Xbees to operate, but since we are using the Xbees for MODBUS which explicitly defines a Master - Slave architecture, we describe below a possible Xbee configuration to mimic a Master - Slave architecture.

Note that all the Xbees sharing the same network need to be assigned the same PAN ID that’s unique to that network. The default PAN ID for Xbees out of the box is 0x3332. This should NEVER be used as is, and should be changed to some other 4 digit number. In the example below, we shall be using 0x10 as the PAN ID for our network.

While there can be only one Master in the system, Slaves can be of two types:

1. Slave with both Tx and Rx functionality
2. Slave with only Rx functionality.

We shall be changing only the following parameters to configure the Xbees:

1. MY ADDRESS (MY)
2. DESTINATION LOW (DL)
3. PAN ID (ID)
4. BAUD RATE (BR)

Thus, we have 3 possible configurations for any Xbee radio as described in sections below.

**B.1 Xbee Master Configuration**

Modbus Xbee Master needs to broadcast it’s messages to all the slaves. Each of the slaves can then filter the message at it’s end based on the Slave ID field of the Modbus message frame. To broadcast messages, Xbees allow a special destination address : 0xFFFF. We use this for the DL field of the Master. MY ADDRESS for the Master can be any unique number which shall be later used as the destination address for slaves.

1. MY ADDRESS (MY) = 0
2. DESTINATION LOW (DL) = FFFF
3. PAN ID (ID) = 10
4. BAUD RATE (BR) = 4 (Stands for 19200)

**B.2 Xbee Slave Configuration : Rx and Tx Enabled**

Since this type of Slave needs to talk only with the Master, it has to give Master’s MY ADDRESS as it’s own DESTINATION LOW ADDRESS.
1. MY ADDRESS (MY) = 1
2. DESTINATION LOW (DL) = 0
3. PAN ID (ID) = 10
4. BAUD RATE (BR) = 4 (Stands for 19200)

**B.3 Xbee Slave Configuration : Only Rx Enabled**

This kind of Slave only receives messages from the Master and doesn’t transmit anything back to the Master. As such it’s DESTINATION LOW address is configured to be any number that’s NOT been assigned as any of the Xbee device’s MY ADDRESS. Essentially, this kind of Xbee Slave is configured to transmit data to a device that doesn’t exist on the network.

1. MY ADDRESS (MY) = 2
2. DESTINATION LOW (DL) = 1234 (assuming there is no Xbee device on the network with MY ADDRESS as 1234)
3. PAN ID (ID) = 10
4. BAUD RATE (BR) = 4 (Stands for 19200)

Apart from the above fields, make sure that the below fields are configured as shown for all types of devices, irrespective of whether they serve as Master or Slaves of any kind.

1. CO-ORDINATOR ENABLE (CE) = 0
2. DESTINATION HIGH (DH) = 0
3. ASSOCIATION 1 (A1) = 0
4. ASSOCIATION 2 (A2) = 0
Note that the approach described here is just one of the several possible configurations to model a Master-Slave network and other approaches (like using $CE = 1$ with other appropriate settings) could also work.
Appendix C

High Level Flow Chart of Software on Beagle Bone Black

The BBB acts as an interface between the external world and the DSPs for the DESD. Considering this model, the application code on DESD is a multi-threaded software with one thread each for communicating with the DSPs (via Modbus/Serial link) and the external world (via MQTT/JSON file interface).

The file `thread_dsp_comm.py` hosts the thread that handles the communication with the DSPs while also including algorithms for SoC estimation and SoC Balancing. It fetches the raw data from the BMS DSPs, process it for SoC estimation, and then dispatches new reference commands to the power DSPs based on the total dispatch power command and the SoC balancing algorithm while taking into account the desired slew rate.

The file `thread_json.py` hosts the thread that performs read/write operations on the JSON file for the DESD. The JSON file is also concurrently handled by MQTT Client application which handles the underlying MQTT communication.
The two threads interface with each other using the concept of Queues, as is commonly done in multi threaded applications.

Figure C.1 explains this multi-threaded operation in detail.

Figure C.1 Flowchart for application software on BBB