AUTONOMOUS POWER MANAGEMENT OF
SERIES-CASCaded AND HYBRID
MICROGRIDS

A thesis submitted in fulfillment of the requirements
for the degree of Master of Engineering

by

Siji Das
B. Tech. (2010), Mahatma Gandhi University, Kerala, India.
M. Tech. (2013), Mahatma Gandhi University, Kerala, India.

Supervisor: Dr. Inam Ullah Nutkani
Associate Supervisor: Dr. Carlos Teixeira

School of Engineering
College of Science, Engineering, and Health
RMIT University

November 2019
Declaration

I certify that except where due acknowledgement has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program; any editorial work, paid or unpaid, carried out by a third party is acknowledged; and, ethics procedures and guidelines have been followed.

I acknowledge the support I have received for my research through the provision of an Australian Government Research Training Program Scholarship.

Siji Das
06/11/2019
Abstract

Microgrids with power electronics interfaced Distributed Generation units are gaining high popularity due to its zero emission characteristics. Control and co-ordination of these generation units are the most crucial factors that will determine the effective utilisation and flexibility of microgrids. Conventional microgrid structure with droop controlled parallel distributed generation units are being replaced by the series-cascaded structure due to its reduced conversion stages and inherent harmonic sharing capability.

This research work first aims to develop a microgrid architecture integrating dispatchable and non-dispatchable distributed generation units in a series-cascaded manner. Existing control strategies for cascaded microgrids focus on dispatchable type generation only. However, adequate power sharing and voltage regulation of a microgrid containing mixed dispatchable and non-dispatchable cascaded generation units demand new control approaches to achieve operational performance and reliability comparable to the conventional parallel-topology microgrid. To ensure maximum utilisation of non-dispatchable units a novel microgrid architecture formed by a dispatchable master unit followed by a set of non-dispatchable slave photovoltaic units in a series-cascaded manner is developed. A fully decentralised control scheme is proposed, which achieves autonomous power balancing and voltage regulation, ensures full utilisation of non-dispatchable generation units, and allows surplus power curtailment under light load conditions.

Further, this research work aims to extend the series topological arrangement to form a hybrid microgrid, where low voltage converters are cascaded as a string unit to achieve rated output voltage, and these strings are then paralleled to obtain higher
redundancy and power rating. The extension of the arrangement to a hybrid microgrid requires the development of new control strategies, since existing schemes cannot be applied in their original form. As of now hybrid microgrids are controlled using either distributed or centralised schemes to achieve accurate power sharing among the distributed generation units at the cost of complex communication infrastructure. Therefore, a new control scheme is proposed for the hybrid microgrid which aims to achieve accurate power sharing among the paralleled units while maintaining adequate synchronisation among the cascaded converters without any communication link.

Fundamental concepts as well as mathematical and simulation models of the existing and proposed control schemes are presented. All the proposed control strategies are validated through extensive simulation results and the series-cascaded microgrid control is validated through matching simulation and experimental results.
Acknowledgements

First, I would like to express my sincere gratitude to my supervisors Dr. Inam Ullah Nutkani and Dr. Carlos Teixeira for their valuable guidance and support throughout this work. As a beginner to the research filed, I may not be able to complete this work without their helping hands. I appreciate time and effort they dedicate to the completion of this work. I am especially thankful to Dr. Inam for his valuable time, critical feedback and moral support provided throughout this journey. I am extremely grateful to Dr. Carlos for the help and support provided from the beginning onwards. A big salute for his patience and skills to transcend mediocrity.

Next, I would like to thank my family for the support they provided over last two years. I wish to thank my mom for the great support and encouragement she has provided. A special thanks to my husband, Nijin John Thomas for being so supportive and understanding partner. Also, I would like to thank my little champ, Aldrich, my world is not complete without you.

Further, I would like to dedicate this thesis for the loving memory of my dad, Mr. Das Thomas.

Above all, I would like to thank almighty for blessing me with the best.
Publications

Several parts of the work and ideas presented in this thesis have been published by the author during this research. These publications are listed below.


Contents

Abstract .................................................................................................................................................. iii
Acknowledgements ................................................................................................................................. v
Publications .............................................................................................................................................. vi
Contents ............................................................................................................................................... vii
List of Figures ..................................................................................................................................... xi
List of Tables ......................................................................................................................................... xv
Glossary of Terms ................................................................................................................................. xvii
List of Symbols ...................................................................................................................................... xix
List of Subscripts .................................................................................................................................. xxi
Chapter 1 Introduction .......................................................................................................................... 1
  1.1. Motivation ........................................................................................................................................ 1
  1.2. Problem Statement .......................................................................................................................... 2
  1.3. Objectives ......................................................................................................................................... 4
  1.4. Identification of Original Contribution ........................................................................................... 6
  1.5. Thesis Structure ............................................................................................................................... 7
Chapter 2 Overview of Parallel, Series and Hybrid Microgrids ......................................................... 9
  2.1. Microgrid Concept .......................................................................................................................... 9
  2.2. Overview of Microgrid Topologies and Power Management Strategies .................................... 13
    2.2.1. Parallel Topology Microgrid .................................................................................................... 13
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.2. Series-Cascaded Topology Microgrid</td>
<td>22</td>
</tr>
<tr>
<td>2.2.3. Hybrid Topology Microgrid</td>
<td>27</td>
</tr>
<tr>
<td>2.3. Summary</td>
<td>28</td>
</tr>
<tr>
<td>Chapter 3 Series-Cascaded Microgrid with Non-Dispatchables</td>
<td>31</td>
</tr>
<tr>
<td>3.1. Proposed Series-cascaded Microgrids</td>
<td>32</td>
</tr>
<tr>
<td>3.2. Master-slave Control Strategy for Proposed Series-Cascaded Microgrids</td>
<td>33</td>
</tr>
<tr>
<td>3.2.1. Power factor versus frequency $(pf - \omega)$ droop control</td>
<td>34</td>
</tr>
<tr>
<td>3.2.2. Active Power versus frequency $(P - \omega)$ droop control</td>
<td>38</td>
</tr>
<tr>
<td>3.3. Simulation and Experimental Validation</td>
<td>40</td>
</tr>
<tr>
<td>3.3.1. Validation System</td>
<td>40</td>
</tr>
<tr>
<td>3.3.2. Simulation Results</td>
<td>41</td>
</tr>
<tr>
<td>3.4. Experimental Verification</td>
<td>53</td>
</tr>
<tr>
<td>3.5. Frequency Response Analysis</td>
<td>59</td>
</tr>
<tr>
<td>3.6. Summary</td>
<td>64</td>
</tr>
<tr>
<td>Chapter 4 Hybrid Microgrid</td>
<td>65</td>
</tr>
<tr>
<td>4.1. Hybrid Microgrid Architecture</td>
<td>65</td>
</tr>
<tr>
<td>4.2. Proposed Hybrid Microgrid Architecture</td>
<td>66</td>
</tr>
<tr>
<td>4.3. Master-Slave Control Strategy for Proposed Hybrid Microgrid</td>
<td>68</td>
</tr>
<tr>
<td>4.3.1. Master DG</td>
<td>68</td>
</tr>
<tr>
<td>4.3.2. Slave DG</td>
<td>70</td>
</tr>
<tr>
<td>4.4. Performance Validation</td>
<td>72</td>
</tr>
<tr>
<td>4.5. Summary</td>
<td>81</td>
</tr>
<tr>
<td>Chapter 5 Description of the Simulation Systems</td>
<td>82</td>
</tr>
<tr>
<td>5.1. PSIM</td>
<td>82</td>
</tr>
</tbody>
</table>
C.2. Background Routines – C Source File.................................................................108
C.3. Foreground Routines – Header File.................................................................116
C 4. Source File ......................................................................................................118
C.5. Grab Routine – Header File ..............................................................................154
References.............................................................................................................156
List of Figures

Figure 2.1: Exemplar microgrid model with power electronics interfaced DGs............11
Figure 2.2: Primary level control block diagram..........................................................12
Figure 2.3: Parallel topology microgrid structure............................................................14
Figure 2.4: Droop curve for parallel topology microgrids with resistive line (a) Active power droop (b) Reactive power droop.................................................................15
Figure 2.5: Inner controller block diagram....................................................................16
Figure 2.6: Simulation setup for parallel microgrid .........................................................17
Figure 2.7: Simulation results for parallel topology microgrid showing DGs and microgrid (a) Active power (b) Voltage (c) Reactive power (d) Frequency...........19
Figure 2.8: Simulation results for parallel topology microgrid showing (a) Active power droop (b) Reactive power droop.................................................................20
Figure 2.9: Series-cascaded microgrid architecture with low gain boost stage...........22
Figure 2.10: Effect of synchronisation in cascaded microgrid (a)Equal voltage sharing with synchronisation (b) Unequal voltage sharing without synchronisation (c) Equal power sharing with synchronisation (d) Unequal power sharing without synchronisation........................................................................................................24
Figure 2.11: Hybrid topology microgrid ........................................................................27
Figure 3.1: Series-cascaded microgrid with solar PV units........................................32
Figure 3.2: Series-cascaded microgrid with \( pf - \omega \) control, (a) Control block diagram and (b) \( pf - \omega \) droop curve for the master DG .................................................................34
Figure 3.3: Series-cascaded microgrid with \( pf - \omega \) control - Control block diagram for the salve DGs .........................................................................................................................36
Figure 3.4: Series-cascaded microgrid with $P-\omega$ control, (a) Control block diagram and (b) $P-\omega$ droop curve for the master DG ..........................................................39

Figure 3.5: Series-cascaded microgrid with $P-\omega$ control, (a) Control block diagram and (b) $Q-\omega$ droop curve for the slave DGs ..........................................................40

Figure 3.6: Series-cascaded microgrid (a) Simulation setup (b) \[ pf - \omega \] droop curve .41

Figure 3.7: Simulation results for series-cascaded microgrid Scenario A showing (a) microgrid frequency, and (b) Active power output (c) Voltage of the DGs and microgrid ..........................................................42

Figure 3.8: Simulation results for series-cascaded microgrid Scenario A showing DGs and microgrid (a) Power factor and (b) Reactive power output for synchronisation mode operation ..........................................................44

Figure 3.9: Phasor diagram for series-cascaded microgrid Scenario A (a) MPPT=0.15kW (b) MPPT=0.3kW (c) MPPT=0.45kW for synchronization mode operation .................................................................................44

Figure 3.10: Simulation results for series-cascaded microgrid Scenario A showing DGs and microgrid (a) Power factor and (b) Reactive power output for power factor dispatch mode operation .................................................................................46

Figure 3.11: Phasor diagram for series-cascaded microgrid Scenario A (a) MPPT=0.15kW (b) MPPT=0.3kW (c) MPPT=0.45kW for power factor dispatch mode operation .................................................................................46

Figure 3.12: Simulation results for series-cascaded microgrid Scenario A showing DGs and microgrid (a) Power factor (b) Reactive power output for unity power factor operation .................................................................................48

Figure 3.13: Phasor diagram for series-cascaded microgrid Scenario A (a) MPPT=0.15 kW (b) MPPT=0.3 kW (c) MPPT=0.45 kW for unity power factor operation .................................................................................48

Figure 3.14: Simulation results for series-cascaded microgrid scenario A showing DGs and microgrid (a) active power and (b) voltage output with actual irradiance profile .................................................................................49

Figure 3.15: Simulation results for series-cascaded microgrid Scenario B showing DGs and microgrid (a) active power and (b) reactive power output, and (c) microgrid frequency .................................................................................51

Figure 3.16: Simulation results for series-cascaded microgrid Scenario B showing DGs and microgrid (a) operating power factor (b) reactive power output .................................................................52

Figure 3.17: Simulation and experimental setup for series-cascaded microgrid.......53

Figure 3.18: Validation results showing active power output of the DGs ..........56
Figure 4.9: Simulation results for hybrid microgrid Scenario B, showing (a) microgrid frequency (b) String and microgrid active power, (c) microgrid voltage, and (d) String and microgrid reactive power.

Figure 5.1: Series-cascaded microgrid model power stage in PSIM.

Figure 5.2: Series-cascaded microgrid control structure implemented in PSIM.

Figure 5.3: Hybrid microgrid power stage in PSIM.

Figure 5.4: Hybrid microgrid control structure implemented in PSIM.

Figure 5.5: DG power stage in PSIM.

Figure 5.6: Single phase PWM modulator in PSIM.

Figure 5.7: Inner control loop (a) Current controller (b) Voltage controller.

Figure 5.8: Single phase PLL in PSIM.

Figure 5.9: Single phase power calculation in PSIM.

Figure 5.10: (a) Power factor regulator and (b) MPPT tracking in PSIM.

Figure 5.11: Simulation results for series-cascaded microgrid showing MPPT tracking.

Figure 6.1: Simplified schematic diagram of the RMIT-U01 power stage board.

Figure 6.2: Illustrative representation of the general event manager configuration.

Figure 6.3: Functional block diagram of the CPT-DA2810 controller card [67].

Figure 6.4: Functional block diagram of the CPT-E13 controller motherboard. [68].

Figure 6.5: RMIT-U01 Power stage board in series-cascaded arrangement.

Figure 6.6: Experimental setup.

Figure 6.7: Experimental results showing DGs and microgrid power factor sharing in synchronisation mode.
List of Tables

TABLE 1.1: PERFORMANCE MATRIX FOR PROPOSED SERIES-CASCaded MICROGRID ..... 4
TABLE 1.2: PERFORMANCE MATRIX FOR PROPOSED HYBRID MICROGRID ............... 6
TABLE 1.3: COMPARISON OF EXISTING AND PROPOSED SERIES-CASCaded MICROGRID .......................................................... 7
TABLE 1.4: COMPARISON OF EXISTING AND PROPOSED HYBRID MICROGRID .......... 8
TABLE 2.1: PERFORMANCE COMPARISON FOR DIFFERENT MICROGRID TOPOLOGIES ... 28
TABLE 2.2: SUMMARY OF REPORTED CONTROL STRATEGIES FOR DIFFERENT MICROGRID TOPOLOGIES .......................................................... 30
TABLE 3.1: SIMULATION SYSTEM PARAMETERS FOR SERIES-CASCaded MICROGRID . 50
TABLE 3.2: EXPERIMENTAL SYSTEM PARAMETERS FOR SERIES-CASCaded MICROGRID. .......................................................... 55
TABLE 4.1: PERFORMANCE VALIDATION PARAMETERS FOR HYBRID MICROGRID...... 73
# Glossary of Terms

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC</td>
<td>Alternating Current</td>
</tr>
<tr>
<td>CSI</td>
<td>Current Source Inverter</td>
</tr>
<tr>
<td>DC</td>
<td>Direct Current</td>
</tr>
<tr>
<td>DG</td>
<td>Distributed Generation</td>
</tr>
<tr>
<td>kVA</td>
<td>Kilo-Volt-Ampere</td>
</tr>
<tr>
<td>kVAR</td>
<td>Kilovar</td>
</tr>
<tr>
<td>kW</td>
<td>Kilowatt</td>
</tr>
<tr>
<td>LC</td>
<td>Inductor capacitor</td>
</tr>
<tr>
<td>MATLAB</td>
<td>Numerical Analysis Software</td>
</tr>
<tr>
<td>MPPT</td>
<td>Maximum Power Point Tracking</td>
</tr>
<tr>
<td>MV</td>
<td>Medium Voltage</td>
</tr>
<tr>
<td>PCC</td>
<td>Point of Common Coupling</td>
</tr>
<tr>
<td>pf</td>
<td>Power Factor</td>
</tr>
<tr>
<td>PI</td>
<td>Proportional-Integral</td>
</tr>
<tr>
<td>PLL</td>
<td>Phase Locked Loop</td>
</tr>
<tr>
<td>PR</td>
<td>Proportional-Resonant</td>
</tr>
<tr>
<td>PSIM</td>
<td>Switched-Time Simulation Software</td>
</tr>
<tr>
<td>PV</td>
<td>Photovoltaic</td>
</tr>
<tr>
<td>PWM</td>
<td>Pulse Width Modulation</td>
</tr>
<tr>
<td>RC</td>
<td>Resistance-Capacitor</td>
</tr>
<tr>
<td>RL</td>
<td>Resistance-Inductor</td>
</tr>
</tbody>
</table>
## List of Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_f$</td>
<td>Filter capacitor</td>
</tr>
<tr>
<td>$f$</td>
<td>Frequency in Hz</td>
</tr>
<tr>
<td>$G_{CC,OL}$</td>
<td>Open loop transfer function for current controller</td>
</tr>
<tr>
<td>$G_{CC,CL}$</td>
<td>Closed loop transfer function for current controller</td>
</tr>
<tr>
<td>$G_{VC,OL}$</td>
<td>Open loop transfer function for voltage controller</td>
</tr>
<tr>
<td>$G_{VC,CL}$</td>
<td>Closed loop transfer function for voltage controller</td>
</tr>
<tr>
<td>$H_c(s)$</td>
<td>PR based current controller transfer function</td>
</tr>
<tr>
<td>$H_{PWM}(s)$</td>
<td>Delay transfer function</td>
</tr>
<tr>
<td>$H_v(s)$</td>
<td>PR based voltage controller transfer function</td>
</tr>
<tr>
<td>$i$</td>
<td>RMS current</td>
</tr>
<tr>
<td>$I$</td>
<td>Peak current</td>
</tr>
<tr>
<td>$I_L$</td>
<td>Feedback inductor current</td>
</tr>
<tr>
<td>$I_{ref}$</td>
<td>Reference current for current controller</td>
</tr>
<tr>
<td>$i(t)$</td>
<td>Instantaneous current</td>
</tr>
<tr>
<td>$K_{i,c}$</td>
<td>Current controller integral gain</td>
</tr>
<tr>
<td>$K_{i,v}$</td>
<td>Voltage controller integral gain</td>
</tr>
<tr>
<td>$K_{p,c}$</td>
<td>Current controller proportional gain</td>
</tr>
<tr>
<td>$K_{p,v}$</td>
<td>Voltage controller proportional gain</td>
</tr>
<tr>
<td>$L_f$</td>
<td>Filter inductor</td>
</tr>
<tr>
<td>$M_p$</td>
<td>Active power droop gain</td>
</tr>
<tr>
<td>$M_{pf}$</td>
<td>Power factor droop gain</td>
</tr>
<tr>
<td>$M_q$</td>
<td>Reactive power droop gain</td>
</tr>
<tr>
<td>$P$</td>
<td>Active power</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>( Q )</td>
<td>Reactive power</td>
</tr>
<tr>
<td>( R )</td>
<td>Line resistance</td>
</tr>
<tr>
<td>( T_d )</td>
<td>Transportation delay</td>
</tr>
<tr>
<td>( v )</td>
<td>RMS voltage</td>
</tr>
<tr>
<td>( V )</td>
<td>Peak voltage</td>
</tr>
<tr>
<td>( V_{dc} )</td>
<td>DC link voltage</td>
</tr>
<tr>
<td>( v(t) )</td>
<td>Instantaneous voltage</td>
</tr>
<tr>
<td>( \omega )</td>
<td>Frequency</td>
</tr>
<tr>
<td>( Z )</td>
<td>Line impedance</td>
</tr>
<tr>
<td>( Z_c(s) )</td>
<td>Filter capacitor transfer function</td>
</tr>
<tr>
<td>( Z_l(s) )</td>
<td>Filter inductor transfer function</td>
</tr>
<tr>
<td>( X )</td>
<td>Line inductance</td>
</tr>
<tr>
<td>( \theta )</td>
<td>Impedance phase angle</td>
</tr>
<tr>
<td>( \phi )</td>
<td>Power factor angle</td>
</tr>
</tbody>
</table>
# List of Subscripts

<table>
<thead>
<tr>
<th>Subscript</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>DG number</td>
</tr>
<tr>
<td>$pcc$</td>
<td>Point of common coupling</td>
</tr>
<tr>
<td>$\text{max}$</td>
<td>Maximum value</td>
</tr>
<tr>
<td>$\text{ref}$</td>
<td>Reference value</td>
</tr>
<tr>
<td>$\text{min}$</td>
<td>Minimum value</td>
</tr>
<tr>
<td>$\text{rms}$</td>
<td>Root Mean Square</td>
</tr>
<tr>
<td>$\text{nom}$</td>
<td>Nominal</td>
</tr>
<tr>
<td>$j$</td>
<td>String number</td>
</tr>
<tr>
<td>$n$</td>
<td>Number of PV units</td>
</tr>
<tr>
<td>$v$</td>
<td>Voltage controller</td>
</tr>
<tr>
<td>$c$</td>
<td>Current controller</td>
</tr>
<tr>
<td>$pf$</td>
<td>Pf regulator</td>
</tr>
<tr>
<td>$\text{est}$</td>
<td>Estimated value</td>
</tr>
</tbody>
</table>
Chapter 1
Introduction

Microgrids offer an effective solution for integration of distributed energy resources such as solar and wind. However, these distributed resources are characterised by technical constraints such as intermittency and variability and hence need to be interfaced with the common bus via suitable power electronic devices such as DC/DC and DC/AC converters [11]. Control and coordination of these converter-interfaced generation units are some of the most challenging issues faced by modern power system operators.

This thesis is concerned with the topological arrangement of the Distributed Generation (DG) units within a microgrid and the decentralised power management of these when in islanded mode of operation. This chapter describes the motivation for this study and outlines the central theme of this thesis and its primary objectives. An overview of the thesis structure and the identification of the original contributions of the research are also provided.

1.1. Motivation

Distributed Energy Resources based on renewable technologies are gaining popularity among modern power system operators due to renewable energy advancements and the increased pressure from climate target constraints [6]. These resources, being of stochastic nature, are interfaced with suitable power electronic interface devices and are referred as DG units. The microgrid notion has been developed as an effective way to integrate these DG units dispersed over a wide geographical area. However, the control and coordination of these resources so as to
achieve an operational flexibility comparable with traditional power plants is a matter of concern for long time.

DG units are traditionally arranged in parallel and droop control is often employed to manage the power flow [21],[23],[24]. However, the system demands high dc link voltage at the input side which limits the application to LV-MV integration. This motivates the research on alternative series-cascaded arrangements which can facilitate direct integration of LV units into the grid [49]. Existing studies focus on cascaded structure with dispatchable units only, a configuration that is rarely found in practical scenarios. Therefore, the underlying motivation of this thesis is that a more advanced microgrid structure can be formed by the series arrangement of dispatchable and non-dispatchable DGs. However, this demands development of new power management schemes that can satisfy the control requirements raised by both dispatchable and non-dispatchable DGs in a decentralised manner.

Another recent structure is the hybrid microgrid topology with series-parallel arrangement of DGs [63]. Coordination of the generation units in hybrid architecture requires suitable control strategies to achieve accurate power sharing at both string and micro converter levels. However, due to conflicting control requirements raised by the parallel and cascaded structures, existing control schemes use centralised control architectures. This raise the need of complex communication channels, which limits its flexibility. Hence it is motivating to explore the possibility of extending the series-cascaded microgrid with mixed resources into a hybrid configuration having parallel and series-connected DGs. A decentralised power management scheme is required to ensure adequate power management among the DGs.

1.2. Problem Statement

Compared to the conventional parallel topology, the series-cascaded microgrid topology can offer superior performance in terms of LV-MV integration and harmonic sharing. However, the application of series-cascaded topology is challenging, in particular due to the potential loss of synchronisation among the cascaded DGs. Current literature on series-cascaded microgrid topology focus on the
integration of dispatchable DGs only, which is rare in practice. However, the concept of a realistic microgrid involves mixing non-dispatchable DGs along with dispatchable DGs. Despite beneficial, the integration of non-dispatchable sources such as solar PV along with traditional dispatchable resources brings additional microgrid control requirements, e.g. the need for maximum power point tracking (MPPT) capability, and the need for power curtailment during light load condition (presuming the absence of energy storage within the microgrid). There has been limited research on control techniques that satisfy these supplementary operational constraints without the use of a communication channel linking the units. To address this concern, an alternative series-cascaded structure with dispatchable and non-dispatchable solar PV units is proposed. A decentralised power management scheme that can satisfy the operational constraints raised by dispatchable and non-dispatchable units is developed. These schemes are explained in detail in the subsequent chapters.

Another recent structure is the hybrid microgrid topology with parallel and series-connected DGs. However, existing control requirements for parallel and series-microgrids are contradictory to each other. More specifically, in parallel topology, the active and reactive power sharing is achieved by droop control theory in which voltage and frequency set points are adjusted based on the nature of line impedance, whereas in series-cascaded structure, an inverse droop is typically used to achieve synchronisation among cascaded DGs (a detailed explanation on power sharing characteristics of DGs in different microgrid topologies is provided on Chapter 2).

Hence, existing schemes for parallel or series-cascaded cannot be applied to a hybrid structure in its original form. In addition, there is limited research on the integration of non-dispatchable DGs to hybrid structures. To address this issue, a new hybrid microgrid integrating dispatchable and non-dispatchable DGs is developed in this work. An impedance based power management scheme is proposed that can achieve adequate power sharing among both parallel and cascaded DGs, while ensuring supplementary control requirements raised by non-dispatchable DGs. The resulting system ensures integration of multiple DGs to grid using reduced number of
conversion stages, while still maintaining the redundancy offered by the parallel structure.

### 1.3. Objectives

This thesis focuses on several objectives as described as follows.

**Series-cascaded Microgrid Integrating Solar Photovoltaic Generation**

Existing series-cascaded microgrids focus on dispatchable DGs only. These dispatchable DGs are fully controllable and can operate at constant voltage irrespective of the load current. Hence, autonomous operation of series-cascaded microgrids is not possible. The series-cascaded topology that allows integration of dispatchable and non-dispatchable units is not possible in existing cascaded topologies. Parallel topology requires special harmonic sharing techniques to address the circulating current issue. Therefore, the design, operating voltage, communication, and network is not possible in parallel topology. The proposed series-cascaded topology allows integration of dispatchable and non-dispatchable units and maintains redundancy under all load operating conditions, even when one or more DGs produce zero power or are not in operation.
structure formed solely by dispatchable units can be achieved autonomously using existing methods. However, practical microgrids must incorporate dispatchable and non-dispatchable DG sources. These non-dispatchable DGs are not fully controllable and operate with constrained voltage or load current depending on the MPPT set points, requiring centralised control strategies.

This thesis aims to develop a series-cascaded microgrid structure formed by dispatchable DGs followed by a set of non-dispatchable DGs. A decentralised control strategy referred as power factor versus frequency droop is proposed with the aim of achieving an operational performance for the series-cascaded microgrid that is comparable to the conventional parallel-topology microgrid. The performance matrix of the system is given in Table 1.1. The performance of the new system is confirmed through extensive simulation and experimental investigations.

**Autonomous Power Management of Hybrid Microgrid**

Recent work has proposed a hybrid microgrid topology in which the DGs are structured with series-parallel arrangement. This architecture offers the potential benefits of low gain dc-dc power conversion stages because of cascaded topology and high redundancy offered by parallel topology. However, the power sharing algorithms between parallel and cascaded structures are conflicting to each other and any of these methods cannot be applied to the hybrid structure. Hence, existing research focuses on centralised control schemes for hybrid microgrid. In addition, integration of non-dispatchable DGs to the hybrid structure has not yet been explored.

A further aim of this thesis is to develop a new hybrid structure that can integrate both dispatchable and non-dispatchable DGs. An autonomous power management scheme is then developed that can achieve power sharing at both DG and string levels. The performance matrix of the system is given in Table 1.2.
### Table 1.2: Performance Matrix for Proposed Hybrid Microgrid

<table>
<thead>
<tr>
<th>Performance Feature</th>
<th>Target</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispatchable and non-</td>
<td>Hybrid topology that exploits the advantages of both series and parallel topologies</td>
<td>Existing research focuses on hybrid topologies with dispatchable DGs only</td>
</tr>
<tr>
<td>dispatchable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Communication Network</td>
<td>Achieves power sharing and voltage regulation autonomously without the need of any communication channel</td>
<td>Existing schemes for hybrid topology require central controller and complex communication channels to achieve power sharing and voltage regulation</td>
</tr>
<tr>
<td>Operating voltage flexibility</td>
<td>Non-dispatchable operating voltage is flexible</td>
<td>Operating voltage of DGs in parallel topology is not flexible and must be equal to microgrid operating voltage</td>
</tr>
<tr>
<td>Harmonic sharing</td>
<td>Unlike cascaded topology, in hybrid structure the string units share current proportional to its rating</td>
<td>Suitable harmonic sharing technique is yet to be developed</td>
</tr>
</tbody>
</table>

#### 1.4. Identification of Original Contribution

This thesis provides a series of novel contributions to microgrid structures and decentralised control strategies for islanded mode of operation. These contributions are summarised as follows.

- **Series-cascaded microgrid integrating non-dispatchable units:**
  - Proposes a series-cascaded microgrid integrating dispatchable and non-dispatchable DGs.
  - Develops a set of decentralised control schemes for series-cascaded microgrid that ensures power sharing among DGs while satisfies the operational constraints raised by non-dispatchable DGs. A comparison of existing system with the proposed microgrid structure is summarised in **Table 1.3**.

- **Hybrid microgrid with parallel-cascaded DGs:**
Chapter 1. Introduction

TABLE 1.3: COMPARISON OF EXISTING AND PROPOSED SERIES-CASCADED MICROGRID

<table>
<thead>
<tr>
<th>Performance Feature</th>
<th>Existing parallel</th>
<th>Existing cascaded</th>
<th>Proposed cascaded</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant voltage operation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Dispatchable + Non dispatchable</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>Frequency control</td>
<td>$P - \omega$ and $Q - V$ droop</td>
<td>Inverse $pf - \omega$ droop</td>
<td>$pf - \omega$ &amp; $P - \omega$ droop</td>
</tr>
<tr>
<td>Redundancy</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>Flexibility</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>Harmonic sharing</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>High rating master unit</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
</tbody>
</table>

- Extends the series-cascaded structure with dispatchable and non-dispatchable DGs to form a hybrid architecture with parallel and series units.
- Develops a new decentralised control scheme that ensures power sharing among series and parallel connected DGs in a decentralised manner. A comparative overview of existing system and proposed arrangement in hybrid configuration is given in TABLE 1.4.

1.5. Thesis Structure

The material presented in this thesis is organised as follows.

Chapter 1 (this chapter) provides an overview of the thesis, its structure, and the identification of the original contribution of this work.

Chapter 2 reviews the existing parallel and cascaded microgrid topologies. This review also identifies power sharing algorithms developed for parallel and cascaded microgrid structures.

Chapter 3 introduces the proposed series-cascaded microgrid structure integrating solar PV units. A master control strategy is presented that effectively manages the power sharing among the cascaded units in a decentralised manner.
Chapter 1. Introduction

Chapters 4 introduces the hybrid structure with parallel and cascaded DGs. A specific power management algorithm is presented that ensures power sharing at string and DG levels.

Chapter 5 provides a description of the switched-time simulation systems used in this study. These structures were utilised to generate the simulation results presented in the previous chapters. This chapter also provides an overview of the simulation techniques and approaches adopted in this research.

**TABLE 1.4: COMPARISON OF EXISTING AND PROPOSED HYBRID MICROGRID**

<table>
<thead>
<tr>
<th>Performance Feature</th>
<th>Existing hybrid</th>
<th>Proposed Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispatchable + non-dispatchable</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>Power sharing</td>
<td>Centralised</td>
<td>( P - \omega ) and ( Q - V ) droop</td>
</tr>
<tr>
<td>Redundancy</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Flexibility</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>High rating master unit</td>
<td>X</td>
<td>✓</td>
</tr>
</tbody>
</table>

Chapter 6 provides a description of the set of experimental systems developed to validate the simulated and analytical results presented in this thesis.

Chapter 7 provides a summary of the work presented. Important contributions are reviewed, and proposals are made for future research work.
Chapter 2
Overview of Parallel, Series and Hybrid Microgrids

This chapter provides an overview of microgrid topologies and their primary level of control strategies for islanded mode of operation. The chapter is divided into three main sections.

The first section introduces the basic structure of microgrid and the widely reported control architecture, called as hierarchical control. The following section explores microgrid topologies and their primary level control. The section begins with the traditional parallel topology microgrid where multiple distributed generation (DG) units are connected in parallel and explains how to achieve autonomous power sharing among them using locally measured parameters such as voltage and frequency. The literature review is then extended to series-cascaded microgrids, hereafter called as “series-topology microgrid”, where DGs are cascaded in series. The performance characteristics and control schemes for the series topology microgrids reported to-date are discussed. The section concludes with the review of hybrid microgrid structure comprising series-cascaded and parallel DGs.

Finally, the research direction of thesis is established based on identified limitations of the existing microgrid topologies and control strategies.

2.1. Microgrid Concept

The electric power system architecture is changing, where large central power plants interconnected via grids of vertical transmission and distribution networks are
being transformed into horizontally dispersed small power generating and distribution system [1][2][3]. The driving factor for this transformation is the pressure from climate target constraints, fossil fuel costs and energy security [4]. The conventional high capacity power plants are fuelled by high emitting fossil fuels and are major contributors of greenhouse gas emissions. Therefore, penetration of renewable based DGs such as solar photovoltaic, fuel cells, battery storage system is rapidly increasing due to their low cost and environmental benefits [5][6].

Furthermore, the concept of microgrid has been proposed for efficient and flexible utilisation of these DGs. By definition, a microgrid is a group of interconnected loads and DGs with clearly defined electrical boundaries and acts as a single controllable entity. It may connect or disconnect from the grid to enable it to operate in both grid-connected and island mode [7][8] and thus provides a more flexible and reliable energy system. While microgrids improve the reliability and efficiency of the overall supply system, regulation of voltage and current, and achieving power balance in the such hybrid system are the challenging tasks since renewable based DGs, such as wind and solar, are inherently intermittent as they rely on natural sources. [9][10].

Moreover, the renewable energy resources are interfaced to microgrid through power electronic converters [9][10][11] as shown in Figure 2.1. These power converters are family of solid-state electronic switches (IGBTs, MOFETs) that convert electrical power from one form to another. Timing of these switches are controlled to transform the varying or DC input voltage and current to any other required DC or AC magnitude and frequency. Generally, the interfacing converters are controlled as Current Source Inverter (CSI) for grid-connected mode and Voltage Source Inverter (VSI) for islanded-mode of operation. [12] Since the inverters are switched at high frequency, their output voltage and current contains high frequency harmonics and they draw non-linear current from AC mains. Therefore, an output inductive-capacitive (LC) or inductive-capacitive-inductive (LCL) filter is used in most applications to shape the output voltage and /or current in any desired manner [13][14].
In addition, DGs in microgrid require proper control in place to overcome the challenges associated with overall system control and management such as power sharing, power quality, stability and economic issues, and to achieve an operational flexibility comparable with the conventional power systems. This is usually achieved by means of three level hierarchical control architecture, comprising primary, secondary and tertiary levels, as described in [15][16][17].

- **Tertiary level control**: This is the higher level of control which is responsible for optimising the microgrid operation, and also controls power flow between microgrid.

Figure 2.1: Exemplar microgrid model with power electronics interfaced DGs
This layer is usually slow acting and therefore need low-bandwidth communication channels.

- **Secondary level control:** The secondary controller is assigned to compensate for the steady-state errors in microgrid voltage and frequency and to restore these parameters to nominal values. This control level relays on low-bandwidth communication network to coordinate action among all DGs.

- **Primary level control:** This layer is devoted to control local variables such as voltage, current, and power. Figure 2.2 demonstrates the typical primary level control of DG unit. This control level is employed as cascaded structure with outer power sharing loop and inner current and voltage control loop. The outer loop deploys a strategy to achieve proper power sharing among DGs and to add so-called virtual inertia to microgrid system. Generally, power sharing control can be implemented in centralised [18], distributed [19] or decentralised communication structures. The centralised and distributed schemes rely on the communication network and hence impose system reliability and expandability constraints. Whereas, decentralised schemes rely only on locally measured feedback signals. Usually, autonomous power sharing among DGs are achieved by adjusting the voltage and frequency set points within limits, and this strategy is often referred as a “droop control”. The droop control is preferred because it is simple and easier to implement.

![Figure 2.2: Primary level control block diagram](image)
Chapter 2. Literature Review

and reliable as it does not rely on the communication network. However, the implementation requirements as well as performance of the droop control vary and depends on several factors such as microgrid network characteristics e.g. resistive or inductive, and the structural arrangement of DGs, i.e. paralleled or series-cascaded etc.

This thesis is mainly concerned with the power management at primary level and the following section overviews the existing microgrid topologies and various control strategies adopted at the primary level.

2.2. Overview of Microgrid Topologies and Power Management Strategies

Based on the structural arrangement of DGs, microgrids can be broadly classified into Parallel, Series-cascaded and Hybrid types. The control requirements for DGs vary in all of these topologies due to their varying power regulation principle. Following section describes various microgrid topologies and control strategies adopted at the primary level [20].

2.2.1. Parallel Topology Microgrid

Structural overview: Figure 2.3 shows the traditional parallel topology microgrid where DGs are connected in parallel across the Point-of-Common-Coupling (PCC) [21]. Theoretically, parallel connected DGs can achieve proportional power sharing by adjusting the load current provided their output voltages are synchronised in terms of amplitude, frequency and phase. Practically, the synchronisation of DGs’ voltage is challenging task without communication links due to different lengths of DGs’ feeders. The varying voltage drops across the DGs feeders affects the power sharing accuracy among the paralleled DGs. This may also result in circulating current among them and can damage or overload them[22][23][24].

Power regulation principles and droop control theory: Considering the parallel topology microgrid depicted in Figure 2.3, the active and reactive power injected into common bus by every DG unit can be expressed as[21][25][26][27][28].
where, $k = 1, 2, 3, \ldots.$ represents the DG number, $V_k$ is the output voltage of DG, $V_{pcc}$ is the common bus (PCC) voltage, $Z_k = R_k + jX_k$ is the line impedance with $R_k$ and $X_k$ as resistance and reactance of the line, $\theta_k$ is the phase of output impedance and $\phi$ is the power angle which can be expressed in terms of frequency $\omega$, viz.:

$$\phi = \int \omega$$

The active and reactive power that is produced from each DG depends on the nature of line impedance. Practically, the output impedance can be predominantly resistive or inductive.

**Resistive Line:** When the line impedance is highly resistive, $Z_k = R_k$ and $\theta_k = 0^\circ$, the active power $P_k$ and reactive power $Q_k$ flow between two nodes (i.e., DG and bus) can be expressed as,

$$P_k = \frac{V_{pcc}(V_k - V_{pcc})}{R_k}$$

$$Q_k = \frac{V_k V_{pcc} \phi}{R_k}$$
Thus, for a resistive line, DG active power output can be controlled by adjusting its terminal voltage, and reactive power can be controlled by regulating its frequency or phase. The droop relation for a DG unit with resistive line can be expressed as,

\[ V_{\text{ref},k} = V_{\text{max},k} - M_p \Delta V \]  
\[ \omega_{\text{ref},k} = \omega_{\text{max},k} - M_q \Delta \omega \]

where \( V_{\text{ref},k}, \omega_{\text{ref},k} \) are the reference voltage and frequency of \( k \)th DG, \( P_{\text{max},k} \) and \( Q_{\text{max},k} \) are the maximum real and reactive power limit of DG. \( M_p \) and \( M_q \) are the active and reactive power droop gain which can be defined as,

\[ M_p = \frac{V_{\text{max},k} - V_{\text{ref},k}}{P_{\text{max},k}} \]  
\[ M_q = \frac{\omega_{\text{max},k} - \omega_{\text{ref},k}}{Q_{\text{max},k}} \]

The corresponding illustrative droop curve for the resistive line-based network are shown in Figure 2.4.

**Inductive Line:** When the network line impedance is highly inductive in nature, \( |Z_k| = X_k \) and \( \theta_k = 90^\circ \), the active and reactive power flow can be expressed as,

\[ P_k = \frac{V_k V_{\text{pcc}}}{X_k} \phi \]  
\[ Q_k = \frac{V_k (V_k - V_{\text{pcc}})}{X_k} \]

Figure 2.4: Droop curve for parallel topology microgrids with resistive line (a) Active power droop (b) Reactive power droop
Thus, for the inductive network, the active power is controlled by controlling the DG frequency and the reactive power is controlled by controlling the DG voltage. The droop relation for the inductive network can be expressed as,

\[
\omega_{ref,k} = \omega_{max} - M_p P_{max,k} \tag{2.12}
\]

\[
V_{ref,k} = V_{max} - M_q Q_{max,k} \tag{2.13}
\]

\[
M_p = \frac{(\omega_{max} - \omega_{ref,k})}{P_{max,k}} = \frac{\Delta \omega}{P_{max,k}} \tag{2.14}
\]

\[
M_q = \frac{(V_{max} - V_{ref,k})}{Q_{max,k}} = \frac{\Delta V}{Q_{max,k}} \tag{2.15}
\]

The voltage (rms) and frequency reference obtained from the droop equations (2.12) and (2.13) can be combined to define the instantaneous voltage reference, viz.:

\[
v_{ref,k}(t) = V_{ref,k} \sin(\omega_{ref,k}t) \tag{2.16}
\]

By regulating the voltage reference (2.16), DGs are expected to produce power according to the droop relationships by (2.12) and (2.13) i.e. and their power output will be proportional to their rated capacity. The power sharing accuracy, however, depends on the inverter capability to track the reference voltage accurately and dynamically, which is achieved through an inner current and voltage control loops, as discussed below [29][30][31][32][33][34][35].

**Inner Control Loops:** The purpose of inner control loops is to regulate the inverter voltage and current and to track the instantaneous voltage reference (2.16) and to ensure system stability. These objectives are typically achieved using a dual loop control structure comprising an inner current control loop and outer voltage control loop [31][32], as shown in Figure 2.5. The reference voltage \( v_{ref}(t) \), generated by the outer power sharing loop, is compared with the inverter voltage \( v_o(t) \), and passed through a voltage regulator. The output of voltage regulator is current reference

![Figure 2.5: Inner controller block diagram](image-url)
$i_{ref}(t)$ for the inverter. Similarly, the current reference is compared with the inverter current $i(t)$ and the resultant error signal is fed to a current regulator which produces a modulation signal for the inverter. The voltage and current regulators can be either Proportional + Integral (PI) or Proportional + Resonant (PR) controllers. Practically, PI controller is preferred for DC applications due to its infinite s-domain gain and zero error tracking capability and PR regulator is used for AC application due to the same reasons, i.e., infinite gain and zero error tracking at tuned frequency [33][34][35].

**Performance Validation of Droop Controlled Parallel topology microgrid:** Figure 2.6 shows the simulation setup with two paralleled DGs with predominantly resistive lines. The inverter control comprises outer power loop, i.e., droop control, and dual inner control loops, as shown in Figure 2.2 and Figure 2.5. The DGs are expected to follow the $P-V$ and $Q-\omega$ droop curves and, hence, share power proportional to their rated capacity. The droop gains are chosen as $M_p = (121-114)/5000 = 0.0024$ and $M_q = (51-49)/3000 = 0.000667$. The system performance is verified for different load steps, and the corresponding simulation results are illustrated in Figure 2.7. For easier interpretation of simulation results and system performance, Figure 2.8 illustrates the operating points of DGs on the active and reactive power droop lines under three different load conditions.

Figure 2.7.a depicts the active power sharing performance for the microgrid. The microgrid load is set to 9 kW $t \in [0, 2]$ sec. DG#1 and #2 being of equally rated
share proportional real power of 4.5kW. For \( t \in [2, 4] \) sec, the microgrid active power demand reduces to 4 kW and the DGs active power sharing drops down to 2 kW each. The microgrid load is further reduced to 2.4 kW for \( t \in [4, 6] \) sec, which is followed by reduction in active power sharing of DG#1 and #2 to 1.2 kW each. Thus, both DGs share proportional active power to satisfy the microgrid load demand. Since the modelled line impedance is more resistive in nature, the active power sharing is achieved by adjusting the DGs’ voltage, as shown in Figure 2.7.b. Initially, the microgrid load demands is high at 9 kW for \( t \in [0, 2] \) sec, the operating voltage is drooped to 115.2 V which is near to minimum value set by the droop curve. With reduction in active power demand to 4 kW at \( t \in [2, 4] \) sec, DG#1 and #2 increase their operating voltage set points to 121.2 V so as to ensure proportional power sharing among them. For \( t \in [4, 6] \) sec, the operating voltage further increases to 123.1 V with the reduced load demand.

Figure 2.7.c demonstrates the reactive power sharing for the system under consideration. The microgrid load is purely resistive for \( t \in [0, 2] \) sec and does not demand any reactive power. Therefore, DG#1 and #2 operate at unity power factor so as to ensure power balance. For \( t \in [2, 4] \) sec, the reactive power demand increases to 4.2 kVAR which is equally shared by both the DGs. Then, the load power factor is increased such that the reactive power demand drops to 1.5 kVAR. Accordingly, DG#1 and #2 reduce their reactive power output to 0.75 kVAR.

Figure 2.7.d. shows the microgrid frequency regulation with the change in reactive power requirements. Initially, the microgrid load is set to operate at unity power factor demanding 0 kVAR reactive power for \( t \in [0, 2] \) sec, and therefore, the microgrid frequency reference is redefined to maximum value of 51 Hz. The increase in microgrid reactive power demand to 4.2 kVAR for \( t \in [2, 4] \) sec is followed by the reduction in microgrid frequency to 49.6 Hz while the DGs produce 2.1 kVAR each. For \( t \in [4, 6] \) sec, the microgrid reactive power demand is reduced to 1.5 kVAR, which is subsequently followed by further increase in operating frequency to 50.6 Hz.
Figure 2.7: Simulation results for parallel topology microgrid showing DGs and microgrid (a) Active power (b) Voltage (c) Reactive power (d) Frequency
Limitations of droop control: As demonstrated above, the droop control can ensure proper power sharing among paralleled DGs in a decentralized manner for linear loads. However, the traditional droop method often fails to achieve harmonic power sharing and nonlinear loads. To address this issue, authors in [36][37][38][39][40] have developed a modified droop control scheme to share both linear and non-linear loads in a decentralised manner. In addition, the higher values of DGs droop gains may lead to system instability particularly with the resistive network. To ensure stable operation, authors of [36][37] proposed designing the droop gains based on the small-signal-stability analysis. Another major issue with the traditional droop control is the line impedance coupling. The droop method is based on the concept that the line impedance is predominantly resistive or inductive in nature which may not be the case practically. To address this issue, several droop variants of droop control have been reported in the literature [41][42]. Among the widely used, the virtual impedance method has been proposed to improve the steady-state and transient response of paralleled DGs. In this method, a virtual output impedance loop is proposed to enforce the output impedance of inverters to act as resistive or inductive in nature. The virtual impedance can provide damping required for stable operation and, can also improve harmonic current sharing among the DGs. However, the design of impedance value which can effectively decouple the power flow and at the same time maintain a good system dynamics and
stability is a challenging task. Another variation of the droop scheme is the virtual real and reactive power control scheme reported in [43][44]. In this method, the voltage and frequency are transformed to virtual frame to obtain a completely decoupled relationship between real and reactive power. But in this method the virtual frequency and voltage operation range need to be carefully designed to ensure that the DGs will not operate beyond the permissible voltage and frequency levels. Authors of [44] have proposed an operating range based virtual frame control strategy with consideration of real and reactive power sharing priorities.

Furthermore, power sharing performance of paralleled DG is affected by the line impedance mismatch between the DGs. Authors in [45] reviewed a new control technique that can automatically account for the effect of line impedance and other variations in the unit parameters. The technique can ensure that the paralleled DGs can share linear as well as nonlinear loads under a range of operating conditions.

Despite offering a completely decentralized approach for power sharing, the traditional droop method often needs significant modifications to satisfy the requirements raised by the interfacing DGs and often makes the control architecture more complex. Moreover, all the DGs in parallel topology microgrid have to strictly operate at the same terminal voltage i.e., equal to nominal or user level voltage, which demands the need for high DC voltage (e.g. 650V DC to produce an output voltage of 415V AC) at the DC link of inverter. Usually, this requires either dual conversion stage at the input side of the converter or high-gain DC-DC converter, which adds further complexity to the system architecture and cost and reduces overall efficiency of the system. Therefore, the parallel topology microgrid may not always be the most viable choice for the integration of DGs. Under certain circumstances, the series-cascaded topology microgrids are reported to be a more viable option, as discussed below.
2.2.2. Series-Cascaded Topology Microgrid

As discussed in the previous section, all DGs in parallel topology are required to operate at rated microgrid voltage, demanding intermediate conversion stages with large input-to-output (dc-dc) voltage boost ratios. This high voltage boost requirement can potentially increase system cost and reduce overall system efficiency. To address these issues, authors in [46] [47] [48] have proposed a modular cascaded system where various low-voltage (LV) converters are connected in series to form a string unit so as to attain rated voltage at the output of string unit. The structure has a single output LC filter, and therefore, cannot offer independent control of real and reactive power flow in microgrid or utility grid connected mode of operation. Hence, a more advanced structure has been proposed in [49] where each series-cascaded DG is structured with LC filter at the output side so as to attain high quality voltage at the load side and enable independent control of active and reactive power. This arrangement is referred as a series-cascaded topology microgrid.

**Structural overview of series-cascaded microgrid:** Figure 2.9 shows an exemplary series topology microgrid where DGs are cascaded in series to form a string unit. This topological arrangement allows synthesising a larger AC output voltage from smaller

![Diagram](image_url)

Figure 2.9: Series-cascaded microgrid architecture with low gain boost stage.
AC voltage rated H-bridge inverters. Cascaded topology offers superior performance as compared to parallel topology microgrid, as summarised below.

**Voltage Rating Flexibility:** Unlike parallel-topology microgrid, it is not compulsory for the individual DGs of different power ratings to have the same voltage output, i.e. equal to microgrid or user level voltage. In series topology microgrid, the voltage supplied to loads is the sum of the individual DGs’ voltages, as shown in Figure 2.9. The overall voltage of microgrid is given by,

\[
V_{pcc} = V_{MG} = V_1 + V_2 + V_3
\]  
(2.17)

This inherent characteristic of series-cascaded microgrid facilitates the integration of multiple low-voltage DGs, e.g. solar PV or fuel-cell generators, to utility grid or microgrid using intermediate DC-DC conversion stages with reduced boost voltage ratios, as shown in Figure 2.9.

**Inherent Harmonic Current Sharing:** The individual DGs in parallel-topology microgrid experience circulating currents when the converters of different ratings are connected to the PCC through cables of varied impedance values. Conversely, all DGs in a series-cascaded microgrid carry the same current, which is equal to the load current under all operating conditions, i.e.,

\[
i_{pcc} = i_1 = i_2 = i_3
\]  
(2.18)

Hence, DGs in series-cascaded microgrids experience the same harmonics current, eliminating the need for additional harmonic droop controllers and therefore, series-arrangement of DGs simplifies the control architecture.

**Control requirements of series-cascaded microgrid:** Despite the significant advantages mentioned above, the practical deployment of series topology microgrids is constrained due to the lack of control and power management strategies that could deliver power quality and reliability comparable to parallel topology microgrids.

Although the voltage magnitude of series-cascaded DGs could be set different and lower than the rated voltage of the microgrid, the voltage phase must be synchronised in order to achieve proper power sharing among them [49]. The concept of voltage phase synchronisation is explained with the help of phasor diagrams as shown in Figure 2.10. With all the three DGs operating at same phase angle, the
individual DG voltages sum up to PCC voltage as depicted on Figure 2.10.a. However, with the DGs operating at different phase angle and same magnitude, the DG voltages cannot sum up to rated value as shown on Figure 2.10.b. Similarly, Figure 2.10.d illustrates the case where voltage phase angle is different for each DGs and therefore, active and reactive power produced by each DGs is unequal while apparent power produced by the DGs is the same. On the other hand, when the DGs have same phase angle, i.e., same displacement power factor, all DG produce equal active and reactive power, as depicted in Figure 2.10.c.

The concept of synchronisation and proportional power sharing as described through a phasor diagram, is mathematically given as,[49]

\[ P_i = V_i \sum_{j=1}^{n} V_j \cos(\theta_i - \theta_k + \theta_{load}) \sqrt{|Z_{load}|} \]  

(2.19)
where, $k = 1,2,3...$ represents the DG number, $\theta_1$ and $\theta_k$ are the voltage phase angle of the 1st and kth DG, $\theta_{load}$ is the load power factor angle, and $Z_{load}$ is the load impedance. Note that the power output equation for the series-cascaded DGs is different from the paralleled DGs. Unlike paralleled DGs, the active and reactive power output of series-cascaded DGs depends on the voltage phase angle $\theta_1 - \theta_k$.

Based on (2.19) and (2.20), series-cascaded DGs will share power proportional to their rated voltage and hence power if the voltage phase is synchronised, i.e., $\theta_1 = \theta_k$.

**Series-cascaded microgrid control strategies**: To-date, several control strategies have been reported in the literature to address the synchronisation issue for series-cascaded DGs in islanded mode of operation. Authors in [50] have described a centralised method for the control of cascaded DGs. In this method, a centralised controller is tasked to send supervisory commands to all the converters through a high-bandwidth communication link. On positive note, the centralized control is robust and maintains system balance during active and reactive power variation and under grid fault. However, control decision made by the central controller heavily relay on the communication network and measured signals including grid voltage, current and DC bus voltage. Moreover, centralised control techniques may not be cost effective and reliable as compared to decentralized techniques.

In order to reduce the complexity of communication system, authors in [51] [52] have developed a distributed control method that can achieve the required control strategy with minimum communication requirement. This control scheme includes one DG module with a current-mode control and is called “Master Controller”. The remaining DGs in the string control their output voltage and are called “Slave Controllers”. The scheme is prone to single-point failure, the loss of any one unit can cause the entire system to malfunction.

A fully decentralised scheme referred as “inverse power factor (pf) droop” control has been reported in [49][53] to address the synchronisation issue with the series-cascaded microgrid in islanded mode of operation. In this method, equal power
sharing performance is achieved by adjusting the microgrid frequency as inverse function of DG power factor, viz.:

\[
\omega_{ref,k} = \omega_{min} + M_{pf} p f_{max}
\]

\[
V_{ref,k} = V_{k,nom}
\]

where \( k = 1, 2, 3, \ldots \) represents the DG number, \( \omega_{ref,k} \) is the microgrid reference frequency, \( \omega_{min} \) is the microgrid frequency at full-load and \( M_{pf} \) is the inverse power factor droop gain and \( p f_{max} \) is the maximum operating power factor, \( V_{ref,k} \) is the DG operating voltage and \( V_{k,nom} \) is the nominal operating voltage for the DGs. Notably, power sharing with this method is achieved by controlling the DGs’ frequency alone while their operating voltage remain constant at the nominal rated value. This method can achieve accurate active and reactive power sharing performance in islanded mode without any communication channels. However, this method is applicable to linear loads only. In addition, the inverse power factor control can cause deviations in steady-state PCC voltage magnitude and frequency, particularly when the line impedance is high.

Authors of [54] reviewed the power factor consistency scheme which is applicable to both resistive-inductive and resistive-capacitive types of loads. In this scheme, the synchronisation among the cascaded DGs is achieved by controlling the microgrid frequency as an inverse function of active power. Another synchronization method has been proposed in [55][56][57][58] where the power sharing is achieved by controlling the frequency in terms of \( P/Q \) ratio. In all these synchronization and power sharing methods, the desired system performance is achieved by adjusting the frequency alone and the operating voltage of the DGs remain the same as the nominal rated value. To further enhance the performance of series-cascaded microgrid, a hierarchical control scheme has been developed in [59]. This method can achieve a flexible power sharing according to the SoC of the battery while maintaining an accurate PCC voltage magnitude and frequency regulation in the microgrid.

It must be noted that, all the control strategies reported above for series-cascaded microgrids are focused on dispatchable type DGs alone, assuming all DGs are fully controllable. However, the concept of microgrid can be realistic only with
integration of non-dispatchable DGs like solar PV along with the dispatchable DGs.[60][61] The integration of non-dispatchable sources such as solar PV along with the traditional dispatchable resources raises additional microgrid control requirements, e.g. the need for MPPT and power curtailment during light load condition (presuming the absence of energy storage within the microgrid) etc[62]. To-date, limited research has been carried on the control techniques that satisfy these supplementary operational constraints without the use of communication channels. This theme of research is one of the main focus of this thesis.

### 2.2.3. Hybrid Topology Microgrid

The parallel and series-cascade topologies are characterised by their benefits and drawbacks as discussed in the previous sections. The parallel topology microgrids exhibit superior performance in terms of redundancy, whereas series topology microgrids are more attractive due to the flexibility of using the DG sources with lower DC link voltage, and superior power sharing and voltage regulation performance, as summarised in Table 2.1. Motivated by these benefits, recently a hybrid structure has surfaced that can offer the voltage flexibility offered by series structure while at the same time maintaining the redundancy of parallel structure [63].

However, the power regulation principle for paralleled and -series-cascaded DGs defined by [21]-[28] and [49]-[59] respectively, are contradicting each other as

**Figure 2.11: Hybrid topology microgrid**
detailed in section 2.2. Therefore, the existing as well as the newly proposed control strategies in this thesis for the standalone series and parallel topology microgrid cannot be applied to hybrid microgrids in their original form. Moreover, there is a limited research on hybrid microgrid topology. In [63], the authors have developed a centralised control scheme that needs a central controller and low-bandwidth communication channels. But the system lacks flexibility due to need of communication channels and central controllers. In addition, as discussed for the series-cascaded structure in previous section, the possibility of integrating dispatchable and non-dispatchable into hybrid form is yet to be explored. Hence, this thesis aims to develop a new hybrid microgrid structure that allows integration of non-dispatchable DGs and to develop suitable power management scheme that is fully decentralised.

### Table 2.1: Performance Comparison for Different Microgrid Topologies

<table>
<thead>
<tr>
<th>Performance Feature</th>
<th>Parallel</th>
<th>Series-Cascaded</th>
<th>Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redundancy</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>DC-DC Boost stage</td>
<td>Dual stage or High-gain single</td>
<td>Low gain single stage</td>
<td>Low gain single stage</td>
</tr>
<tr>
<td>Harmonic sharing techniques</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>Single point failure</td>
<td>X</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Dispatchable + Non-dispatchable</td>
<td>✓</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

### 2.3. Summary

This chapter has reviewed the concept of microgrid, and the hierarchical control scheme reported in the literature. In-depth analysis of primary level control is conducted with emphasis on power sharing algorithms for various microgrid topologies reported in the literature, as summarised in Table 2.2.

The analysis has indicated that the parallel topology microgrid can offer superior performance in terms of redundancy and flexibility. This microgrid topology is found to be an attractive choice among others for very long time partly because the
control and power management technology for the parallel microgrid is well documented in the literature and tested in the field. However, as highlighted earlier, all DGs need to operate at rated PCC or user-level voltage which may demand dual stage conversion at the input side of the inverter, especially for the low power capacity DGs.

The analysis has indicated that the series-cascaded microgrid topology can be a better solution to integrate multiple low voltage and capacity DGs to microgrid or utility grid with reduced conversion stages. However, the existing control strategies for series-cascaded microgrid are not applicable to non-dispatchable DGs. In addition, the series-topology as well as the existing control schemes are prone to single point failure.

The survey has pointed out that the recently developed hybrid structure with parallel and series connected DGs can achieve the benefits of both - parallel and series-cascaded topologies. However, due to conflicting control requirements raised by the parallel and cascaded structure, the decentralised power management for hybrid structure is not addressed till now. Also, the possibility for integration of non-dispatchable into hybrid topology is not explored till now.

The issues and shortcomings highlighted in the above paragraphs have been addressed in the subsequent chapters of the thesis.
### Table 2.2: Summary of Reported Control Strategies for Different Microgrid Topologies

<table>
<thead>
<tr>
<th>Topology Type</th>
<th>Control strategy</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel topology microgrid</td>
<td>Traditional droop control [21][25][26]</td>
<td>Can achieve proportional power sharing for linear loads only</td>
</tr>
<tr>
<td></td>
<td>Modified droop methods [37]</td>
<td>Extend the droop concept for resistive line</td>
</tr>
<tr>
<td></td>
<td>Virtual impedance method [41] [42]</td>
<td>Proposed to improve transient and steady state response.</td>
</tr>
<tr>
<td></td>
<td>Virtual real and reactive power Control [43] [44]</td>
<td>Proposed to address decoupling issue with active and reactive power sharing</td>
</tr>
<tr>
<td></td>
<td>Enhanced droop method [45]</td>
<td>Proposed to compensate for the effect of line impedance</td>
</tr>
<tr>
<td></td>
<td>Inverse pf droop [49]</td>
<td>Can achieve accurate power sharing for RL type loads</td>
</tr>
<tr>
<td></td>
<td>Power factor consistency droop [54] [55]</td>
<td>Proposed to achieve power sharing for RI and RC type of loads.</td>
</tr>
<tr>
<td></td>
<td>SoC based droop [56]</td>
<td>Can achieve synchronization and power sharing for cascade structure with storage element</td>
</tr>
<tr>
<td>Series-Cascaded Microgrid</td>
<td>Hierarchical control [59]</td>
<td>Extended version of inverse pf droop method to achieve flexible power sharing</td>
</tr>
<tr>
<td>Hybrid Microgrid</td>
<td>Centralised control [63]</td>
<td>Need central controller and communication channels</td>
</tr>
</tbody>
</table>
Chapter 3
Series-Cascaded Microgrid with Non-Dispatchables

Series-cascaded microgrids can offer superior performance compared with parallel topology in terms of harmonic sharing and voltage rating flexibility. But DGs in series-cascaded topology require proper strategy to achieve synchronization among their voltage phases which is necessary for an autonomous voltage regulation and proportional power sharing among them. The existing control strategies explained in Section 2.2.2 can well address this issue but they are applicable to fully-controllable and dispatchable DGs only. The concept of microgrid can be more realistic with the integration of non-dispatchables along with dispatchable DGs. The existing control strategies for series topology microgrid cannot ensure coordinated operation of dispatchable and non-dispatchable DGs.

The research work in this chapter addresses this issue by developing a new microgrid structure formed by series arrangement of dispatchable and non-dispatchable DGs, i.e., solar PV units. A decentralised Master-slave control strategy is developed that can achieve proper power sharing among cascaded DGs while satisfying the operational constraints raised by non-dispatchable DGs, like MPPT tracking, power curtailment etc.


3.1. Proposed Series-cascaded Microgrids

The series-cascaded microgrid topology proposed in this work is formed by a dispatchable DG followed by set of non-dispatchable solar PV units as illustrated in Figure 3.1. The dispatchable DG is assigned as master controller, while all solar PV units operate following their individual MPPT set points and are referred as slaves. The master unit is rated at the nominal microgrid capacity and operating voltage. Note that this requirement would also apply if the topology was parallel instead of series-cascaded.

The main features of series-cascaded microgrid topology and associated control scheme proposed in this work are summarised as follows.

**Dispatchable + Non-Dispatchable:** The proposed microgrid structure can integrate dispatchable and non-dispatchable solar PV units in a series-cascaded manner with

![Diagram of Series-cascaded Microgrid with solar PV units](image-url)

Figure 3.1: Series-cascaded microgrid with solar PV units
maximum utilisation of non-dispatchable at any load condition.

**Autonomous control:** The proposed control approach ensures power sharing across the series-cascaded DGs, while satisfying the operational constraints realised by dispatchable and non-dispatchable DGs in a decentralised manner.

**MPPT tracking:** In proposed control all dispatchable DGs can harness the maximum MPPT power available and power curtailment is activated under light load conditions due to absence of storage element.

**Voltage and Power Balancing:** In proposed control architecture, the microgrid voltage is always maintained at rated value irrespective of the load variation and the power sharing among DGs is achieved by controlling microgrid frequency.

**Reliability:** The microgrid steady-state voltage is maintained at rated value under all operating conditions even when one of the DGs is not in operation. Thus, it can offer system reliability comparable to conventional parallel topology.

**Voltage rating flexibility:** The proposed architecture provides flexibility in allowing lower nominal operating voltage for the non-dispatchable DGs (i.e. lower than the microgrid nominal voltage).

### 3.2. Master-slave Control Strategy for Proposed Series-Cascaded Microgrids

In order to satisfy the operational constraints raised by both dispatchable and non-dispatchable DGs in an autonomous manner, the microgrid control is performed using two alternative strategies: (a) power factor versus frequency droop \( \text{pf - } \omega \) droop) and (b) active power versus frequency droop \( P - \omega \) droop). In both control strategies, the dispatchable DG is assigned to regulate the microgrid frequency within a small range and is referred as master unit. Whereas the non-dispatchable DGs follow the microgrid frequency set by the master and is referred as slave units. The detailed control architecture for the master and slave units are explained in the following sections.
3.2.1. Power factor versus frequency \((pf - \omega)\) droop control

**Master DG:** The control block diagram for the master DG is shown in Figure 3.2. In order to decentralise the entire control scheme and to pass on the microgrid loading information to the slave units, the master frequency is regulated according to the microgrid power factor using the droop relation,

\[
\omega_{\text{ref}} = \omega_{\text{max}} - M_{pf} \times pf_{MG}
\]  

(3.1)

where \(\omega_{\text{ref}}\) is the reference operating frequency of the master unit, \(\omega_{\text{max}}\) is the no-load microgrid frequency, \(M_{pf} = \Delta \omega / 1\) is the droop gain and \(pf_{MG}\) is the load power factor.

With the frequency reference obtained in (3.1), the instantaneous voltage reference for the master DG is defined as

\[
v_{\text{ref},1}(t) = V_{\text{nom},1} \sin(\omega_{\text{ref}} t)
\]

(3.2)

where \(V_{\text{nom},1}\) is the microgrid nominal voltage for master DG to regulate, and \(\omega_{\text{ref}}\) is the microgrid frequency defined by (3.1)

Since the master DG is positioned at top in series cascaded structure, the voltage measured at its terminal (with respect to load common phase) is the overall voltage of the microgrid. Therefore, the master DGs can regulate the overall microgrid voltage.

Figure 3.2: Series-cascaded microgrid with \(pf - \omega\) control, (a) Control block diagram and (b) \(pf - \omega\) droop curve for the master DG
under all load conditions.

The voltage reference (3.2) for the master DG is tracked using a typical dual-loop control architecture, as shown in Figure 3.2 and mathematically given by (3.3) and (3.4). The dual loop control architecture has been explained in Section 2.2

\[ i_{ref,1} = H_v(s)(v_{ref,1} - v_j) = \left( K_{p,v} + \frac{K_{i,v}s}{s^2 + \omega^2} \right)(v_{ref,1} - v_j) \]  

(3.3)

\[ V_{PWM,1} = H_c(s)(i_{ref,1} - i_1) = \left( K_{p,c} + \frac{K_{i,c}s}{s^2 + \omega^2} \right)(i_{ref,1} - i_1) \]  

(3.4)

where, \( H_v(s) \) and \( H_c(s) \) are the Proportional-Resonant (PR) based voltage and current controllers tuned to fundamental frequency so as to achieve zero steady-state error for voltage and current reference, i.e., \( i_{ref,1} - i_1 = 0 \) and \( v_{ref,1} - v_j = 0 \), \( V_{PWM,1} \) is the PWM reference voltage to compare with the triangle carrier signal to generate PWM switching pulses. \( K_{p,v} \) and \( K_{p,c} \) are the proportional gain for the voltage and current controllers, respectively, and \( K_{i,v} \) and \( K_{i,c} \) are the integral gain for the voltage and current controller respectively. \( \omega \) is the fundamental frequency i.e., 50 Hz.

Note that the voltage controller defined on for master unit is defined with the string voltage as the feedback variable (instead of its own terminal voltage). This will ensure constant microgrid voltage irrespective of the voltage variation in slave units.

**Slave DG:** Figure 3.3 shows the control block diagram for the slave DG. The slave units are responsible for dispatching real power according to their individual MPPT set-points. This is achieved by defining the individual unit voltage reference proportional to its MPPT setpoint, viz.:  

\[ V_{ref,n} = \frac{P_{max,n}}{I_n \times pf_{ref,n}} \]  

(3.5)

Where \( n = 2, 3, .. \) indicates the slave DG number in series structure, \( I_n \) is the slave DG current (which is the same across all the series cascaded DGs), \( P_{max,n} \) is the MPPT set point power (after taking into consideration the necessary curtailment constraint – further details on the curtailment strategy are provided in subsequent
section), and $p_{f,ref,n}$ is the reference power factor for the slave DGs, which can be defined by one of two possible user defined operating modes, explained as follows.

**Power Factor Dispatch Mode:** This operating mode is characterised by having the slave DG units tracking their individual Maximum Mower Point Tracking (MPPT) set points to harness maximum energy from the solar cells, while the reactive power dispatch operates according to a commanded reference power factor or reactive power. This includes unity power factor reference in which the slave DGs are assigned for active power sharing only and master DG satisfy the entire reactive power demand while compensating for the active power shortfall according to MPPT reference.

**Power Factor Synchronisation Mode:** In this mode of operation, the slave DGs $p_f$ are synchronised with the microgrid power factor ($p_f$). This implies the slave DGs autonomously operate at the load power factor, while active power dispatch is based on their individual MPPT set points. The load power factor can be determined from the microgrid frequency in a decentralised manner by employing reverse power factor droop defined by (3.1), i.e.

$$p_{f,ref,n} = \frac{\alpha_{max} - \alpha_n}{M_{pf}} \approx p_{f} = p_{f,ng}$$

(3.6)
Where $\omega_n$ is the system frequency (set and regulated by the master DG) measured from that can be measured from the slave units terminal current or voltage using a phase-locked-loop (PLL) structure.

Based on this mode of operation, the desired reactive power dispatch is achieved by means of a power factor regulator. The power factor regulator tracks the reference power factor ($p_{f_{\text{ref}}, n} - p_{f_n} = 0$) by generating the compensation angle $\theta_{\text{ref}, n}$ viz.,

$$\theta_{\text{ref}, n} = \left( K_{p, pf} + \frac{K_{i, pf}}{s} \right) \left( p_{f_{\text{ref}}, n} - p_{f_n} \right) \quad (3.7)$$

where, $K_{p, pf}$ and $K_{i, pf}$ are the proportional and integral gains for the power factor regulator. This compensation angle $\theta_{\text{ref}, n}$ is added to the slave DG reference voltage to ensure desired power factor sharing performance. The instantaneous voltage reference voltage for slave units is redefined as:

$$v_{\text{ref}, n}(t) = V_{\text{ref}, n}(t) \sin(\omega_n t + \theta_{\text{ref}, n}) \quad (3.8)$$

where, $V_{\text{ref}, n}$ is the operating voltage proportional to MPPT set point as defined by (3.8), $\omega_n$ is the system frequency measured using PLL, and $\theta_{\text{ref}, n}$ is the compensation angle regulator as defined by (3.7). Similar to the master DGs, the instantaneous voltage reference (3.8) for the slave DG is tracked using the dual loop voltage and current controller as shown in Figure 3.3 and mathematically given as:

$$i_{\text{ref}, n}(s) = H_{v,n}(s) \left( v_{\text{ref}, n} - v_{n} \right) \quad (3.9)$$

$$V_{\text{PWM}, n}(s) = H_{c,n}(s) \left( i_{\text{ref}, n} - i_{n} \right) \quad (3.10)$$

where, $H_{v,n}(s)$ and $H_{c,n}(s)$ are the PR based voltage and current controllers. The voltage controller tracks the voltage reference $v_{\text{ref}, n} - v_{n} = 0$ and generates current reference $i_{\text{ref}, n}$ which is tracks by the current controller. The output of the current controller is the reference voltage $V_{\text{PWM}, n}$. This reference voltage is compared with high-frequency carrier triangle waveform so as to generate switching signals.
PV Curtailment: In the proposed series-cascaded topology, the microgrid is formulated without considering any expensive storage element. Hence the surplus MPPT power should be curtailed when the overall MPPT set-point exceeds the actual power demand. In order to ensure this, the maximum power dispatch from the individual slave DGs is constrained viz.:

$$P_{\text{max},n} = \frac{P_{\text{nom},n}}{P_{\text{nom,mg}}} \times P_{\text{mg,est}}$$

(3.11)

The microgrid active power $P_{\text{mg}}$ loading can be estimated using the terminal voltage frequency information, viz.:

$$P_{\text{mg,est}} = V_{\text{nom,mg}} \times I_n \times pf_{\text{mg}}$$

(3.12)

where $V_{\text{nom,mg}}$ is the rated microgrid voltage which is constant, and $pf_{\text{mg}}$ is the microgrid power factor, determined from the reverse power factor droop as given by (3.6) and $I_n$ is the slave DG current.

3.2.2. Active Power versus frequency ($P-\omega$) droop control

In addition to above mentioned operational features, this control approach aims to improve the utilization of the DGs power capacities, by prioritize their reactive power dispatch based on their active power loading.

Master DG: As illustrated in Figure 3.4, in this control strategy the microgrid frequency is regulated by the master DG according to its active power output. Consequently, the frequency information is used by the slave DGs to assess the master DG active power state of loading and to prioritize their reactive power dispatch autonomously, i.e., when the master DG is lightly loaded, the slave DGs reduce their reactive power output so as to shift the reactive power to the master DG, and vice versa. This objective is achieved using $P-\omega$ droop relation for the master DG, as defined by

$$\omega_{\text{ref}} = \omega_{\text{max}} - M_p P_1$$

(3.13)

where $\omega_{\text{ref}}$, $\omega_{\text{max}}$, and $P_1$ are the reference frequency, no-load frequency, active power output of the master DG and $M_p = \Delta \omega / P_{\text{max},1}$ is the droop gain.
Chapter 3. Series-Cascaded Microgrid with Non Dispatchables

Figure 3.4: Series-cascaded microgrid with $P-\omega$ control, (a) Control block diagram and (b) $P-\omega$ droop curve for the master DG

**Slave DG:** Figure 3.5 shows the control block diagram for the slave DGs with $P-\omega$ control strategy. The prioritisation of reactive power dispatch by the slave DGs after taking into consideration the master DG active power loading is achieved via a proposed reactive power droop control, viz.:

$$Q_{n,\text{ref}} = \frac{\omega_{\text{max}} - \omega_{\text{ref}}}{M_p}$$  \hspace{1cm} (3.14)

The reference reactive power set by (3.14) depends on system frequency which reflects the active power loading condition of the master DG, e.g. the slave DGs increase their reactive power output when active power loading on the master DG high. Using (315), the reference power factor for each slave DG can be calculated as:

$$p_{f,\text{ref}} = \frac{P_{p,\text{max}}}{\sqrt{P_{p,\text{max}}^2 + Q_{f,\text{ref}}^2}}$$  \hspace{1cm} (3.15)

Each slave DG tracks the individual reference power factor by means of their power factor regulators, as explained in the previous section 3.2.1. This control strategy ensures that as the master DG loading reduces, the slave DG reduce their reactive power output and vice versa.
Curtailment: In this control strategy, the system frequency carries loading information of the master DG i.e., $P_1$. Accordingly, the overall microgrid active power load can be estimated using,

$$P_{mg} = P_1 + \left( \frac{P_{mg,nom}}{P_{n,nom}} \right) \times P_{MPPT}$$

(3.16)

3.3. Simulation and Experimental Validation

The Master-slave control strategy for the proposed series-cascaded structure has been extensively evaluated using time-based switched simulations and matching experimental investigations.

3.3.1. Validation System

The system performance is first validated for a single phase microgrid where dispatchable master DG#1 is rated at 1 kVA, 120 V and two slave DGs are rated at 0.5 kVA, 60 V, each. The slave DGs MPPT reference is set at 0.15 kW for $t \in [0, 1]$ sec, 0.30 kW for $t \in [1, 2]$ sec and 0.45 kW for $t \in [2, 3]$ sec. The system frequency is allowed to vary over a range of $\omega_{min} = 49 \text{Hz} \leftrightarrow \omega_{max} = 51 \text{Hz}$ and the
droop gain is chosen as $M_p = (51 - 49) / 2 = 2$. A complete description of the simulation and experimental systems is given in Chapter 5 and Chapter 6.

### 3.3.2. Simulation Results

**Scenario A: Power factor vs. frequency (pf-ω) droop:**

In this scenario, the cascaded structure is formed by three DGs as shown in Figure 3.6.a. The microgrid load is set to 0.7 kW, 0.7 pf and the system performance is verified for different modes of operation. Since the microgrid is operating at 0.7 power factor, the expected microgrid frequency according to the droop setting (Figure 3.6.b) is 49.6 Hz. This expected result is shown in Figure 3.7.a.

The active power sharing performance for the microgrid scenario-A is shown in Figure 3.7.b. The slave DGs are tracking the MPPT set points, 0.15 kW for $t \in [0, 1]$ sec and 0.30 kW for $t \in [1, 2]$ sec. However, for $t \in [2, 3]$ sec, the overall MPPT power reference exceeds the microgrid load demand and hence the active power is curtailed at 0.35 kW. With the curtailment active for $t \in [2, 3]$ sec, the entire microgrid/string load is met from slave DG#2 and DG#3, i.e., solar PV units, alone and master DG#1 power output drops down to zero.

Corresponding voltage sharing performance is shown on Figure 3.7.c in which the slave DGs are changing the voltage profile to track the varying MPPT, and master

![Diagram](image)

Figure 3.6: Series-cascaded microgrid (a) Simulation setup (b) pf - ω droop curve
DG adjust its operating voltage so as to maintain nominal grid voltage of 120V.

Synchronisation mode: Figure 3.8a shows the power factor sharing performance when slave DGs are operating in power factor synchronisation mode. In this case, the slave DG#2 and #3 tracks the microgrid power factor of 0.7 for $t \in [0, 3]$ sec and DG#1 compensates for the active and reactive power shortfall by operating at power factor of 0.7 for $t \in [0, 1]$ sec, 0.85 for $t \in [1, 2]$ sec and 0.95 for $t \in [2, 3]$ sec. Figure

Figure 3.7: Simulation results for series-cascaded microgrid Scenario A showing (a) microgrid frequency, and (b)Active power output (c) Voltage of the DGs and microgrid
3.8.b demonstrates the reactive power sharing performance for this mode of operation. Corresponding to every MPPT incremental step, slave DG#2 and #3 adjust their reactive power reference based on the reference pf calculated using (3.6), as 0.14 kVAr for \( t \in [0, 1] \) sec, 0.32kVAr for \( t \in [1, 2] \) sec and 0.35 kVAr for \( t \in [2, 3] \) sec. Master DG#1 compensates for the overall reactive power demand by sharing 0.42 kVAr, 0.06kVAr for \( t \in [0, 1] \) and \( t \in [1, 2] \) sec, respectively and 0kVAr for \( t \in [2, 3] \) when the active power curtailment is active for the salve DGs.

The corresponding voltages produced by the DGs and microgrid is shown in Figure 3.7.c and explained through phasor diagram, as shown in Figure 3.9. Figure 3.9.a corresponds to the voltage and power phasor when MPPT setpoint is 0.15kW. Slave DG#2 and 3 operate at 26 V/each while delivering 0.15kW+j0.14kVAr and maintaining the microgrid power factor of 0.7 (since they are operating in synchronisation mode). The master DG operates at 68 V and compensates for the overall power demand by producing 0.40kW+j0.42kVAr.

The voltage phasor diagram in Figure 3.9.b corresponds to MPPT setpoint of 0.30kW, for \( t \in [0, 1] \) sec. Slave DG#2 and #3 increase their operating voltage to 52 V and operate at 0.30kW+j0.32kVAr. Master DG#1 voltage magnitude and angle is changed so as to operate at 16 V at power factor of 0.85 while producing the shortfall of 0.010kW+0.06kVAr.

Figure 3.9.c demonstrates operating points of the DGs when the load demand is low and slave power output is curtailed at 0.45 kW. During this interval, the microgrid load demand is met by the salve DGs alone. DG#2 and #3 operate at 60 V while sharing 0.35kW+j0.35kVAR/each and master DG#1 voltage as well as power drops down to zero to avoid reverse power flow, assuming it is permitted by the inverter topology.

Thus, in synchronisation mode of operation, the slave DG units are tracking the MPPT profile for active power dispatch. Whereas the reactive power sharing is adjusted so as to maintain synchronisation with microgrid power factor. The master DG compensates for the overall real and reactive power demand maintaining overall string power as 0.7kW+0.7kVAr.
Figure 3.8: Simulation results for series-cascaded microgrid Scenario A showing DGs and microgrid (a) Power factor and (b) Reactive power output for synchronisation mode operation

Figure 3.9: Phasor diagram for series-cascaded microgrid Scenario A (a) MPPT=0.15kW (b) MPPT=0.3kW (c) MPPT=0.45kW for synchronization mode operation
**Power factor dispatch mode:** Figure 3.10 shows the system performance for power factor dispatch mode of operation when the slave DGs power factor is set to 0.85. As shown in Figure 3.10.a, Slave DG#2 and #3 track the commanded power factor of 0.85 for $t \in [0, 3]$ sec by producing $0.09\text{kVAr}$ for $t \in [0, 1]$ sec, $0.18\text{kVAr}$ for $t \in [1, 2]$ sec and $0.21\text{kVAr}$ for $t \in [2, 3]$ sec respectively. Accordingly, master DG#1 compensates $0.52\text{kVAr}$ at 0.6pf for $t \in [0, 1]$ sec, $0.34\text{kVAr}$ at 0.3pf for $t \in [1, 2]$ sec.

While curtailment is activated for $t \in [2, 3]$ sec, the master DG#1 reactive power dispatch is set to 0.28 kVAr.

The corresponding phasor diagram is given on Figure 3.11. The system performance for the MPPT setpoint of 0.15kW is shown in Figure 3.11.a. In this case slave DG#2 and #3 track the MPPT setpoint of 0.15 kW and commanded power factor of 0.85 by operating at 22 V while sharing $0.15\text{kW}+j0.09\text{kVAR}$. Master DG#1 satisfies the overall power demand and operates at 76 V and $0.40\text{kW}+j0.52\text{kVAR}$.

The phasor diagram in Figure 3.11.b corresponds to $t \in [1, 2]$ when MPPT set point is increased to 0.30kW. Slave DG#2 and #3 maintains the commanded power factor of 0.85 by operating at $0.30\text{kW}+j0.18\text{kVAR}$ with voltage output of 42 V. Slave DG#1 operates at 34 V while sharing $0.10\text{kW}+0.34\text{kVAR}$ at 0.3 power factor so as to meet the overall load demand of $0.70\text{kW}+0.70\text{kVAR}$.

Figure 3.11.c corresponds to curtailment condition, during $t \in [2, 3]$, when DG#2 and #3 share $0.35\text{kW}+j0.21\text{kVAR}$ at 50V. Master DG#1 power factor drops down to zero while it is producing 20V and $0\text{kW}+0.28\text{kVAR}$.

Thus, the slave DGs always produce active power according to their MPPT setpoints and reactive power based on the commanded power factor control – dispatched or synchronized with master. Accordingly, the slave DGs adjust their voltage output (magnitude and phase angle).
Figure 3.10: Simulation results for series-cascaded microgrid Scenario A showing DGs and microgrid (a) Power factor and (b) Reactive power output for power factor dispatch mode operation.

Figure 3.11: Phasor diagram for series-cascaded microgrid Scenario A (a) MPPT=0.15kW (b) MPPT=0.3kW (c) MPPT=0.45kW for power factor dispatch mode operation.
Chapter 3. Series-Cascaded Microgrid with Non Dispatchables

**Power factor dispatch mode – unity pf operation:** The expected performance is achieved with the commanded power factor of unity, as shown in Figure 3.12. During this mode of operation, master DG#1 compensates for the active and reactive power variation caused by the slave DGs and thus maintains power balance in the microgrid. As shown in Figure 3.12, Slave DG#2 and #3 operate at unity power factor for \( t \in [0, 1] \) sec and master DG#1 operates at pf of 0.7 while generating 0.7kVAR for the entire range of operation, i.e., \( t \in [0, 3] \) sec.

The phasor representation for unity power factor operation is shown in Figure 3.13. As shown in Figure 3.13.a, slave DG#2 and #3 produce 18V and 0.15kW+0kVAR, ensuring unity power factor. Master DG#1 satisfies the overall power demand by sharing 0.40kW+0.70kVAR at 0.5 pf with operating voltage of 84V. Figure 3.13.b corresponds to \( t \in [1, 2] \) and MPPT setpoint of 0.3kW where slave DG#2 and #3 produce 36V and 0.30kW+j0kVAR. Master DG#1 operates at 48 V and 0.10kW+0.7kVAR, meeting the overall power demand of 0.70kW+0.7kVAR. Figure 3.13.c. represents the curtailment condition during \( t \in [2, 3] \). Slave DG#2 and #3 operate at 0.35kW+0kVAR/each and 42V/each and master DG#1 operates at 36V and 0kW+0.70kVAR.

The simulation result showed that the slave DG are producing active power according to their MPPT setpoints and reactive power based on the commanded power factor reference. The slave DG adjusts their voltage (magnitude and phase) so as to in accordance with the changing active power (MPPT) and power factor references.

Further evaluation of the system performance is conducted by considering the MPPT time varying profile rescaled from the actual irradiance day profile. Figure 3.14.a shows the active power sharing performance, where the slave DGs follow the MPPT reference power and the master DG compensates for the shortfall. The power curtailment of slave DGs is activated for a short period of time, i.e. \( t \in [2.5, 3] \) sec, while the power demand is solely met by the slave DGs. Note that while the power output of the string DGs changes continuously, the overall string power and voltage is constant during the entire duration.
Figure 3.12: Simulation results for series-cascaded microgrid Scenario A showing DGs and microgrid (a) Power factor (b) Reactive power output for unity power factor operation

Figure 3.13: Phasor diagram for series-cascaded microgrid Scenario A (a) MPPT=0.15 kW (b) MPPT=0.3 kW (c) MPPT=0.45 kW for unity power factor operation.
Figure 3.14: Simulation results for series-cascaded microgrid scenario A showing DGs and microgrid (a) active power and (b) voltage output with actual irradiance profile
<table>
<thead>
<tr>
<th>Description</th>
<th>Label</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Master DG</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DC-link voltage</td>
<td>$V_{DC}$</td>
<td>200V</td>
</tr>
<tr>
<td>Filter Inductor</td>
<td>$L_f$</td>
<td>5 [mH]</td>
</tr>
<tr>
<td>Filter Capacitor</td>
<td>$C_f$</td>
<td>22 [uF]</td>
</tr>
<tr>
<td>Current controller proportional gain</td>
<td>$K_{p,c}$</td>
<td>0.09</td>
</tr>
<tr>
<td>Current controller integral gain</td>
<td>$K_{i,c}$</td>
<td>45</td>
</tr>
<tr>
<td>voltage controller proportional gain</td>
<td>$K_{p,v}$</td>
<td>0.009</td>
</tr>
<tr>
<td>Voltage controller integral gain</td>
<td>$K_{i,v}$</td>
<td>9</td>
</tr>
<tr>
<td><strong>Slave DG</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DC-link voltage</td>
<td>$V_{DC}$</td>
<td>105V</td>
</tr>
<tr>
<td>Filter Inductor</td>
<td>$L_f$</td>
<td>5 [mH]</td>
</tr>
<tr>
<td>Filter Capacitor</td>
<td>$C_f$</td>
<td>22 [uF]</td>
</tr>
<tr>
<td>Current controller proportional gain</td>
<td>$K_{p,c}$</td>
<td>0.09</td>
</tr>
<tr>
<td>Current controller integral gain</td>
<td>$K_{i,c}$</td>
<td>45</td>
</tr>
<tr>
<td>voltage controller proportional gain</td>
<td>$K_{p,v}$</td>
<td>0.009</td>
</tr>
<tr>
<td>Voltage controller integral gain</td>
<td>$K_{i,v}$</td>
<td>9</td>
</tr>
<tr>
<td><strong>Power factor regulator</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proportional gain</td>
<td>$K_{p,pf}$</td>
<td>0.08</td>
</tr>
<tr>
<td>Integral gain</td>
<td>$K_{i,pf}$</td>
<td>0.00002</td>
</tr>
<tr>
<td><strong>Droop gains</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output voltage (RMS)</td>
<td>$V_{rms}$</td>
<td>120V</td>
</tr>
<tr>
<td>Frequency</td>
<td>$f$</td>
<td>50±2% [Hz]</td>
</tr>
<tr>
<td>Power factor droop gain for $pf - f$</td>
<td>$M_{pf}$</td>
<td>2</td>
</tr>
<tr>
<td>Active power droop gain $P - f$</td>
<td>$M_P$</td>
<td>0.008</td>
</tr>
</tbody>
</table>
Scenario B: Active Power vs. Frequency (P-ω) droop with priority dispatch

In this scenario, the simulation setup is considered to be same as Figure 3.6.a and microgrid load is set to 0.9 kW, and the simulation parameters are the same as given in Table 3.1. The simulation results are shown in Figure 3.15 and Figure 3.16. Figure 3.15.a show the active power sharing performance for the proposed system. Slave DG#2 and #3 track the MPPT power set-point of 0.15 kW for $t \in [0, 1]$ sec and master DG#1 operates at 0.6kW so as to satisfy the overall string power demand of 0.9 kW. For $t \in [1, 2]$ sec, slave DG#2 and #3 tracks the increased MPPT set-point of

Figure 3.15: Simulation results for series-cascaded microgrid Scenario B showing DGs and microgrid (a) active power and (b) reactive power output, and (c) microgrid frequency
0.3 kW and correspondingly master DG#1 active power sharing drops down to 0.3 kW. For \( t \in [2, 3] \) sec, the MPPT power set-point is increased to 0.45 kW and the master DG output power drops down to zero and the slave DGs active power is curtailed at 0.45 kW. Since master DG#1 has low active power demand, the reactive power responsibility has been autonomously prioritised, according to the prioritization method given in section 3.2.2. Therefore, the reactive power output of slave DG#2 and #3 drops to zero and the master DG#1 contributes to the entire reactive power demand by sharing 0.46 kVAr as shown on Figure 3.15.b.

Figure 3.15.c illustrates the microgrid frequency which is drooped according to the defined droop lines of the master DG, as explained in section 3.2.2. For \( t \in [0, 1] \) sec, master DG#1 share 0.75 kW where the corresponding microgrid frequency is 49.8Hz. When the MPPT set-point is increased to 0.30 kW for \( t \in [1, 2] \) sec, the master DG power output reduces to 0.60 kW, and thereby, the microgrid frequency is increased to 50.4Hz. For \( t \in [2, 3] \) sec, active power output of master DG#1 drops to zero and hence the microgrid frequency is at its maximum value i.e., 51Hz.

![Diagram](image-url)

Figure 3.16: Simulation results for series-cascaded microgrid Scenario B showing DGs and microgrid (a) operating power factor (b) reactive power output
Figure 3.16.a show the power factor sharing performance in which the slave DG#2,#3 operates at power factor of 0.57 for \( t \in [0, 1] \text{sec} \), 0.92 for \( t \in [1, 2] \text{sec} \) and unity for \( t \in [2, 3] \text{sec} \). Accordingly the master DG#1 operates at unity pf for \( t \in [0, 1] \text{sec} \), and reduces after that operating at 0.86 for \( t \in [1, 2] \text{sec} \) and zero pf for \( t \in [2, 3] \text{sec} \), when MPPT is at maximum Figure 3.16.b depicts the voltage sharing performance in which slave DG#2 and #3 adjust its operating voltage to 33.23 V for \( t \in [0, 1] \text{sec} \), 40 V for \( t \in [1, 2] \text{sec} \) and 54 V for \( t \in [2, 3] \text{sec} \) so as to track the varying MPPT profile. Accordingly, the master DG# operates at 68 V for \( t \in [0, 1] \text{sec} \), to 40 V for \( t \in [1, 2] \text{sec} \) and 39 V for \( t \in [2, 3] \text{sec} \) to maintain overall microgrid voltage of 120 V.

3.4. Experimental Verification

The proposed power factor vs. frequency (pf-\( \omega \)) droop method described in section 3.2.1 has been extensively evaluated using time-based switched simulations and matching experimental investigations. An experimental cascaded microgrid comprises two DGs as shown in Figure 3.17. The microgrid is set to operate at 0.7 kW and 0.82 power factor and the system performance is validated for varying MPPT set points under different modes of operation. The detailed experimental setup is explained in Chapter 6 and the system parameters are given in Table 3.2.
Figure 3.18 shows the experimental and matching simulation results for the considered series cascaded microgrid. The slave DG tracks the MPPT set-point power of 0.20 kW for \( t \in [0, 5] \) sec, 0.25 kW for \( t \in [5, 10] \) sec, and 0.35 kW for \( t \in [10, 15] \) sec. Corresponding to every incremental step in MPPT, the master DG reduces its active power output so as to maintain power balance in the microgrid.

**Synchronisation mode:** Figure 3.19 demonstrates the performance of the proposed control for synchronisation mode where the cascaded DG, as expected, operates at the same power factor as the master DG and load, i.e., 0.82. In this mode, the slave DG autonomously adjust its reactive power and pf reference (according to the methodology given in 3.2.1) to attain synchronisation.

The experimental results for the DGs and overall microgrid instantaneous voltages are shown in Figure 3.20. The DGs voltage is synchronised under all MPPT operating conditions of MPPT, as shown in the zoomed figures.

**Power factor dispatch mode:** Figure 3.21 illustrates the system performance for power factor sharing mode of operation where the slave DG is commanded with a reference power factor of 0.9. From the simulation and experimental results, it can be seen that slave DG#2 follows the targeted power factor of 0.90. Corresponding reactive power output is given in Figure 3.22 in comparison with the simulation results.

**Power factor dispatch mode with unity:** Figure 3.23 and Figure 3.24 demonstrates the system performance when the slave DG is commanded to operate at unity power factor. As shown in Figure 3.23, the slave DGs are operating at unity power factor, and reactive power demand is satisfied by the master DG the corresponding reactive power output are also shown in Figure 3.24.
# Chapter 3. Series-Cascaded Microgrid with Non Dispatchables

## Table 3.2: Experimental System Parameters for Series-Cascaded Microgrid.

<table>
<thead>
<tr>
<th>Description</th>
<th>Label</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Master DG</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DC-link voltage</td>
<td>$V_{DC}$</td>
<td>250V</td>
</tr>
<tr>
<td>Filter Inductor</td>
<td>$L_f$</td>
<td>4.4 [mH]</td>
</tr>
<tr>
<td>Filter Capacitor</td>
<td>$C_f$</td>
<td>40 [uF]</td>
</tr>
<tr>
<td>Current controller proportional gain</td>
<td>$K_{p,c}$</td>
<td>0.1</td>
</tr>
<tr>
<td>Current controller integral gain</td>
<td>$K_{i,c}$</td>
<td>100</td>
</tr>
<tr>
<td>Voltage controller proportional gain</td>
<td>$K_{p,v}$</td>
<td>0.35</td>
</tr>
<tr>
<td>Voltage controller integral gain</td>
<td>$K_{i,v}$</td>
<td>3.88</td>
</tr>
<tr>
<td><strong>Slave DG</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DC-link voltage</td>
<td>$V_{DC}$</td>
<td>125V</td>
</tr>
<tr>
<td>Filter Inductor</td>
<td>$L_f$</td>
<td>4.4 [mH]</td>
</tr>
<tr>
<td>Filter Capacitor</td>
<td>$C_f$</td>
<td>40 [uF]</td>
</tr>
<tr>
<td>Current controller proportional gain</td>
<td>$K_{p,c}$</td>
<td>0.17</td>
</tr>
<tr>
<td>Current controller integral gain</td>
<td>$K_{i,c}$</td>
<td>170</td>
</tr>
<tr>
<td>Voltage controller proportional gain</td>
<td>$K_{p,v}$</td>
<td>0.01</td>
</tr>
<tr>
<td>Voltage controller integral gain</td>
<td>$K_{i,v}$</td>
<td>200</td>
</tr>
<tr>
<td><strong>Power factor regulator</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proportional gain</td>
<td>$K_{p,pf}$</td>
<td>0.1</td>
</tr>
<tr>
<td>Integral gain</td>
<td>$K_{i,pf}$</td>
<td>0.001</td>
</tr>
<tr>
<td><strong>Droop gains</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output voltage (RMS)</td>
<td>$V_{rms}$</td>
<td>120V</td>
</tr>
<tr>
<td>Frequency</td>
<td>$f$</td>
<td>50±2% [Hz]</td>
</tr>
<tr>
<td>Power factor droop gain for $pf - f$</td>
<td>$M_{pf}$</td>
<td>2</td>
</tr>
<tr>
<td>Active power droop gain $P - f$</td>
<td>$M_p$</td>
<td>0.008</td>
</tr>
</tbody>
</table>


Figure 3.18: Validation results showing active power output of the DGs
(a) Experimental (b) Simulation
Figure 3.19: Validation results showing DGs operating power factor for synchronization mode (a) Experimental (b) Simulation

Figure 3.20: Experimental results showing the instantaneous voltage out of the DGs and microgrid for synchronisation mode of operation
Figure 3.21: Experimental and simulation results showing operating pf of the slave DG for pf dispatch mode with commanded power factor of 0.9

Figure 3.22: Validation results showing the reactive power output of the DGs for pf dispatch mode (a) Simulation (b) Experimental
Chapter 3. Series-Cascaded Microgrid with Non Dispatchables

3.5. Frequency Response Analysis

In order to examine the stability of the controllers and to analyse the power sharing accuracy of the developed system, this section discuss the frequency response of the series-cascaded structure used in simulation and experiment.
Figure 3.25 shows the simplified control block diagram of the voltage source inverter with dual-loop control architecture which has been used in simulation as well experimental validation (Figure 3.17).

The open loop and closed loop transfer function of the system with current controller only is given as,

$$ G_{cc,OL} = \frac{I_L(s)}{I_{ref}(s)} = \frac{V_{dc}H_c(s)H_{PWM}(s)}{Z_L(s)} $$ \hspace{1cm} (3.17)

$$ G_{cc,CL} = \frac{I_L(s)}{I_{ref}(s)} = \frac{V_{dc}H_c(s)H_{PWM}(s)}{Z_L(s)} $$ \hspace{1cm} (3.18)

where $H_c(s)$ is the transfer function of the Proportional-Resonant (PR) based current controller,

$$ H_c(s) = \left( K_{p.c} + \frac{K_{i.c}s}{s^2 + \omega^2} \right) $$ \hspace{1cm} (3.19)

and $H_{PWM}(s)$ is the transportation delay:

$$ H_{PWM}(s) = \frac{1}{1+sT_d} $$ \hspace{1cm} (3.20)

$Z_L(s)$ is the filter inductor transfer function,

$$ Z_L(s) = sL_f $$ \hspace{1cm} (3.21)

The stability of the VSI control and closed-loop performance is verified using the parameters given in Table 3.2 and have been used for the experimental verification.

The open loop and closed-loop frequency response of the VSI with current controller is plotted in Figure 3.26. Open loop response graph as shown in Figure
3.26. a indicates that the system has positive gain margin of 5.71 dB at 1.6 kHz and positive phase margin of $40^\circ$ at 837 Hz.

Figure 3.26. b shows the current controller closed-loop response which indicates current tracking performance of the current controller. The system has zero gain and phase across the desired range of frequency.

In order to analyse the performance and confirm stable behaviour of VSI with dual loop control structure, an open loop and closed loop forward path gain is derived using Mason’s gain formula as,

$$G_{r,ol} = \frac{V_c(s)}{V_{ref}(s)} = \frac{V_{dc}H_v(s)H_c(s)H_{PWM}(s)Z_c(s)}{Z_L + V_{dc}H_c(s)H_{PWM}(s)}$$

(3.22)

$$G_{r,ol} = \frac{V_c(s)}{V_{ref}(s)} = \frac{H_v(s)H_c(s)H_{PWM}(s)\left(\frac{Z_L(s)}{Z_c(s)}\right)}{1 + \left(H_v(s)H_c(s)H_{PWM}(s)\left(\frac{Z_L(s)}{Z_c(s)}\right)\right) + \left[H_v(s)H_{PWM}(s)\left(\frac{1}{Z_L(s)}\right)\right] + \left(\frac{Z_c(s)}{Z_c(s)}\right)}$$

(3.23)

where $H_v(s)$ is the PR based voltage controller defined as

$$H_v(s) = \left(\frac{K_p}{s^2 + \omega^2}\right)$$

(3.24)

The capacitor impedance $Z_c(s)$ is defined as

$$Z_c = \frac{1}{sC_f}$$

(3.25)

Figure 3.27. shows the open-loop and closed-loop frequency response of the voltage controller with dual loop control. Figure 3.27. a. shows the open loop response with gain margin of 18.2 dB at 945 Hz and phase margin of $42.3^\circ$ at 110 Hz. The closed-loop response is shown in Figure 3.27. b where it shows a good tracking response over the operating frequency range of 49-51 Hz.
Figure 3.26: Voltage Source Inverter (VSI) with PR current controller (a) Open loop frequency response (b) Closed-loop frequency response
Figure 3.27: Voltage Source Inverter (VSI) with PR voltage controller (a) Open loop frequency response (b) Closed-loop frequency response
3.6. Summary

This chapter has presented a series-cascaded microgrid topology integrating dispatchable and non-dispatchable DGs. Decentralised master-slave control strategy based on two different droop schemes has been developed for power management in the series-cascaded microgrid system. The first method based on pf-ω droop can ensure different operation for the slave DG units, whereas the second method based on P-ω droop can prioritise the reactive power sharing based on MPPT power.

Simulation results have demonstrated that while operating in master-slave mode, the dispatchable and non-dispatchable DGs have achieved effective power balancing and constant voltage regulation in a fully decentralised manner. The proposed method also has demonstrated an effective utilisation of the non-dispatchable units along with the provision of appropriate power curtailment under light load conditions.

The performance of proposed series-cascaded microgrid topology and associated control schemes has been verified through extensive simulation investigations for different operating conditions. Compared to the traditional parallel-topology microgrid, the series-cascaded microgrid proposed in this paper offers more flexibility in setting the operating voltage level of the non-dispatchable DGs without compromising the microgrid operational performance and supply reliability. The master slave control strategy based on pf-ω droop is further validated by extensive matching simulation and experimental results.

The stability and tracking performance of the controllers used in the setup is verified through frequency response analysis. The results indicate that with proposed control strategy, both voltage and current controllers have enough gain and phase margin to ensure stability of the system and can offer good tracking performance over desired range of operating frequencies.
Chapter 4  
Hybrid Microgrid  

This chapter develops a hybrid topology microgrid that integrates dispatchable and non-dispatchable DGs in series-parallel arrangement. The microgrid is formed by parallel-connected string units where each string unit comprises a set of series-cascaded dispatchable DG and multiple low-voltage non-dispatchable DGs. The proposed hybrid topology offers the benefits of using low-gain DC-DC power conversion stages because of the cascaded topology, and high redundancy offered by the parallel topology. As the power regulation principle for series-cascaded and paralleled DGs are contradictory to each other, the existing control schemes for individual parallel and series-cascaded microgrids cannot be applied to the proposed hybrid microgrid, and therefore a new control scheme is developed to achieve autonomous power management in the proposed hybrid microgrid.

4.1. Hybrid Microgrid Architecture  

Traditional microgrids are formed by paralleled DGs and load, with droop control being often employed to achieve autonomous power sharing among the dispatchable DGs. However, DGs in parallel topology microgrid are required to have a high DC-link voltage at the input-side of inverter for an adequate synthesis of the microgrid nominal voltage. This usually demands a high-gain DC-DC conversion stage, particularly for DGs with low rated dc voltage source such as solar PVs or fuel-cells.

The material in this chapter has been accepted for publication in ICPES, Perth, Australia 2019:
The series-cascaded topology microgrids provide a more efficient and flexible alternative to integrate low-voltage DGs to medium voltage networks. In a series topology microgrid, DGs are cascaded in series and their terminal voltages are summed together to achieve the user level or microgrid nominal voltage, as described in section 2.2.2. Consequently, unlike the parallel topology, DGs of a series topology microgrid can operate with a lower dc-link voltage, and this eliminates the need for high-gain DC-DC conversion stages. Moreover, all series-cascaded DGs share the same line current, therefore the circulating harmonic current issue is inherently managed. Despite these benefits, the cascaded topology is prone to single point failure.

The concept of hybrid microgrid was developed with the aim of exploiting the benefits of both parallel and cascaded topologies. However, the DGs exhibit contradicting power regulation principle in series and parallel topologies and existing control schemes for parallel or series structure cannot be applied to the hybrid structure in its original form. In addition, the possibility of integrating non-dispatchable DGs into hybrid structure has not been not addressed in the literature till now.

4.2. Proposed Hybrid Microgrid Architecture

Figure 4.1. Shows the proposed hybrid microgrid architecture in islanded mode of operation. The system consists of two parallel string units, where each string is formed by a series of dispatchable, referred as “master”, and non-dispatchable, referred as “slaves”, DGs. The master DG is rated at the overall microgrid nominal voltage, while the slave DGs have lower nominal output voltages. The operational performance matrix for the proposed microgrid is defined as follow:

*Power Sharing at String Level*: The paralleled string units (#1 and #2) should exhibit a power sharing performance identical to the parallel topology microgrid, as discussed in section Figure 4.1. More specifically, the string units should: i) regulate the string voltage magnitude and frequency within a permissible range defined by the
droop relation, and ii) produce active and reactive power proportional to their respective power ratings.

**Power Generation at DG Level:** The non-dispatchable DGs in each string should track their MPPT power reference and operate at the string or microgrid power factor. Since the proposed system is developed considering the absence of energy storage elements, active power curtailment should be automatically activated during surplus power conditions, i.e. when the sum of MPPT reference power of the non-dispatchable DGs exceeds the load power demand.

**Autonomous Control:** The control strategy should be simple and independent of communication links. To achieve this performance matrix and control features, a master-slave hierarchical scheme is proposed, as detailed in the following section.
4.3. Master-Slave Control Strategy for Proposed Hybrid Microgrid

In order to satisfy the operational constraints raised by parallel and cascaded DGs, a master-slave control strategy has been proposed. In proposed control strategy, the dispatchable DG facilitates power sharing among the paralleled string units and also provides frequency reference to cascaded DGs within respective string among other tasks as detailed below.

4.3.1. Master DG

The dispatchable master DGs are responsible for regulating the microgrid voltage and frequency and for ensuring proportional power sharing among the string units. In order to avoid circulating currents within the paralleled string units, and to achieve proportional power sharing, conventional droop theory is applied to the master DG in each string, as indicated on the control block diagram shown in Figure 4.2, viz.:

\[ P_{j} = \frac{\omega_{\text{ref}} - \omega_{\text{min}}}{\omega_{\text{max}} - \omega_{\text{min}}} P_{\text{max},j} \]

\[ Q_{j} = \frac{V_{\text{ref},j} - V_{\text{min}}}{V_{\text{max}} - V_{\text{min}}} Q_{\text{max},j} \]

Figure 4.2: Master DG in hybrid topology microgrid (a) Control block diagram (b) Active power droop curve (c) Reactive power droop curve.
\[ \omega_{\text{ref}} = \omega_{\text{max}} - M_p \times P_j \]  
\[ V_{\text{ref},j_1} = V_{\text{max}} - M_q \times Q_j \]  
where \( j = 1 \text{or} 2 \) represents the string number, \( \omega_{\text{ref}} \) and \( V_{\text{ref},j_1} \) are the reference frequency and voltage of the master DG (#11 or #21), \( \omega_{\text{max}} \) and \( V_{\text{max}} \) are the frequency and voltage of the string or microgrid at no-load, and \( P_j \) and \( Q_j \) are the active and reactive power output of the string units.

The reference voltage for master DG can be defined as
\[ v_{\text{ref},j_1}(t) = V_{\text{ref},j_1} \sin(\omega_{\text{ref}}t) \]  
In addition, Master DGs need to compensate for the power variations of the non-dispatchable slave DGs, and as such are responsible for regulating the overall string voltage \( V_j \) instead of their terminal voltage \( V_{j,1} \).

Accordingly, master DG’s reference voltage is tracked via the dual loop control architecture as shown in Figure 4.2, and discussed in section 2.1 via,
\[ i_{\text{ref},j_1} = H_v(s)(v_{\text{ref},j_1} - v_j) = \left( K_{p,v} + \frac{K_{i,v}s}{s^2 + \omega^2} \right)\left( v_{\text{ref},j_1} - v_j \right) \]  
\[ V_{\text{PWM},j_1} = H_c(s)(i_{\text{ref},j_1} - i_1) = \left( K_{p,c} + \frac{K_{i,c}s}{s^2 + \omega^2} \right)\left( i_{\text{ref},j_1} - i_1 \right) \]  
where, \( H_v(s) \) and \( H_c(s) \) are the PR based voltage and current controllers tuned to fundamental frequency so as to achieve zero steady-state error for the voltage \( v_{\text{ref},j_1} - v_j = 0 \) and current reference, \( i_{\text{ref},j_1} - i_1 = 0 \). \( K_{p,v} \) and \( K_{i,v} \) are the proportional gain and integral gain for master DG voltage controller and \( K_{p,c} \), \( K_{i,c} \) are the proportional gain and integral gain for master DG current controller. \( \omega \) is the fundamental frequency of 50 Hz.

Note that the voltage controller defined by (4.4) for master unit tracks the string voltage instead of its own terminal voltage. This will ensure constant microgrid voltage irrespective of the voltage variation due to slave DGs, i.e., non-dispatchables.
4.3.2. Slave DG

The slave DGs are tasked to follow their respective MPPT reference for active power dispatch, and microgrid or string power factor (pf) for reactive power dispatch, as shown in Figure 4.3. The MPPT reference tracking of the slave DGs is achieved by defining their individual operating voltage proportional to their MPPT reference power, viz.:

$$V_{\text{ref},jn} = \frac{P_{\text{MPPT},jn}}{I_{jn} \times pf_{\text{ref},jn}} \quad (4.6)$$

Where $i = 1 \text{ or } 2$ represent the string number, and $n = 2, 3, 4,...$ represent the slave DG number, $V_{\text{ref},jn}$ is the reference voltage of the slave DGs, $I_{jn}$ is the slave DG or string current, that can be measured locally at the slave DG terminal, $P_{\text{MPPT},jn}$ is the slave DG MPPT reference power, and $pf_{\text{ref},jn} \approx pf_{j}$ is the reference pf of the microgrid which is calculated based on the overall string power demand. The string power demand can be estimated locally using the reverse droop relation, viz.:

Figure 4.3: Control block diagram of the Slave DGs in hybrid topology microgrid
\[ P_{\text{ext},j} = \frac{\omega^* - \omega_{jn}}{m_{p,j}} \]  
\[ (4.7) \]
\[ pf_{\text{ref},jn} = \frac{P_{\text{ext},j}}{V_{\text{nom},j} \times I_{jn}} \]  
\[ (4.8) \]

Since all slave DGs need to operate at equal pf for reactive power support, they need to track the reference pf defined by (4.8). This is achieved via a power factor regulator given by,

\[ \theta_{\text{ref},jn} = \left( K_{p,pf} + \frac{K_{i,pf}}{s} \right) \left( pf_{\text{ref},jn} - pf_{\text{ext},jn} \right) \]  
\[ (4.9) \]

where, \( K_{p,pf} \) and \( K_{i,pf} \) are the proportional and integral gains of power factor regulator. The power factor regulator generates a compensation angle \( \theta_{\text{ref},jn} \), that is then added to the slave DGs voltage reference, viz.:

\[ v_{jn}(t) = V_{\text{ref},jn} \sin(\omega_{\text{ref},j} t + \theta_{\text{ref},jn}) \]  
\[ (4.10) \]

where \( \omega_{\text{ref},j} \) is the microgrid frequency, measured locally at the terminals of the slave DGs. This voltage reference is again tracked using the typical dual loop control architecture, as shown in Figure 4.3.

\[ i_{\text{ref},jn} = H_v(s)(v_{\text{ref},jn} - v_{jn}) = \left( K_{p,v} + \frac{K_{i,v} s}{s^2 + \omega^2} \right)(v_{\text{ref},jn} - v_{jn}) \]  
\[ (4.11) \]

\[ V_{\text{PWM},jn} = H_c(s)(i_{\text{ref},jn} - i_{jn}) = \left( K_{p,i} + \frac{K_{i,i} s}{s^2 + \omega^2} \right)(i_{\text{ref},jn} - i_{jn}) \]  
\[ (4.12) \]

In (4.11) and (4.12), \( i_{\text{ref},jn} \) and \( i_{jn} \) are the slave DG reference current and output current, respectively, \( v_{\text{ref},jn} \) and \( v_{jn} \) are the slave DG reference and terminal voltage, respectively, and \( V_{\text{PWM},jn} \) is the PWM reference voltage used to synthesize the switching signal.

**Slave DGs Power Curtailment:** In the absence of storage system, the slave DGs power need to be curtailed under certain operating conditions, i.e. when the slave DGs’ MPPT power is larger than the microgrid load demand. The curtailed maximum
power reference of the slave DGs can be calculated based on the real-time overall string power demand, viz.:

\[
P_{\text{max},jn} = \left( \frac{P_{\text{nom},jn}}{P_{\text{nom},j}} \right) P_{\text{est},j}
\]  \hspace{1cm} (4.13)

where \( P_{\text{nom},jn} \) and \( P_{\text{nom},j} \) are the rated power of the slave DGs and string unit, respectively, \( P_{\text{est},j} \) is the string power estimated using (4.7). The calculated maximum power \( P_{\text{max},jn} \) constraint is then included in the slave inverter’s control block diagram, as shown in Figure 4.3.

### 4.4. Performance Validation

The performance of the proposed hybrid microgrid and control strategy are validated using PSIM simulations only. As mentioned earlier, the hardware validation for the hybrid microgrid is not done due to limited resources. The considered hybrid microgrid is rated at 3 \( kVA \), 120\( V_{\text{rms}} \), and the control and system parameters are
given in Table 4.1. A complete description of the hybrid microgrid simulated in this section is given in Chapter 5. The system power sharing, voltage and frequency regulation performance is validated for two different scenarios of hybrid microgrid.

**Table 4.1: Performance Validation Parameters for Hybrid Microgrid.**

<table>
<thead>
<tr>
<th>Description</th>
<th>Label</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Master DG</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DC-link voltage</td>
<td>$V_{DC}$</td>
<td>200V</td>
</tr>
<tr>
<td>Filter Inductor</td>
<td>$L_f$</td>
<td>2 [mH]</td>
</tr>
<tr>
<td>Filter Capacitor</td>
<td>$C_f$</td>
<td>45 [uF]</td>
</tr>
<tr>
<td>Current controller proportional gain</td>
<td>$K_{p,c}$</td>
<td>0.1</td>
</tr>
<tr>
<td>Current controller integral gain</td>
<td>$K_{i,c}$</td>
<td>100</td>
</tr>
<tr>
<td>Voltage controller proportional gain</td>
<td>$K_{p,v}$</td>
<td>0.35</td>
</tr>
<tr>
<td>Voltage controller integral gain</td>
<td>$K_{i,v}$</td>
<td>3.88</td>
</tr>
<tr>
<td><strong>Slave DG</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DC-link voltage</td>
<td>$V_{DC}$</td>
<td>100V</td>
</tr>
<tr>
<td>Filter Inductor</td>
<td>$L_f$</td>
<td>2 [mH]</td>
</tr>
<tr>
<td>Filter Capacitor</td>
<td>$C_f$</td>
<td>45 [uF]</td>
</tr>
<tr>
<td>Current controller proportional gain</td>
<td>$K_{p,c}$</td>
<td>0.17</td>
</tr>
<tr>
<td>Current controller integral gain</td>
<td>$K_{i,c}$</td>
<td>170</td>
</tr>
<tr>
<td>Voltage controller proportional gain</td>
<td>$K_{p,v}$</td>
<td>0.01</td>
</tr>
<tr>
<td>Voltage controller integral gain</td>
<td>$K_{i,v}$</td>
<td>200</td>
</tr>
<tr>
<td><strong>Power factor regulator</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proportional gain</td>
<td>$K_{pf}$</td>
<td>0.1</td>
</tr>
<tr>
<td>Integral gain</td>
<td>$K_{i, pf}$</td>
<td>0.001</td>
</tr>
<tr>
<td><strong>Droop gains</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output voltage (RMS)</td>
<td>$V_{rms}$</td>
<td>120±5% [V]</td>
</tr>
<tr>
<td>Frequency</td>
<td>$f$</td>
<td>50±2% [Hz]</td>
</tr>
<tr>
<td>Active power droop gain</td>
<td>$M_p$</td>
<td>0.0013</td>
</tr>
<tr>
<td>Reactive power droop gain</td>
<td>$M_q$</td>
<td>0.008</td>
</tr>
</tbody>
</table>
**Scenario A: Equal rating string units and unequal feeder impedance.**

In this scenario, the microgrid comprises two string units, as shown in Figure 4.4. String#1 consists of one dispatchable master (DG#11) and two non-dispatchable slaves (DG#12 and DG#13), while String#2 consists of one dispatchable master (DG#21) and one non-dispatchable slave (DG#22).

The system performance is validated under various load and MPPT operating conditions. The microgrid load is set to 2.5 kW, 0.9 pf, and the MPPT power reference of String#1 slave DGs are set to 0.25 kW for \( t \in [0, 1] \) sec, 0.5 kW for \( t \in [1, 2] \) sec, and 0.75 kW for \( t \in [2, 3] \) sec, while the MPPT power reference of String#2 slave DGs are set to 0.5 kW for \( t \in [0, 1] \) sec, 1 kW for \( t \in [1, 2] \) sec, and 1.5 kW for \( t \in [2, 3] \) sec.

**String Units Power Sharing:** Simulation results for the active and reactive power of the DGs, and microgrid voltage and frequency, are provided in Figure 4.5. The microgrid load is equally shared between both string units since they are equally rated. From Figure 4.5.a and Figure 4.5.b, String#1 and #2 deliver 1.25 kW and \( \sim 0.6 \) kVAr at the three different MPPT setpoints. Accordingly, the microgrid voltage and frequency deviates from its nominal value to 49.34 Hz and 120.94 Vrms, respectively, which mitigates the droop principle explained for parallel topology microgrid in section 2.2.1. The power sharing performance of the proposed hybrid microgrid is comparable the conventional droop-controlled parallel topology microgrid, where real and reactive power sharing are achieved by drooping the voltage and frequency of the DGs.

**Series DG Active Power Output:** The active power produced by the cascaded DGs in String#1 are shown in Figure 4.5.b. From this figure, the slave DGs track the MPPT power reference of 0.25 kW for \( t \in [0, 1] \) sec and 0.5 kW for \( t \in [1, 2] \) sec. The increase in power produced by the slave DGs is followed by a reduction in active power output of the master DG, who is responsible for the overall power balance within the string and for ensuring proportional power sharing among the strings. Accordingly, the master DGs delivers 0.75 kW for \( t \in [0, 1] \) sec and 0.25 kW for \( t \in [1, 2] \) sec. From 2 sec onwards, the MPPT power reference for the slaves DG#12
and #13 is set to 0.75 kW/unit, exceeding the overall string power demand of 1.25 kW. Therefore, the active power for the slaves DG#12 and DG#13 are curtailed at 0.625 kW and the master DG#11 commanded active power drops down to zero as the entire power demand is fulfilled by the two solar PV slave DGs. Similar power sharing performance is achieved for String#2, as shown in Figure 4.5.c where the MPPT set points are changed in 0.5 kW steps. From this figure, slave DG#22 tracks the MPPT set points till 2 sec and power curtailment at 1.25 kW occurs from 2 sec onwards, while master DG#21 satisfies the overall power demand and produces 0.75 kW for \( t \in [0, 1] \) sec, 0.25 kW for \( t \in [1, 2] \) sec, and ~0 kW for 2 sec onwards.

**Series DGs Reactive Power Output:** The reactive power produced by the DGs in String #1 and #2 are based on the principle of equal power factor (pf) operation. As shown in Figure 4.6.a the slave DGs #12 and #13 share 0.115 kVAr while produce 0.25 kW for \( t \in [0, 1] \) sec, and 0.22 kVAr while produce 0.5 kW for \( t \in [1, 2] \) sec, so as to maintain constant P/Q ratio, i.e. pf synchronisation. The master DG#11 compensates for the reactive power shortfall by producing 0.37 kVAr for \( t \in [0, 1] \) sec and 0.15 kVAr for \( t \in [1, 2] \) sec. Similar reactive power sharing performance is achieved for the DGs in String#2, as shown in Figure 4.6.b.

As discussed in section 4.3, the power output of the cascaded DGs is controlled by regulating their operating voltage instead of operating current as for the parallel topology microgrid. This performance is illustrated in Figure 4.6.c for String#1. Up to 1 sec, the operating voltage of master DG#11 is 78.76 Vrms and of slave DG#12 and #13 is 21.09 Vrms and the overall string voltage is 120.94 Vrms. From 1 sec onwards, the operating voltage of DGs change with their output power, while the overall microgrid voltage is maintained within a narrow range (120 Vrms ± 5%), following the droop principle for parallel topology microgrid.

Further evaluation of the system performance is conducted by considering the MPPT set points rescaled from the actual irradiance profile. Figure 8 shows the power output of the DGs in String#1, where the slave DGs follow the MPPT reference power and the master DG compensates for the shortfall. The power curtailment of slave DGs is activated for a short time period, i.e. \( t \in [2.1, 2.4] \) sec, while the power
Figure 4.5: Simulation results for hybrid microgrid Scenario A showing (a) microgrid frequency, DGs and string units active power output for (b) String#1 (c) String#2 demand is solely met by the slave DGs. Note that while the power output of the string DGs changes continuously, the overall string power is considerably constant and shared proportionally according to the respective DG power rating, as shown in Figure 4.7.a and Figure 4.7.b. The corresponding output voltage of DGs in String #1 is also shown in Figure 4.7.c and Figure 4.7.d, where it can be seen that the overall string voltage is constant for the entire operating range. Note that there is a slight error in reactive power sharing and voltage sharing performance at string level due to the effect of unequal line impedances.
Figure 4.6: Simulation results for hybrid Scenario A, showing, DGs and string units reactive power output for (a) String#1 and (b) String#2, and DGs and string units operating voltage for (c) String#1 and (d) String#2.
Figure 4.7: Simulation results for hybrid microgrid Scenario A showing, DGs and string units active power output for (a) String#1 and (b) String#2 (c) DGs and string units operating voltage for (c) String#1 and (d) String#2, with actual irradiance profile.
Scenario B: Unequal rating string units and unequal line impedance

The system architecture for the second scenario is shown in Figure 4.8.a where the microgrid is formed by string units of unequal ratings. The system parameters are the same as given above in Table 4.1, except the droop gain for DG#21 which is now twice the prior value. Microgrid load is set at 1.9 kW, 0.707 pf and the MPPT is kept constant at 0.5 kW and the corresponding simulation results are shown on Figure 4.9.

The active power sharing performance with the proposed master-slave control is shown in Figure 4.9.b where it can be seen that the active power shared by the string units is proportional to their power ratings. Specifically, string#1 unit with twice the power capacity than string#2 unit produces 1.35 kW and string#2 produces 0.65 kW. The corresponding microgrid frequency according to the droop curves is 49.25Hz and this result is shown in Figure 4.9.a. The reactive power sharing performance is illustrated in Figure 4.9.d where string#1 share 1.34kVAr and string#2 share 0.6 kVAr. At these reactive power loadings, the expected microgrid voltage is 115.28V, as shown in Figure 4.9.c.

Figure 4.8: Simulation setup for hybrid microgrid Scenario B (a) Simulation setup (b) Active power droop curves (c) Reactive power droop curves
Figure 4.9: Simulation results for hybrid microgrid Scenario B, showing (a) microgrid frequency (b) String and microgrid active power, (c) microgrid voltage, and (d) String and microgrid reactive power.
4.5. Summary

This chapter has presented a new hybrid topology microgrid that exploits the benefits of both series and conventional parallel topologies. The proposed microgrid provides flexibility to integrate non-dispatchable DGs with lower rated dc-link voltages, while also provides the redundancy and reliability offered by the parallel topology. A fully decentralised master-slave control strategy has been developed that effectively coordinates the power sharing between the paralleled string units and achieves power balance among the cascaded DGs within the strings. The proposed control strategy ensures proportional power sharing among the string units, and satisfies the operational constraints set by the non-dispatchable DGs such as MPPT tracking, PV curtailment, and reactive power support. The performance of the proposed microgrid and power management scheme, under various operating conditions, have been extensively verified using PSIM simulations.
Chapter 5
Description of the Simulation Systems

The validity of the proposed microgrids and control strategies presented in Chapter 3 and Chapter 4 is verified through simulation models developed in PSIM. This chapter provides an overview of the simulation platform and simulation systems from which the simulation results have been obtained.

5.1. PSIM

Powersim is a powerful simulation platform that offers comprehensive simulation and design capabilities for power electronics research and product development. The component models are relatively simple but sufficient for most electrical and electronics applications. In this section the building blocks of the simulation systems used in previous chapters are described in detail with the description of each individual system divided into power stage, modulation, and control structures.

5.1.1. Overall System Diagram

Figure 5.1 and Figure 5.2 show the power stage and control block diagram of the series cascaded microgrid implemented in PSIM software for performance validation in thesis section 3.3.1. Likewise, the power stage and control block diagram of the
hybrid microgrid implemented in PSIM software for performance validation in thesis section 4.4 are shown in Figure 5.3 and Figure 5.4.

Figure 5.1: Series-cascaded microgrid model power stage in PSIM
Figure 5.2: Series-cascaded microgrid control structure implemented in PSIM
Figure 5.3: Hybrid microgrid power stage in PSIM
Figure 5.4: Hybrid microgrid control structure implemented in PSIM
5.1.2. DG Power Stage

Figure 5.5 shows the DG power stage of one inverter among multiple similar inverters as shown in Fig. 5.1 and 5.3. The power stage comprises a single-phase H-Bridge structure with four IGBT switches, an input DC voltage source, \( V_{dc} \), representing a dispatchable energy source such as fuel-cell, and a low-pass inductive-capacitive (LC) filter at the output. An LC filter attenuates high frequency voltage and current harmonics that are present due to PWM switching. The inductor current \( I_o \) and capacitor voltage \( V_o \) feedback signals are measured using voltage and current sensors given in the PSIM library.

5.1.3. PWM Modulator

The switching signals for the IGBT switches are generated by a PWM modulator, as shown in Figure 5.6. The system receives modulation command \( V_{PWM} \) or \( M \) from the inner control loops, which is then compared with a periodic triangular waveform oscillating at usually high frequency, call as switching frequency \( f_{sw} \). The comparators generate on and off commands for the IGBT switches by comparing the fundamental frequency modulation signals with the high frequency carrier waveform.
Chapter 5. Description of the Simulation Systems

5.1.4. Inner Control Loops

The PWM modulator receives modulation signals from the inner current control loop. As discussed in section 2.1, 3.2 and 4.3, dual loop structure comprising cascaded voltage and current controllers, which are used to track inverter reference voltage and current. Figure 5.7.a illustrates the detailed implementation of PR based current controller used in the control. In this system, the inductor feedback current, $I_o$, is compared with the a reference current $I_{ref}$ to generate current error $I_{err}$. The PR controller tracks the current error to zero by producing a modulation signal $M$. An anti-windup circuitry is included to avoid over-modulation. The delay block represents transportation delay associated with digital implementation.

![Figure 5.6: Single phase PWM modulator in PSIM](image1)

(a)

![Figure 5.7: Inner control loop (a) Current controller (b) Voltage controller](image2)

(b)
Chapter 5. Description of the Simulation Systems

The current controller receives the reference current $I_{ref}$ from the voltage controller block shown on Figure 5.9.b. The voltage controller block compares the capacitor $V_o$ with reference voltage $V_{ref}$ and generates the voltage error signal, $V_{err}$. The voltage regulator tracks the reference voltage and the resultant output is the current reference signal $I_{ref}$, which is an input to the current controller.

5.1.5. Phase Locked Loop (PLL) Circuit

Figure 5.8 shows the PLL circuit developed for frequency measurement. The slave DG current $I_o$ is measured and transformed into synchronous frame, i.e., orthogonal components $I_{o\_alpha}$ and $I_{o\_beta}$ using Second-Order-Generalised-Integrator (SOGI)[64][65][66]. These orthogonal components are further transformed to stationery frame components $I_{o\_d}$ and $I_{o\_q}$ using a built in abc-dq transformation block in PSIM. The PLL loop filter drives the $I_{o\_q}$ component to zero and generates the frequency deviation from nominal value. This component is added to the fundamental frequency i.e., 50Hz to get the actual frequency.

Figure 5.8: Single phase PLL in PSIM

Figure 5.9: Single phase power calculation in PSIM
5.1.6. Power Calculation

The single phase power is calculated using FFT blocks in PSIM as shown in Figure 5.9. The FFT block outputs the peak magnitude and phase angle of the input signal, i.e., voltage and current, and then, active and reactive power is calculated as:

\[
P_v = 0.5 \times (V_{o, pk} I_{o, pk}) \cos(\theta_v - \theta_I) \quad \text{and} \quad Q_v = 0.5 \times (V_{o, pk} I_{o, pk} \sin(\theta_v - \theta_I))
\]

5.1.7. MPPT tracking and Power Factor Regulation

Figure 5.10.a shows the pf regulator implementation which compares the reference power factor, \( pf_{ref} \), with the actual power factor \( pf_{act} \). The built in PI controller in PSIM is tuned with appropriate gains so as to drive the error signal to zero and generates the compensation angle \( \theta_{comp} \).

Figure 5.10.b shows the MPPT to voltage calculation. The MPPT power is

Figure 5.11: Simulation results for series-cascaded microgrid showing MPPT tracking
varied in steps for $T_1$, $T_2$, and $T_3$ seconds and limiter at the output curtails the MPPT power to $P_{pv}$ which is then translated to operating voltage $V_{pv}$.

Figure 5.11 shows the MPPT tracking performance for simulation setup described in Chapter 3 and Chapter 4. The slave DG#2 and #3 tracks the commanded MPPT set points of 0.15 kW for $t \in [0, 1]$ sec and 0.30 kW for $t \in [1, 2]$ sec. However, for $t \in [2, 3]$ sec, the overall MPPT power reference exceeds the microgrid load demand and hence the active power is curtailed at 0.35 kW. With the curtailment active for $t \in [2, 3]$ sec, the entire microgrid/string load is met from slave DG#2 and DG#3, i.e., solar PV units, alone and master DG#1 power output drops down to zero. Further results and detailed explanations are provided on Chapter 3 and Chapter 4.

5.2. Summary

This chapter has provided details on the simulation platform and systems that have been used in this thesis to acquire verification of the proposed control strategies and ideas. The simulations are carried out in PSIM software, which is a discrete time-based program for solving electrical and electronic circuits.
Chapter 6
Description of the Experimental Systems

The performance of series-cascaded microgrid structure with master-slave control strategy presented in Chapter 3 is demonstrated through experimental investigations. This chapter explains the general-purpose hardware and software building blocks that have been used in these investigations.

6.1. Hardware Building Blocks

The power converter system employed in this thesis is essentially formed by a power stage structure which is operated by a DSP controller unit. A general-purpose power stage structure, designed and constructed by the Power and Energy Group at RMIT University, and a general-purpose controller unit, developed and built by the company Creative Power Technologies (CPT) [49], were utilised for this work.

The power stage for the series-cascaded structure is formed from elements of RMIT-U01 general purpose inverter board. The system incorporates eight 75 A / 600 V discrete TO-247 IGBTs arranged in a 4-phase leg structure rated at 2kVA. Figure 6.1 shows the simplified schematic diagram of this board. Terminals $T_9$ and $T_{10}$ provide connection for the incoming AC voltage followed with resistors $R_3$ and $R_4$ forming inrush current limiting stage. $K_1$ and $K_2$ are auxiliary relays, $D_1$ to $D_4$ are bridge rectifier diodes, $S_1$ to $S_4$ are power switches forming H-Bridge inverter, $C_1$ and $C_2$ are DC link capacitors, and two power terminals $T_1$ and $T_2$ for off-board DC-link output connection. Links $J_1$ and $J_2$ provide provision for
Figure 6.1: Simplified schematic diagram of the RMIT-U01 power stage board
operating the system as two independent H-Bridge converters. For this work, links $J_1$ and $J_2$ are removed and the series structure is formed by connecting $T_6$ to $T_3$ and the load is connected across $T_4$ and $T_5$.

The converters power stage was controlled by TMS320F2810 Digital Signal Processor (DSP) developed by Texas Instruments (TI) for power electronics control applications. This consist of an analog to digital converter (ADC) module with built in sample-and-hold (S/H) circuitry and an event manager (EV) that produces PW signals for the power switches.

Figure 6.2 shows an illustrative representation of EV configuration. The carrier signals $carA$ and $carB$ are produced by the EV timer at frequency $f_C$. The compare registers $Eva.CMPRx$ and $Evb.CMPRx$ are sampled by the individual sample-and-hold (S/H) structures with sampling frequency $f_{SS} = 2f_C$. The PWM output signals result from the comparison of each synchronously sampled compare register value with the respective EV carrier value.

The interface between DSP with motherboard and software programmer is achieved through CPT-DA2810 controller card [68]. As indicated in Fig. 6.3, this controller card contains all necessary auxiliary circuitry for DSP functionality, such as regulated power supply, analog inputs conditioning and protection, digital signals buffering, on-board flash memory, and suitable physical connections for chip programming and serial communications. The CPT-DA2810 controller card is directly mounted on the CPT-E13 motherboard. Figure 6.4 shows a functional block diagram of the CPT-E13 board.

Figure 6.2: Illustrative representation of the general event manager configuration.
Chapter 6. Description of the Experimental System

Figure 6.3: Functional block diagram of the CPT-DA2810 controller card [67].

Figure 6.4: Functional block diagram of the CPT-E13 controller motherboard. [68].
6.2. Software Building Blocks

The DSP code for running the inverter boxes in the experiments are extended from previously developed base code provided by CPT, which contain serial communication, analog to digital converter readings, PWM modulator, and a user interface. The tasks within the code are either interrupt-driven foreground tasks or low-priority background tasks. The closed-loop regulator and PWM modulator are some of the foreground tasks, while the background tasks interact with the operator through the serial communication port.

6.2.1. Foreground Tasks

The foreground task is assigned for reading the analog inputs, sinusoidal waveform generation and PWM command updates. The sine wave look up table stored in the DSP program memory is used to generate the sinusoidal pattern required for the current controller reference.

6.2.2. Background Tasks

The background code is executed in a continuous-loop, during interrupts of the foreground tasks. The reading of the current state of these parameters and their alteration is facilitated by the serial connection to the HMI laptop computer.

6.3. Experimental Setup

The experimental validation is conducted on a series cascaded structure formed with two DG units as described in section 3.4. The 3-phase RMIT-U01 power stage board is reconfigured by removing the links \( J_1 \) and \( J_2 \) so as to allow independent operation of two single phase DGs, with each of the units supplied with individual DC link voltages of \( v_{dc,1} \) and \( v_{dc,2} \). In order to minimise the effect of capacitive coupling to earth present in the adjustable DC supplies used in the experiment, common-mode filter inductors are included at the output of each DG. Also, LC filters
Chapter 6. Description of the Experimental System

are provided at the individual DG outputs so as to smoothen the output voltage waveform, and series connection is achieved across the filters as shown in Figure 6.5.

Figure 6.5. RMIT-U01 Power stage board in series-cascaded arrangement
Chapter 6. Description of the Experimental System

Figure 6.6: Experimental setup

The actual experimental setup is shown on Figure 6.6 in which the power stage board in configured to operate as two single phase DGs, DG#1 and DG#2 each fed from two independent dc sources $V_{dc1}$ and $V_{dc2}$. Common mode filter and low pass LC filters are provided at output of both DGs to minimise the coupling capacitance and to get better quality wave form at output. The cascaded connection is established at the output of LC filter and then connected across the load.

Figure 6.7 shows the experimental results showing power factor sharing in synchronisation mode of operation. In this mode the cascaded DG, operates at the microgrid power factor of 0.82 and the slave DG autonomously adjust its reactive
6.4. Summary

This chapter has presented the hardware and software building blocks utilised for the experimental validation of series-cascaded microgrid.

Figure 6.7: Experimental results showing DGs and microgrid power factor sharing in synchronisation mode.

power and pf reference (according to the methodology given in 3.2.1) to attain synchronisation. Detailed explanation and more results are explained on Chapter 3.
Chapter 7
Conclusions and Future Work

Distributed Generations based on renewable energy resources proved to be an effective alternate for conventional renewable energy resources and the microgrid concept has been developed as promising solution to integrate these resources through dedicated power electronics interface devices. Control and co-ordination of these resources is being a challenging issue among the researches since it has to ensure multiple targets like effective utilisation, flexibility in operation comparable to traditional power plants, maintaining voltage and frequency limits and ensure proper power sharing among them.

Compared with the traditional parallel microgrid, the series cascaded and hybrid microgrid structure proves to be an effective solution to integrate low voltage DG units to microgrid with reduced conversion stages or low gain boost dc-dc stage. Reasonable efforts are made by researchers to address the power sharing issues in series-cascaded and hybrid microgrid. However, the possibility of integrating dispatchable DG with non-dispatchable are not explored till now.

This thesis has two main contributions in this field. First a series-cascaded microgrid topology with mixed type resources is developed and the structure is extended to hybrid arrangement. Moreover, decentralised power management strategies are developed for both the microgrid structures.

This chapter summarises the main findings of this research and identifies possible avenues for further work.
Chapter 7. Conclusions

7.1. Summary of work

The contributions of this work can be classified into two major areas. These areas and their related findings are as follows.

7.1.1. Series-cascaded Topology Microgrid with Non-Dispatchable

This thesis has developed a new microgrid topology integrating dispatchable and non-dispatchable DGs (such as Solar PV system) in a series-cascaded manner. The system can effectively integrate low voltage non-dispatchable DGs to microgrid with reduced conversion stages in order to ensure proper power sharing among cascaded DGs and to satisfy the supplementary requirements raised by the non-dispatchables. A decentralised master-slave control based on two different droop schemes have been developed in this thesis. Both the control schemes can ensure accurate MPPT tracking with curtailment being active under light load conditions. In addition, the proposed control strategy can ensure constant microgrid voltage irrespective of the MPPT variation which demonstrates an enhanced system reliability.

7.1.2. Hybrid Microgrid

The proposed series-cascaded microgrid topology is then extended to form a hybrid microgrid structure with series-parallel arrangement of DGs. In this arrangement, the string units are formed by series arrangement of dispatchable and non-dispatchable DGs, and they are then paralleled across the point-of-common-coupling (PCC). Due to contradicting control requirements raised by the series and parallel connected DGs, a new power management scheme is developed that can ensure proper power sharing at string level and at DG level. This hybrid topology can exploit the benefits of both series-cascaded and parallel structure while ensuring direct integration of low voltage solar PV units. The power management schemes developed in this work can ensure effective utilisation of non-dispatchable DGs (solar PV units) while offering
provision to curtail surplus power under light load conditions. The system can thus offer an effective power balance even without any storage element.

7.1.3. Further Research

Research work presented in this thesis for series-cascaded and hybrid microgrid can enable direct integration of low-voltage units to microgrid with reduced or low gain boost conversion stages. While the control strategies developed in this work can offer better power management with mixed type of resources, there is still scope for further research in this area.

7.1.4. Series-cascaded Topology Microgrid with Non-Dispatchable

The Master-slave control strategy developed for series-cascaded microgrid can ensure proper power management among cascaded DGs and address the requirements raised by non-dispatchable solar PV units. However, the proposed architecture includes only one dispatchable DG in the cascaded structure. Integration of multiple dispatchable and non-dispatchable DGs should be explored. Hence further study has been conducted to carry out for the integration of multiple dispatchables and non-dispatchables in a series-cascaded manner.

Secondly, the proposed series-cascaded microgrid is developed without considering any expensive storage element and excess power is curtailed under light load condition. Even though, this seems to be a better idea in terms of initial cost, adding storage element can ensure maximum utilisation of available resources. Integrating storage element to proposed structure need to satisfy suitable control strategy to maintain the State-of-Charge (SoC) of storage element within the constraints.

7.1.5. Hybrid Topology Microgrid with Non-Dispatchable

The control strategy developed for hybrid microgrid can match the power sharing characteristics demonstrated by the traditional parallel structure. The
Chapter 7. Conclusions

The proposed method is developed for inductive line and droop curves have to be modified for resistive type of line.

Moreover, the proposed scheme has been tested for linear type of loads. Since the system uses parallel arrangement of string units, special harmonic sharing techniques has to be developed to achieve power sharing at string level for nonlinear loads.

Integration storage elements into existing hybrid structure need communication channels for the SoC management and hence a detailed study has to be carried out on this issue.
Appendix A
MATLAB Script

This appendix provides MATLAB script developed for frequency response analysis of the master DG

% CurrRegBodePlot.m

clear;
clc;

VDC = 250;  % Dc link voltage
Lf=4.4e-3;  % Filter inductor
Cf=40e-6;  % Filter inductor

w0= 50*2*pi;
fic = 5000; % Switching frequency
Td = 3/4/fc;
phi_m = 45;

phi_m_rad = phi_m*pi/180;
omega_c = (pi/2 - phi_m_rad)/Td
Kpc = omega_c*Lf/VDC % Proportional gain
Trc = 10/omega_c% Integral gain

Kpv = Kpc/5
Trv = Trc

Gplant = tf([1],[Lf 0]);
delay_sys = exp(-tf([1 0],[1])*Td);

Zc=tf([1],[Cf 0])

Gcc_ol=Gpr_c*VDC*delay_sys*Gplant;
Gcc_cl=feedback(Gcc_ol,1);

Gvc_ol=(Gpr_v*delay_sys)/(1+(Gpr_c*Gpr_v*delay_sys)+(Gplant/Zc)+(Gpr_c*delay_sys/Zc));

ww = logspace(0,6,10e3);
[magcm(:,1),phasecm(:,1)] = bode(Gvc_cl,ww);
Chapter 7. Conclusions

```matlab
margin(Gvc_c1)
figure();
subplot(2,1,1);
semilogx(ww./2./pi,20*log10(magcm),'b--','linewidth',3);
ylabel('Magnitude (dB)');
axis([min(ww./2./pi) max(ww./2./pi) -20 80]);
set(gca,'XTickLabel',[]);
% line([945 945],[-20 80],'linewidth',1,'color','black','linestyle','--');
line([110 110],[-210 110],'linewidth',1,'color','black','linestyle','--');
line([1 20000],[0 0],'linewidth',1,'color','black','linestyle','--');
line([1 20000],[-180 -180],'linewidth',1,'color','black');
xlabel('Frequency (Hz)');

subplot(2,1,2);
semilogx(ww./2./pi,phasecm,'b--','linewidth',3);
ylabel('Phase (deg)');
axis([min(ww./2./pi) max(ww./2./pi) -500 0]);
% line([omega_c/2/pi omega_c/2/pi],[-210 110],'linewidth',1,'color','black','linestyle','--');
line([110 110],[-210 110],'linewidth',1,'color','black','linestyle','--');
line([1 20000],[-110 -110],'linewidth',1,'color','black','linestyle','--');
line([1 20000],[-180 -180],'linewidth',1,'color','black');
xlabel('Frequency (Hz)');
```

105
Appendix B
DSP Source Code: Experimental System

This appendix provides the TMS320F2810 DSP C source code used in experimental system.

C.1. Background Routines – Header File

```c
// main.h (Background H file)
/*=========================================================================*/
#define SYSCLK_OUT (150e6)
#define HSPCLK (SYSCLK_OUT)
#define LSPCLK (SYSCLK_OUT/4)
#define INIT_SYSCLK ((Uint16)(SYSCLK_OUT/1e6)) ///< Initial System Clock in MHz
#define INIT_LSPCLK ((Uint32)(LSPCLK/1e3)) ///< Initial Low Speed Clock kHz
#define INIT_HSPCLK ((Uint32)(HSPCLK/1e3)) ///< Initial High Speed Clock kHz
#define TABLE_SIZE 512 /// boot ROM sine table size
#define MAX_SINE_TABLE 16384 /// boot ROM sine table magnitude
#define SW_FREQ 5000.0 ///< switching frequency in Hz
#define F_FREQ 50.0 ///< fundamental frequency
#define VHI_AB_NOM 250.0 ///200.0 ///< target hv dc bus Vdc
#define VHI_CD_NOM 125.0 ///200.8 ///< target hv dc bus Vdc
#define IAC_NOM 7.5 ///< rated AC current in Arms
#define MOD_MAX 10000 ///< max mod depth in 0.01%
#define MOD_100 10000 ///< 100% mod depth in 0.01%
#define MOD_STEP_SMALL 10
```
funcPtr f;  ///< pointer to present state function
#define MOD_STEP_LARGE 200
#define I_MAX 1000  ///< max current ref in 0.01A
#define I_STEP_SMALL 1
#define I_STEP_LARGE 10
#define IREF_10 10  ///< current Ref scaling in 0.1A
#define V_MAX 2500  ///< max current ref (10 = 1V)
#define V_STEP_SMALL 10
#define V_STEP_LARGE 100
#define VREF_10 10

/*====================================================================
==== __Fault_Definitions() 
====================================================================*/

/** @name Fault codes */
//@{
#define FAULT_PDPINT 0x0001  ///< hardware gate fault trip
#define FAULT_2 0x0002  ///< User Defined Fault Type 2
#define FAULT_3 0x0004  ///< User Defined Fault Type 3
#define FAULT_4 0x0008  ///< User Defined Fault Type 4
#define FAULT_5 0x0010  ///< User Defined Fault Type 5
#define FAULT_6 0x0020  ///< User Defined Fault Type 6
#define FAULT_7 0x0040  ///< User Defined Fault Type 7
#define FAULT_8 0x0080  ///< User Defined Fault Type 8
#define FAULT_9 0x0100  ///< User Defined Fault Type 9
#define FAULT_10 0x0200  ///< User Defined Fault Type 10
#define FAULT_11 0x0400  ///< User Defined Fault Type 11
#define FAULT_12 0x0800  ///< User Defined Fault Type 12
#define FAULT_13 0x1000  ///< User Defined Fault Type 13
#define FAULT_14 0x2000  ///< User Defined Fault Type 14
#define FAULT_15 0x4000  ///< User Defined Fault Type 15
#define FAULT_16 0x8000  ///< User Defined Fault Type 16
//@}
#define FAULT_MAX 16  // faults that prevent converter operation
#define FAULT_FATAL (0xFFFF & (~FAULT_16))

extern Uint16 detected Faults;

// State function type
typedef void (*funcPtr)(void);

// Simple state machine type
typedef struct{
  funcPtr f;  ///< Pointer to present state function
  int first;  ///< Flag set for first execution of present state function
 } State_Type;

/* State Handling Macros */
#define NEXT_STATE(_s_,_f_){ _s_.f = (funcPtr)&_f_; 
                         _s_.first = 1; }
#define IS_FIRST_STATE(_s_)(_s_.first == 1)  
#define DONE_FIRST_STATE(_s_)_s_.first = 0  
#define DO_STATE(_s_)((*(_s_.f))())
#define IS_CURRENT_STATE(_s_,_f_) (_s_.f == (funcPtr)&_f_)

/*====================================================================
==== __Watchdog_Timer_definitions() 
====================================================================*/

/** @name Watchdog Timer Definitions */
//@{
// fast 1 msec watchdogs
#define WD_TIMER_MAX 6

// @}
### C.2. Background Routines – C Source File

// main.c (Background C-File for System #1)
// compiler standard include files
#include <DSP281x_Device.h>
#include <DSP281x_Examples.h>
#include <bios0.h>

// board standard include files
#include <lib_da2810.h>
#include <lib_e13.h>
#include <conio.h>

// local include files
#include "lib_u01.h"
#include "grab.h" // uses local project directory instance of the header file
#include "main.h"
#include "vsi.h"

/*

---------------------------------------------
__Definitions()
---------------------------------------------
#define HELP_START 50 //< Start Location in Switch of Help Text
#define HELP_FINISH (HELP_START+15) //< End Location in Switch of Help Text
/

---------------------------------------------
__Typedefs()
---------------------------------------------

// Time related flag type
/** This structure holds flags used in background timing. */
typedef struct {
  Uint16 msec:1, //< millisecond flag
  sec0_1:1, //< tenth of a second flag
  sec:1; //< second flag
} type_flag;

// Time type
/** This structure holds the variables used in background timing. */
typedef struct {
  type_flag flag; //< bitwise flag structure
  Uint16 count_msec; //< count of milliseconds since last second event
} type_time;
/

---------------------------------------------
__Variables()
---------------------------------------------

#define WD_CHARGE 0
#define WD_RAMP 1
#define WD_O485 2
#define WD_COMMs_BD 3
#define WD_BIOS0 4
#define WD_BIOS1 5
extern Uint16 wd_timer[WD_TIMER_MAX]; // watchdog countdown timers
//@}
Appendix B. DSP Source Code: Experimental System

```c
#ifndef BUILD_RAM
// These are defined by the linker (see build_flash.cmd)
extern Uint16 RamfuncsLoadStart;
extern Uint16 RamfuncsLoadEnd;
extern Uint16 RamfuncsRunStart;
#endif

// Background variables
Uint16
quit = 0, ///< exit flag
ref_mod = 0, ///< modulation depth reference in 0.01%
ref_i = 0, ///< current reference in 0.1 A
ref_v_AD = 0, ///< vAD voltage reference in 0.1 V
ref_v_CD = 0, ///< vCD voltage reference in 0.1 V
vsi_mode = 0; ///< operating mode

/// timing variables
type_time
time =
{
  0,0,0,
  0
};

Uint16
wd_timer[WD_TIMER_MAX]; ///< watchdog countdown timers

char
*vsi_state_str[] =
{
  "Stop ", "Ramp ", "Run ", "Fault ", "Error "
};

double debug_varA = 0.0;
double debug_varB = 0.0;
double debug_varC = 0.0;
double debug_varD = 0.0;
extern Uint32 TransientCounter;
extern Uint16 EnablePFReg ;
/

===================================================================== 
=======
__Local_Function_Prototypes() 
===================================================================== 
=======
// display operating info
void com_display(Uint16 mode);

// process keyboard input
void com_keyboard(void);

/// 1ms interrupt for display
interrupt void isr_cpu_timer0(void);

void InitWDTimers(void);

===================================================================== 
====
__Main() 
==================================
=================================== 
=======
This is the main function. It:
\li Initialises the DSP and its peripherals
\li Initialises the DA2810 PCB into a safe condition
\li Copies the RAM based functions to RAM
\li Starts the 1ms timers for background timing
\li Sets up the com port
\li Initialises the various software modules
\li Runs the background loop

*/
void main(void)
```

Appendix B. DSP Source Code: Experimental System

{  
DINT;
// Initialise DSP for PCB
lib_e13_init(INIT_SYSCLK, INIT_LSPCLK, INIT_HSPCLK);
RESET_GATES();

// Initialize the PIE control registers to their default state.
InitPieCtrl();
// Disable CPU interrupts and clear all CPU interrupt flags:
IER = 0x0000;
IFR = 0x0000;
// Initialize the PIE vector table with pointers to the shell
// Interrupt Service Routines (ISR).
// This will populate the entire table, even if the interrupt
// is not used in this example. This is useful for debug purposes.
// The shell ISR routines are found in DSP281x_DefaultIsr.c.
// This function is found in DSP281x_PieVect.c.
InitPieVectTable();
#endif
// Copy time critical code and Flash setup code to RAM
// The RamfuncsLoadStart, RamfuncsLoadEnd, and RamfuncsRunStart
// symbols are created by the linker. Refer to the F2810.cmd file.
MemCopy(&RamfuncsLoadStart, &RamfuncsLoadEnd, &RamfuncsRunStart);

// Call Flash Initialization to setup flash waitstates
// This function must reside in RAM
InitFlash();
# endif
// Initialize ADC
InitCpuTimers(); // Initialize CPU Timers
InitWDTimers();
bios_init_COM0(38400L); // Initialize COM Port 0 (SCIA at 9600
// Configure CPU-Timer 0 to interrupt every millisecond:
// 150MHz CPU Freq, 0.001 second Period (in uSeconds)
ConfigCpuTimer(&CpuTimer0, 150.0/*MHz*/, 1000.0/*us*/);
StartCpuTimer0();

// Map interrupt to interrupt service routine.
EALLOW;
PieVectTable.TINT0 = &isr_cpu_timer0;
EDIS;

// Enable TINT0 in the PIE: Group 1 interrupt 7
PieCtrlRegs.PIEIER1.bit.INTx7 = 1;
IER |= M_INT1; // Enable CPU Interrupt 1
EnableInterrupts(); // Enable Interrupts

//puts_COM0("CPT-DA2810 VSI " __DATE__ "__TIME__\n");
//puts_COM0("IIla Andreev STATCOM\n");
GrabInit();

vsi_set_mod(ref_mod);
deebug_varA = 0.0;

/*
void main_loop(void)
*/
while(quit == 0)
{
  if (time.flag.msec != 0) // millisecond events
  {
    time.flag.msec = 0;
    vsi_state_machine();
droop_control();
  }
  else if (time.flag.sec0_1 != 0) // tenth of second events
  { 
    time.flag.sec0_1 = 0;
    com_keyboard(); // process key press
    com_display(0); // display steps at 1/10th second
    if (GrabAvail())

  }
}
{ 
    GrabDisplay();  
}
}
}   
else if (time.flag.sec != 0) // one second events  
{
    time.flag.sec = 0;  
    if (!GrabAvail()) // if there isn't any grab data to print
        com_display(1); // then trigger new one second display 
}
else // low priority events
{
}
} /* end while quit == 0 */

EvaRegs.TICON.bit.TENABLE = 0;  
EvaRegs.ACTRA.all = 0x0000;

DINT;
} /* end main */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * */
/**  
Display operating information out COM0.  
*/

\author A.McIver  
\par History:  
\li 22/06/05 AM - initial creation  
*/

void com_display(Uint16 mode)
{
    static Uint16  
        display_state = 0xFFFF;  
    int16  
        fault_state = 0;

    // don't trigger until existing information has printed
    if ((mode == 1) && (display_state > HELP_FINISH))
    {
        /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
        * * * * */
    }

    display_state = 0;
    else if (mode == 2) // display help parameters
        display_state = HELP_START;
    else
        display_state++;

    switch (display_state)
    {
        case 0:
            fault_state = vsi_get_status();
            puts_COM0("\n");  
            if (fault_state == -1)  
            {
                puts_COM0("F");
                putxx(detected_faults);
                puts_COM0(" ");
            }
            puts_COM0(vsi_state_str[vsi_get_state()]);
            break;

        case 1:
            if (vsi_mode == 0)
                puts_COM0("Mode: OpenLoop ");
            else if (vsi_mode == 1)
                puts_COM0("Mode: CurrentReg ");
            else if (vsi_mode == 2)
                puts_COM0("Mode: VoltageReg ");
            break;

        case 2:
            puts_COM0("mod: ");
            putdbl((double)ref_mod/100.0,2);
            puts_COM0("% ");
            break;

        case 3:
            puts_COM0("Iref: ");
            putdbl((double)ref_i/10.0,1);
            puts_COM0("A ");
            break;

    }
case 4:
    puts_COM0("v_AD_pk: ");
    putdbl((double)ref_v_AD/10.0,1);
    puts_COM0("V ");
    break;

case 5:
    puts_COM0("v_CD_pk: ");
    putdbl((double)ref_v_CD/10.0,1);
    puts_COM0("V ");
    break;

// Print Help Screen
    case HELP_START:
        puts_COM0("n4 phase leg VSI basic functions\n\n");
        break;
    case HELP_START+1:
        puts_COM0("\n\ne - enable active rectifier switching\n");
        break;
    case HELP_START+2:
        puts_COM0("\n\nE - enable PWM outputs\n");
        break;
    case HELP_START+3:
        puts_COM0("\n\nd - disable active rectifier switching\n");
        break;
    case HELP_START+4:
        puts_COM0("\n\nD - disable PWM outputs\n");
        break;
    case HELP_START+5:
        puts_COM0("\n\nF - clear faults\n");
        break;
    case HELP_START+6:
        puts_COM0("\n\nV - Open Loop PWM\n");
        break;
    case HELP_START+7:
        puts_COM0("\n\nC - Closed Loop PI Current Regulation\n");
        break;
    case HELP_START+8:
        puts_COM0("\n\ni/m - Increment/Decrement moduation\n");
        break;
    case HELP_START+9:
        puts_COM0("\n\ni/n - Increment/Decrement Current Reference\n");
        break;
    case HELP_START+10:
        puts_COM0("\n\ni/n - Fast Increment/Decrement Current Reference\n");
        break;
    case HELP_START+11:
        puts_COM0("\n\ni/N - Fast Increment/Decrement Current Reference\n");
        break;
    case HELP_START+12:
        puts_COM0("\n\ng - Start Grab (must be set up first)\n");
        break;
    case HELP_START+13:
        puts_COM0("\n\nG - Stop Grab (must be set up first)\n");
        break;
    case HELP_START+14:
        puts_COM0("\n\n? - Print Help Screen\n");
        break;
    default:
        display_state = 0xF000;
        break;
    } /* end com_display */

    /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
    /* void com_keyboard 
Parameters: none 
Returns: nothing 
Description: Process characters from COM0. 
Notes: 
q - quit 
e - enable VSI switching 
d - disable VSI switching 
F - clear faults 
i - increment reference modulation depth by 1 
I - fast increment reference modulation depth by 500 
m - decrement reference modulation depth by 1 
M - fast decrement reference modulation depth by 500 
i - increment reference modulation depth by 10 
U - fast increment reference modulation depth by 50 */
n - decrement reference modulation depth by 10
N - fast decrement reference modulation depth by 50

Grab Code
- g - start grab
- G - stop Grab Interrupt Data

History:
22/06/05 AM - initial creation
*/
void com_keyboard(void)
{
char
c;
if (kbhit_COM0())
{
c = getcc_COM0();
switch (c)
{
    case 'e':
        vsi_set_mod(ref_mod);
        vsi_enable();
        break;
    case 'd':
        vsi_disable();
        break;
    case 'E':
        EvaRegs.ACTRA.all = 0x6666;
        EvbRegs.ACTRB.all = 0x6666;
        break;
    case 'D':
        EvaRegs.ACTRA.all = 0x0000;
        EvbRegs.ACTRB.all = 0x0000;
        break;
    case 'F':
        detected_faults = vsi_check_fault();
        break;
    case 'C':
        vsi_set_mode(VSI_CURR);
        vsi_mode = 1;
        puts_COM0("\n\nCurrent Regulation\n\n");
        break;
    case 'O':
        vsi_set_mode(VSI_OL);
        vsi_mode = 0;
        puts_COM0("\n\nOpen Loop Modulation\n\n");
        break;
    case 'V':
        vsi_set_mode(VSI_VOLT);
        vsi_mode = 2;
        puts_COM0("\n\nVoltage Regulation\n\n");
        break;
    case 'i':
        if (ref_mod < MOD_MAX - MOD_STEP_SMALL)
            ref_mod += MOD_STEP_SMALL;
        vsi_set_mod(ref_mod);
        break;
    case 'I':
        if (ref_mod <= MOD_MAX - MOD_STEP_LARGE)
            ref_mod += MOD_STEP_LARGE;
        vsi_set_mod(ref_mod);
        break;
    case 'm':
        if (ref_mod > MOD_STEP_SMALL)
            ref_mod -= MOD_STEP_SMALL;
        vsi_set_mod(ref_mod);
        break;
    case 'M':
        if (ref_mod >= MOD_STEP_LARGE)
            ref_mod -= MOD_STEP_LARGE;
        vsi_set_mod(ref_mod);
        break;
    case 'u':
        if (ref_i < I_MAX - I_STEP_SMALL)
            ref_i += I_STEP_SMALL;
        vsi_set_i(ref_i);
        break;
    case 'U':
        if (ref_i < I_MAX - I_STEP_LARGE)
            ref_i += I_STEP_LARGE;
        vsi_set_i(ref_i);
        break;
    case '0':
        vsi_set_mod(VSI_OL);
        vsi_mode = 0;
        puts_COM0("\n\nOpen Loop Modulation\n\n");
        break;
    case 'V':
        vsi_set_mode(VSI_VOLT);
        vsi_mode = 2;
        puts_COM0("\n\nVoltage Regulation\n\n");
        break;
    case 'i':
        if (ref_mod < MOD_MAX - MOD_STEP_SMALL)
            ref_mod += MOD_STEP_SMALL;
        vsi_set_mod(ref_mod);
        break;
    case 'I':
        if (ref_mod <= MOD_MAX - MOD_STEP_LARGE)
            ref_mod += MOD_STEP_LARGE;
        vsi_set_mod(ref_mod);
        break;
    case 'm':
        if (ref_mod > MOD_STEP_SMALL)
            ref_mod -= MOD_STEP_SMALL;
        vsi_set_mod(ref_mod);
        break;
    case 'M':
        if (ref_mod >= MOD_STEP_LARGE)
            ref_mod -= MOD_STEP_LARGE;
        vsi_set_mod(ref_mod);
        break;
    case 'u':
        if (ref_i < I_MAX - I_STEP_SMALL)
            ref_i += I_STEP_SMALL;
        vsi_set_i(ref_i);
        break;
    case 'U':
        if (ref_i < I_MAX - I_STEP_LARGE)
            ref_i += I_STEP_LARGE;
        vsi_set_i(ref_i);
        break;
    case '0':
        vsi_set_mod(VSI_OL);
        vsi_mode = 0;
        puts_COM0("\n\nOpen Loop Modulation\n\n");
        break;
    case 'V':
        vsi_set_mode(VSI_VOLT);
        vsi_mode = 2;
        puts_COM0("\n\nVoltage Regulation\n\n");
        break;
    case 'i':
        if (ref_mod < MOD_MAX - MOD_STEP_SMALL)
            ref_mod += MOD_STEP_SMALL;
        vsi_set_mod(ref_mod);
        break;
    case 'I':
        if (ref_mod <= MOD_MAX - MOD_STEP_LARGE)
            ref_mod += MOD_STEP_LARGE;
        vsi_set_mod(ref_mod);
        break;
    case 'm':
        if (ref_mod > MOD_STEP_SMALL)
            ref_mod -= MOD_STEP_SMALL;
        vsi_set_mod(ref_mod);
        break;
    case 'M':
        if (ref_mod >= MOD_STEP_LARGE)
            ref_mod -= MOD_STEP_LARGE;
        vsi_set_mod(ref_mod);
        break;
    case 'u':
        if (ref_i < I_MAX - I_STEP_SMALL)
            ref_i += I_STEP_SMALL;
        vsi_set_i(ref_i);
        break;
    case 'U':
        if (ref_i < I_MAX - I_STEP_LARGE)
            ref_i += I_STEP_LARGE;
        vsi_set_i(ref_i);
        break;
    case '0':
        vsi_set_mod(VSI_OL);
        vsi_mode = 0;
        puts_COM0("\n\nOpen Loop Modulation\n\n");
        break;
    case 'V':
        vsi_set_mode(VSI_VOLT);
        vsi_mode = 2;
        puts_COM0("\n\nVoltage Regulation\n\n");
        break;
    case 'i':
        if (ref_mod < MOD_MAX - MOD_STEP_SMALL)
            ref_mod += MOD_STEP_SMALL;
        vsi_set_mod(ref_mod);
        break;
    case 'I':
        if (ref_mod <= MOD_MAX - MOD_STEP_LARGE)
            ref_mod += MOD_STEP_LARGE;
        vsi_set_mod(ref_mod);
        break;
    case 'm':
        if (ref_mod > MOD_STEP_SMALL)
            ref_mod -= MOD_STEP_SMALL;
        vsi_set_mod(ref_mod);
        break;
    case 'M':
        if (ref_mod >= MOD_STEP_LARGE)
            ref_mod -= MOD_STEP_LARGE;
        vsi_set_mod(ref_mod);
        break;
    case 'u':
        if (ref_i < I_MAX - I_STEP_SMALL)
            ref_i += I_STEP_SMALL;
        vsi_set_i(ref_i);
        break;
    case 'U':
        if (ref_i < I_MAX - I_STEP_LARGE)
            ref_i += I_STEP_LARGE;
        vsi_set_i(ref_i);
        break;
    case '0':
        vsi_set_mod(VSI_OL);
        vsi_mode = 0;
        puts_COM0("\n\nOpen Loop Modulation\n\n");
        break;
    case 'V':
        vsi_set_mode(VSI_VOLT);
        vsi_mode = 2;
        puts_COM0("\n\nVoltage Regulation\n\n");
        break;
    case 'i':
        if (ref_mod < MOD_MAX - MOD_STEP_SMALL)
            ref_mod += MOD_STEP_SMALL;
        vsi_set_mod(ref_mod);
        break;
    case 'I':
        if (ref_mod <= MOD_MAX - MOD_STEP_LARGE)
            ref_mod += MOD_STEP_LARGE;
        vsi_set_mod(ref_mod);
        break;
    case 'm':
        if (ref_mod > MOD_STEP_SMALL)
            ref_mod -= MOD_STEP_SMALL;
        vsi_set_mod(ref_mod);
        break;
    case 'M':
        if (ref_mod >= MOD_STEP_LARGE)
            ref_mod -= MOD_STEP_LARGE;
        vsi_set_mod(ref_mod);
        break;
    case 'u':
        if (ref_i < I_MAX - I_STEP_SMALL)
            ref_i += I_STEP_SMALL;
        vsi_set_i(ref_i);
        break;
    case 'U':
        if (ref_i < I_MAX - I_STEP_LARGE)
            ref_i += I_STEP_LARGE;
        vsi_set_i(ref_i);
        break;
    case '0':
        vsi_set_mod(VSI_OL);
        vsi_mode = 0;
        puts_COM0("\n\nOpen Loop Modulation\n\n");
        break;
    case 'V':
        vsi_set_mode(VSI_VOLT);
        vsi_mode = 2;
        puts_COM0("\n\nVoltage Regulation\n\n");
        break;
    case 'i':
        if (ref_mod < MOD_MAX - MOD_STEP_SMALL)
            ref_mod += MOD_STEP_SMALL;
        vsi_set_mod(ref_mod);
        break;
    case 'I':
        if (ref_mod <= MOD_MAX - MOD_STEP_LARGE)
            ref_mod += MOD_STEP_LARGE;
        vsi_set_mod(ref_mod);
        break;
    case 'm':
        if (ref_mod > MOD_STEP_SMALL)
            ref_mod -= MOD_STEP_SMALL;
        vsi_set_mod(ref_mod);
        break;
    case 'M':
        if (ref_mod >= MOD_STEP_LARGE)
            ref_mod -= MOD_STEP_LARGE;
        vsi_set_mod(ref_mod);
        break;
    case 'u':
        if (ref_i < I_MAX - I_STEP_SMALL)
            ref_i += I_STEP_SMALL;
        vsi_set_i(ref_i);
        break;
    case 'U':
        if (ref_i < I_MAX - I_STEP_LARGE)
            ref_i += I_STEP_LARGE;
        vsi_set_i(ref_i);
        break;
    case '0':
        vsi_set_mod(VSI_OL);
        vsi_mode = 0;
        puts_COM0("\n\nOpen Loop Modulation\n\n");
        break;
    case 'V':
        vsi_set_mode(VSI_VOLT);
        vsi_mode = 2;
        puts_COM0("\n\nVoltage Regulation\n\n");
        break;
    case 'i':
        if (ref_mod < MOD_MAX - MOD_STEP_SMALL)
            ref_mod += MOD_STEP_SMALL;
        vsi_set_mod(ref_mod);
        break;
    case 'I':
        if (ref_mod <= MOD_MAX - MOD_STEP_LARGE)
            ref_mod += MOD_STEP_LARGE;
        vsi_set_mod(ref_mod);
        break;
    case 'm':
        if (ref_mod > MOD_STEP_SMALL)
            ref_mod -= MOD_STEP_SMALL;
        vsi_set_mod(ref_mod);
        break;
    case 'M':
        if (ref_mod >= MOD_STEP_LARGE)
            ref_mod -= MOD_STEP_LARGE;
        vsi_set_mod(ref_mod);
        break;
    case 'u':
        if (ref_i < I_MAX - I_STEP_SMALL)
            ref_i += I_STEP_SMALL;
        vsi_set_i(ref_i);
        break;
    case 'U':
        if (ref_i < I_MAX - I_STEP_LARGE)
            ref_i += I_STEP_LARGE;
vsi_set_i(ref_i);
break;
case 'n':
    if (ref_i > I_STEP_SMALL)
        ref_i = I_STEP_SMALL;
    vsi_set_i(ref_i);
    break;
case 'N':
    if (ref_i > I_STEP_LARGE)
        ref_i = I_STEP_LARGE;
    vsi_set_i(ref_i);
    break;

// vAD
case 'a':
    if (ref_v_AD < V_MAX - V_STEP_SMALL)
        ref_v_AD += V_STEP_SMALL;
    vsi_set_v_AD(ref_v_AD);
    break;
case 'A':
    if (ref_v_AD < V_MAX - V_STEP_LARGE)
        ref_v_AD += V_STEP_LARGE;
    vsi_set_v_AD(ref_v_AD);
    break;
case 'z':
    if (ref_v_AD > V_STEP_SMALL)
        ref_v_AD = V_STEP_SMALL;
    vsi_set_v_AD(ref_v_AD);
    break;
case 'Z':
    if (ref_v_AD > V_STEP_LARGE)
        ref_v_AD = V_STEP_LARGE;
    vsi_set_v_AD(ref_v_AD);
    break;

// Transient condition
case 't':
    TransientCounter = 1;
    break;
case 'p':
    EnablePFReg = 1;
    break;
case 'l':
    EnablePFReg = 0;
    break;

// Grab Code Start
case 'g': // grab interrupt data
    GrabRun();
    break;
case 'G': // stop grab interrupt data
    GrabStop();
    GrabClear();
    break;
Appendix B. DSP Source Code: Experimental System

```c
// Display Help
case '?': // print help information
    com_display(2);
    break;

} /* end com_keyboard */

/__Interrupts()
======================================================================
======= */

/*===================================================================
__Interrupts()
===================================================================*/

/*===================================================================
**
This interrupt occurs every millisecond. It sets flags at different
intervals for the background loop to detect for event triggering. It also
processes the watchdog timers.

\author A.McIver
\par History:
  \li 22/06/05 AM - initial creation (derived from k:startup.c)
  \li 11/01/07 AM - added delay_counter
  \li 07/03/07 AM - added timing structure
  \li 09/03/07 AM - added watchdog timers
*/

#ifndef BUILD_RAM
#pragma CODE_SECTION(isr_cpu_timer0, "ramfuncs");
#endif

interrupt void isr_cpu_timer0(void)
{
    static Uint16
    i_count = 0,
    ii;

    for (ii=0; ii<WD_TIMER_MAX; ii++)
    {
        if (wd_timer[ii] > 0)
            wd_timer[ii]--;
        i_count++;
        if (i_count >= 100)
        {
            i_count = 0;
            time.flag.sec0_1 = 1;
            time.flag.msec = 1;
            time.count_msec++;
            if (time.count_msec >= 1000)
            {
                time.count_msec = 0;
                time.flag.sec = 1;
                // Acknowledge this interrupt to receive more interrupts from group 1
                PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
            }
        }

    } /* end isr_cpu_timer0 */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*/

/**
Ensures Watchdog timers are initialized to zero.
*/

void InitWDTimers(void)
{
    Uint16 i;

    for (i=0; i<WD_TIMER_MAX; i++)
    {
        wd_timer[i] = 0;
    }
    // end InitWDTimers
```
C.3. Foreground Routines – Header File

`// vshi.h (Foreground H-File )
/** @name VSI operating states */
#define VSI_STATE_STOP 0 ///< VSI is waiting for start signal
#define VSI_STATE_RAMP 1 ///< VSI is ramping up to target output
#define VSI_STATE_RUN 2 ///< VSI is operating normally
#define VSI_STATE_FAULT 3 ///< VSI is in a fault condition
#define VSI_STATE_ERROR 4 ///< VSI is in a fault condition
/** @name Inverter Operating Modes */
#define VSI_OL 0 ///< open loop mode
#define VSI_CURR 1 ///< Current regulated mode
#define VSI_VOLT 2 ///< Voltage regulated mode
/**
  * @name VSI operating states
  */
  //@{
    #define VSI_STATE_STOP 0 ///< VSI is waiting for start signal
    #define VSI_STATE_RAMP 1 ///< VSI is ramping up to target output
    #define VSI_STATE_RUN 2 ///< VSI is operating normally
    #define VSI_STATE_FAULT 3 ///< VSI is in a fault condition
    #define VSI_STATE_ERROR 4 ///< VSI is in a fault condition
  //@}

  /** @name Inverter Operating Modes */
  //@{
    #define VSI_OL 0 ///< open loop mode
    #define VSI_CURR 1 ///< Current regulated mode
    #define VSI_VOLT 2 ///< Voltage regulated mode
  //@}
Appendix B. DSP Source Code: Experimental System

```
Uint16 vsi_check_fault(void);

//Ilia Closed Loop Functions
double sin_calc(double theta_calc); //Interpolated Sin Lookup
double VdcReg_Compute(); //Update IdRef
void PLL_Compute(); //Update ThetaEst
void abc_ref_gen(); //Update Iabc_ref
void PR_Ireg_Compute(); //Update switching times

// droop control
void droop_control(void);

/** This structure hold variables relating to a single ADC channel. These
variables are used for filtering, averaging, and scaling of this
analog quantity. */
type_adc_ch

typedef struct
{
    int16 raw, ///< raw ADC result from last sampling
    filt;  ///< decaying average fast filter of raw data
    int32 rms_sum, ///< interrupt level sum of data
    rms_sum_bak, ///< background copy of sum for averaging
    dc_sum, ///< interrupt level sum
    dc_sum_bak; ///< background copy of sum for processing
    double real;  ///< background averaged and scaled measurement
} type_adc_ch;

/** This structure holds all the analog channels and some related
variables for the averaging and other processing of the analog inputs. There
are also virtual channels for quantities directly calculated from the analog
inputs.

There are two separate RMS calculations. The output AC currents are calculated
every fundamental cycle based on the VSI phase variable. The input AC
voltages are calculated every 0.2 seconds (~10 fundamental cycles). This is
because the input AC is not synchronous with the VSI. The maximum error over 10
cycles is +/-2.5%. The DC bus voltage and output DC voltage and current and
power are also calculated at this rate. */
typedef struct
{
    Uint16 count_cal, ///< counter for low speed calibration summation
    count_rms, ///< counter for full fund. period for RMS calculations
    count_rms_bak, ///< background copy of RMS counter
    count_dc, ///< counter for DC average calculations
    flag_cal, ///< flag set to trigger background calibration
    flag_rms, ///< flag set to trigger background RMS averaging
    flag_dc, ///< flag set to trigger background DC averaging
    type_adc_ch iac_a, ///< AC Current Phase A
    iac_b, ///< AC Current Phase B
    iac_c, ///< AC Current Phase C
    iac_a_dc, ///< AC Current Phase A - DC measurement
    iac_b_dc, ///< AC Current Phase B - DC measurement
    iac_c_dc; ///< AC Current Phase C - DC measurement
    idc, ///< DC Bus Current
    hvdc1, ///< DC Bus C/D Voltage
    hvdc2, ///< DC Bus A/B Voltage
```

117
Appendix B. DSP Source Code: Experimental System

```
hvdc, ///< Front Panel DC Voltage
vac_an, ///< Front Panel AC Voltage AN
vac_bn, ///< Front Panel AC Voltage BN
vac_an_dc, ///< Front Panel AC Voltage AN - DC measurement
vac_bn_dc, ///< Front Panel AC Voltage BN - DC measurement
yHA, ///< bank A high reference
yLA, ///< bank A low reference
yHB, ///< bank B high reference
yLB; ///< bank B low reference
}

type_adc;

/// Control loop type
/** This structure holds variables relating to a PI control loop. */
typedef struct
{
  int16 ref_adc, ///< reference quantity set by background in ADC counts
targ_adc, ///< target set by ramp or other control in ADC counts
  err, ///< error in ADC counts
  Kp, ///< proportional gain
  Ki; ///< integral gain
Uint16 overflow, ///< flag set if output overflows
  underflow; ///< flag set if output underflows
  int32 err_prop, ///< proportional error term
  err_int, ///< integral error term
  err_int_sum;///< summation of integral error term
}
type_pi_control;

/// Control loop type
/** This structure holds variables relating to a PR control loop. */
typedef struct
{
  int16 targ_adc, ///< target reference calculated in ISR
targ_adc, ///< target set by ramp or other control in ADC counts
  err, ///< error in ADC counts
  err_res,
  Kp, ///< proportional gain
  alpha0, ///< delta operator constant
  alpha2, ///< delta operator constant
  beta0, ///< delta operator constant
  beta1, ///< delta operator constant
  beta2, ///< delta operator constant
  s0, ///< delta operator state
  s1, ///< delta operator state
  s2, ///< delta operator state
  y0; ///< delta operator output
Uint16 v_sat; ///< count of switching saturation events
}
type_pr_control;
```

C 4. Source File

```
// csi.c (Foreground C-File for System #1)

#include "vlc_u01.h"
```
Appendix B. DSP Source Code: Experimental System

```c
#include "grab.h" // uses local project directory instance of the header file
#include "main.h"
#include "vsi.h"

#endif

/*
** Definitions()
======= */
#define __SQRT2 1.414213562
#define __SQRT3 1.732050808
#define __PI (3.141592653) ///< PI, &pi;

// make sure that PERIOD_2 is an integer
#define PERIOD_2 ((Uint16)(HSPCLK/SW_FREQ/4.0))
#define PERIOD (2*PERIOD_2)
#define VSI_FSW (HSPCLK/(PERIOD*2.0))
#define VSI_FINT (2.0*VSI_FSW)

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/// ADC averaging time
#define ADC_CAL_TIME 0.5 // seconds
#define ADC_COUNT_CAL (Uint16)(ADC_CAL_TIME * VSI_FSW)
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

// VSI definitions
#define MAX_TIME (signed int)(PERIOD_2-6)
#define POS_SAT 1 ///< Saturation - Positive
#define NEG_SAT -1 ///< Saturation - Negative

#define I_POS_SAT_A (10.0) ///< Voltage loop positive saturation value in Amp
#define I_NEG_SAT_A (-10.0) ///< Voltage loop negative saturation value in Amp

#define I_POS_SAT_ADCNT (int16)(I_POS_SAT_A/ADC_IAC_SC) /////< Voltage loop positive saturation value in adcnt
#define I_NEG_SAT_ADCNT (int16)(I_NEG_SAT_A/ADC_IAC_SC) /////< Voltage loop negative saturation value in adcnt

#define NOT_SAT 0 ///< Not in Saturation
/* the phase is scaled so that one fundamental is 2^32 counts */
#define FUNDAMENTAL_COUNTS (double)(4294967296.0)
#define PHASE_STEP_SC (FUNDAMENTAL_COUNTS/VSI_FSW/2.0)
#define PHASE_STEP (Uint32)(PHASE_STEP_SC*F_FREQ)

#define PHASE_120 (Uint32)(FUNDAMENTAL_COUNTS/3.0)
```
Appendix B. DSP Source Code: Experimental System

// ramp rate in 0.01% mod depth / ms
#define STEP_MOD_TARG 5

/*
   __Control Loop Controller Constants()
   *  *
*/
/** @name Input AC Current Ramp Step Size Definitions */
/** @name Control Loop Controller Constants */
//@{
/// supply freq in rad/s
#define F_OMEGA (2.0*PI*VSI_FSW)

//Current controller
#define PH_MARGIN_DEG (50.0)  ///< degrees
#define PH_MARGIN (PH_MARGIN_DEG*PI/180.0)  ///< radians
#define TRANS_DELAY (0.75/VSI_FSW)  ///< seconds
#define W_CRIT ((PI/2-PH_MARGIN)/TRANS_DELAY)  ///< rad/s

/// Proportional gain
#define KP_IAC_AB (W_CRIT*L_IN/VHI_AB_NOM)
#define KP_IAC_CD (W_CRIT*L_IN/VHI_CD_NOM)

/// Integral time
#define TINT_IAC (10.0/W_CRIT)

/** @name PI constants scaled for calculations (in time
   *  r counts and
   *  adc units) */
//@{
/// Scaled proportional gain
#define P_SHIFT_IAC_AB (5)
#define DEF_KP_IAC_AB ((int16)(KP_IAC_AB*ADC_IAC_SC*PERIOD_2*(1L<<P_SHIFT_IAC_AB)))
#define P_SHIFT_IAC_CD (5)
#define DEF_KP_IAC_CD ((int16)(KP_IAC_CD*ADC_IAC_SC*PERIOD_2*(1L<<P_SHIFT_IAC_CD)))

/// Scaled integral gain
#define I_SHIFT_IAC (14)
#define DEF_KI_IAC ((int16)(1.0/VSI_FINT/TINT_IAC*(1L<<I_SHIFT_IAC)))

/// Delta operator definitions
#define DELTA_Q (5)
#define DELTA_R (1<<(DELTA_Q-1))
#define ALPHA0_Q (14)
#define ALPHA0_R (1<<(ALPHA0_Q-1))

/// Error adc count scaling
#define ERR_SH_IAC_AB (1)
#define P_ERR_R_IAC_AB (1<<(P_SHIFT_IAC_AB+ERR_SH_IAC_AB-1))
#define ERR_SH_IAC_CD (1)
#define P_ERR_R_IAC_CD (1<<(P_SHIFT_IAC_CD+ERR_SH_IAC_CD-1))

/// Voltage controller
/// Proportional gain
#define KP_VAC_AD (KP_IAC_AB/5.0)
#define KP_VAC_CD (KP_IAC_CD/5.0)

/// Integral time
#define TINT_VAC_AD (10.0/W_CRIT)
#define TINT_VAC_CD (10.0/W_CRIT)

/// Scaled proportional gain
#define P_SHIFT_VAC_AD (9)
#define DEF_KP_VAC_AD    ((int16)((KP_VAC_AD*ADC_VAC_SC/ADC_IAC_SC)*(1L<<P_SHIFT_VAC_AD)))

#define P_SHIFT_VAC_CD   (9)
#define DEF_KP_VAC_CD    ((int16)((KP_VAC_CD*ADC_VAC_SC/ADC_IAC_SC)*(1L<<P_SHIFT_VAC_CD)))

/// Scaled integral gain
#define I_SHIFT_VAC_AD   (14)
#define DEF_KI_VAC_AD    ((int16)(1.0/VSI_FINT/TINT_VAC_AD*(1L<<I_SHIFT_VAC_AD)))

#define I_SHIFT_VAC_CD   (14)
#define DEF_KI_VAC_CD    ((int16)(1.0/VSI_FINT/TINT_VAC_CD*(1L<<I_SHIFT_VAC_CD)))

/// Error adc count scaling
#define ERR_SH_VAC_AD    (1)
#define P_ERR_R_VAC_AD   (1<<(P_SHIFT_VAC_AD+ERR_SH_VAC_AD-1))

#define ERR_SH_VAC_CD    (1)
#define P_ERR_R_VAC_CD   (1<<(P_SHIFT_VAC_CD+ERR_SH_VAC_CD-1))

# define MIN_VDC_AB_COMP 100.0 // V
#define MIN_ADC_VDC_AB_COMP ((int16)(MIN_VDC_AB_COMP/ADC_VDC_SC))
#define AB_COMP_SHIFT   9

#define VHI_AB_NOM_COUNT (VHI_AB_NOM/ADC_VDC_SC)
#define VHI_AB_NOM_COUNT_SHFT (int32)(VHI_AB_NOM_COUNT*(1L<<AB_COMP SHIFT))

#define MIN_VDC_CD_COMP 50.0 // V
#define MIN_ADC_VDC_CD_COMP ((int16)(MIN_VDC_CD COMP/ADC_VDC_SC))
#define CD_COMP_SHIFT   9

#define VHI_CD_NOM_COUNT (VHI_CD_NOM/ADC_VDC_SC)
#define VHI_CD_NOM_COUNT_SHFT (int32)(VHI_CD_NOM_COUNT*(1L<<CD_COMP_SHIFT))

#define QSG_K (_SQRT2)
#define F_NOM  (50.0)
#define W_N    (2.0*__PI*F_NOM )  /// nominal angular frequency
#define T_S    (1.0/VSI_FINT)  /// sampling period
#define KP_LF   (5)/3.8/2  /// loop filter proportional gain
#define TI_LF   (100e-3)/(400e-3)/950  /// loop filter integral time constant
#define KI_LF   ((KP_LF/TI_LF)*T_S)  /// loop filter integral gain

#define F_max   (51.0)
#define w_max   (2.0*__PI*F_max)  /// nominal angular frequency
#define Mp      (12.56)

/// 90deg offset in sine table to get cosine
#define TABLE OFFSET 90 256

#define UPPER LIMIT_AD (1000000.0)/(1000000.0)
#define LOWER LIMIT_AD (0.000000.0)/(0.000000.0)

#define LOWER LIMIT_CD (0.000000.0)/(0.000000.0)

#define FUNDAMENTAL COUNTS (double)(4294967296.0)
#define PHASE_180DEG (Uint32)(FUNDAMENTAL COUNTS/2.0 + 0.5)

//8}
Appendix B. DSP Source Code: Experimental System

/**

---_Macros()
---*/

#define VSI_FAST_STOP() {EvaRegs.ACTRA.all = 0x0000;
EvbRegs.ACTRB.all = 0x0000;}
#define VSI_ENABLE() {EvaRegs.ACTRA.all = 0x6666;
EvbRegs.ACTRB.all = 0x6666;}

// extracts the low 16 bits from a 32 bit number for grabbing
#define LOW16(_val_) ((int16)(_val_&0x0000FFFF))

// extracts the high 16 bits from a 32 bit number for grabbing
#define HIGH16(_val_) ((int16)(_val_>>16))

/*

---_Variables()
---*/

// state machine level variables
Uint16
is_vsi_switching = 0, ///< flag set if VSI switching is active
op_mode_vsi = VSI_OL, ///< operating mode of the VSI

/** @name VSI interrupt variables */
//@{
Uint16
detected_faults = 0x0000,
pwm_int_count = 0; ///< counter for interrupt level timing

// Boot ROM sine table starts at 0x003ff000 and has 641 entries of 32
// bit sine

// values making up one and a quarter periods (plus one entry). For
16 bit
// values, use just the high word of the 32 bit entry. Peak value is
0x40000000
#endif
int16
*sin_table = (signed int *)0x003ff0000; ///< ptr to sine table in
boot ROM
#endif

int16 phase_offset; ///< round off amount from sine lookup
int16
Ic_al_diff, ///< interpolation temp variable
Ic_al_lo, ///< interpolation temp variable
sin_Ic_al, ///< interpolated sine table values
cos_Ic_al; ///< interpolated sine table values

Uint32
phase_step_a = PHASE_STEP, ///< change in phase angle each interrupt
phase_step_a_new = PHASE_STEP, ///< change in phase angle each
interrupt
phase_step_b = PHASE_STEP, ///< change in phase angle each interrupt
phase_step_b_new = PHASE_STEP, ///< change in phase angle each
interrupt
phase_a = 0L, phase_b = 0L, phase_c = PHASE_180DEG, ///< running
phase angle (2^32 == 360 deg)
DCHBUS_comp_AB, DCHBUS_comp_CD; ///< DC bus compensation variable

Uint16
index_a = 0, index_b = 0, index_c = 0; ///< index into sine look-up
table
Uint16
bc_en_outputs = 0, ///< trigger to turn on VSI outputs
mod_ref = 0, ///< mod depth in 0.01% from background
mod_targ = 0, ///< target mod depth in 0.01% during ramp
mod_isr = 0, ///< mod depth scaled for ISR use

122
mod_offset = 0; // mod depth offset to study switching
Uint16
i_ref = 0, ///< current reference in 0.1A from background
i_targ = 0, ///< target current reference in 0.1A during ramp
i_isr = 0; ///< current ref scaled for ISR use

Uint16
v_ref_AD = 0, ///< Voltage reference
v_targ_AD = 0, ///< target Voltage reference during ramp
v_isr_AD = 0, ///< Voltage reference scaled for ISR use
v_ref_CD = 0, ///< Voltage reference
v_targ_CD = 0, ///< target Voltage reference during ramp
v_isr_CD = 0; ///< Voltage reference scaled for ISR use

int16
V_Asat = 0, ///< Phase A saturation entry flag for asymmetrical PWM
V_Bsat = 0, ///< Phase B saturation entry flag for asymmetrical PWM
V_Csat = 0, ///< Phase C saturation entry flag for asymmetrical PWM
V_Dsat = 0, ///< Phase D saturation entry flag for asymmetrical PWM
V_Asat_prev = 0, ///< Phase A saturation entry flag for asymmetrical PWM (previous state)
V_Bsat_prev = 0, ///< Phase B saturation entry flag for asymmetrical PWM (previous state)
V_Csat_prev = 0, ///< Phase C saturation entry flag for asymmetrical PWM (previous state)
V_Dsat_prev = 0; ///< Phase D saturation entry flag for asymmetrical PWM (previous state)

/// int16

Uint16
period = PERIOD,
period_2 = PERIOD_2;

int16
t_A, t_B, t_C, t_D, ///< switching times
t_off, ///< 3rd harmonic offset
t_a_cnt, t_b_cnt, t_c_cnt, t_d_cnt, ///< switching times from current loop
iref_ad, iref_cd, ///< ref currents from voltage control loops
adc_vdc_comp_AB, adc_vdc_comp_CD; ///< minimum clamped DC bus voltage for compensation

/** @name Control Loop Variables
* Structure to hold all of the control loop variables in one location
*/
//@{

//type_pi_control
//iac_a =
//{
//  0, // ref_adc
//  0, // targ_adc
//  0, // err
//  DEF_KP_IAC_AB, // Kp
//  DEF_KI_IAC, // Ki
//  0, // overflow
//  0, // underflow
//  0L, // err_int
//  0L // err_int_sum
// }; ///< Iac Phase A current control loop
//@}

//type_pi_control
//iac_b =
//{
//  0, // ref_adc
//  0, // targ_adc
//  0, // err
//  DEF_KP_IAC_AB, // Kp
//  DEF_KI_IAC, // Ki
//  0, // overflow
//  0, // underflow
//  0L, // err_int
//  0L // err_int_sum

123
Appendix B. DSP Source Code: Experimental System

```
//@}

// type_pr_control
// vac_an =
// {
// 0, // ref_adc_inst
// 0, // err
// 0, // err_res
// (DEF_KP_VAC_AD), // Kp
// (16384), // alpha0
// (517), // alpha1
// (16557), // alpha2
// (381), // beta0
// (24375), // beta1
// (0), // beta2
// (0), // s0
// (0), // s1
// (0), // s2
// (0), // y0
// (0), // v_sat
// }//< Iac phase C control loop

// type_pr_control
// vac_bn =
// {
// 0, // ref_adc_inst
// 0, // err
// 0, // err_res
// (DEF_KP_VAC_AD), // Kp
// (16384), // alpha0
// (517), // alpha1
// (16557), // alpha2
// (381), // beta0
// (24375), // beta1
// (0), // beta2
// (0), // s0
// (0), // s1
// (0), // s2
// (0), // y0
// (0), // v_sat
// }//< Phase AD voltage control loop
```
(0), // err_res
(DEF_KP_VAC_CD), // Kp
(16384), // alpha0
(517), // alpha1
(16557), // alpha2
(381), // beta0
(24375), // beta1
(0), // beta2
(0), // s0
(0), // s1
(0), // s2
(0), // y0
(0), // v_sat
}; ///< Phase CD voltage control loop

/** @name ADC variables */
//@{

type_adc adc =
{
  0, // count_cal
  0, // count_rms
  0, // count_rms_bak
  0, // count_dc
  0, // flag_cal,
  0, // flag_rms
  0, // flag_dc
  0, // raw
  0, // filt
  0L, // rms_sum
  0L, // rms_sum_bak
  0L, // dc_sum
  0L, // dc_sum_bak
  0.0 // real
}, // real
　}; // ADC calibration variables
#ifdef PSIM_VERSION // calibration factors are set to zero for PSIM environment

int16 cal_gainA = 1<<14, ///< calibration gain factor for A channel
cal_gainB = 1<<14, ///< calibration gain factor for B channel
cal_offsetA = 0,  ///< calibration offset for A channel
cal_offsetB = 0,  ///< calibration offset for B channel
cal_offset_idc = 0, ///< calibration offset for Idc

cal_offset_iac_a_dc = 0, ///< calibration offset for Iac_a_dc

cal_offset_iac_b_dc = 0, ///< calibration offset for Iac_b_dc

cal_offset_iac_c_dc = 0, ///< calibration offset for Iac_c_dc

cal_offset_hvdc1 = 0, ///< calibration offset for HVDC1

cal_offset_hvdc2 = 0, ///< calibration offset for HVDC2

cal_offset_hvdc = 0, ///< calibration offset for HVDC

cal_offset_vac_an = 0, ///< calibration offset for VAC1

cal_offset_vac_bn = 0;
#endif

#elifndef PSIM_VERSION // otherwise calibration factors are set to their tuned values for experiment

int16 cal_gainA = 1<<14, ///< calibration gain factor for A channel
cal_gainB = 1<<14, ///< calibration gain factor for B channel
cal_offsetA = 0,  ///< calibration offset for A channel
cal_offsetB = 0,  ///< calibration offset for B channel
cal_offset_Idc = 0,  ///< calibration offset for Idc:
cal_offset_Iac_a_dc = 7,  ///< calibration offset for Iac_a_dc
cal_offset_Iac_b_dc = 0,  ///< calibration offset for Iac_b_dc
cal_offset_Iac_c_dc = -29, ///< calibration offset for Iac_c_dc

cal_offset_hvdc1 = 0, ///< calibration offset for HVDC1

cal_offset_hvdc2 = 0, ///< calibration offset for HVDC2

cal_offset_vac_an_dc = -6; ///< calibration offset for VAC1

cal_offset_vac_bn_dc = -12; ///< calibration offset for VAC2

#endif

double cal_gain_A, cal_gain_B,
cal_offset_A, cal_offset_B;
//@}
Uint16 debug_var1 = 0;
Uint16 debug_var2 = 0;
Uint16 debug_var3 = 0;

int16 vAB = 0;
int16 vAD = 0;
int16 vCD = 0;

double Ic, Ia, V_AD, V_CD;

double Ic, Ia, V_AD, V_CD;

double qsg_x_Ic = 0.0; ///< internal variable of quadrature signal
generator
double qsg_x_Ia = 0.0;
double qsg_x_V_AD = 0.0; ///< internal variable of quadrature
generator
double qsg_x_V_CD = 0.;

double Ic_al = 0.0;
double Ic_be = 0.0;
double V_AD_al = 0.0;
double V_AD_be = 0.0;
double V_CD_al = 0.0;
double V_CD_be = 0.0;
double Ia_al = 0.0;
double Ia_be = 0.0;
double Ic_d = 0.0;
double Ic_q = 0.0;
double sin_Ic_al_f;
double cos_Ic_al_f;
double Kp_lf = KP_LF;  ///< loop filter proportional gain
double Ki_lf = KI_LF;  ///< loop filter integral gain
double if_prop = 0.0;
double if_int = 0.0;
double if_int_sum = 0.0;
double if_y = 0.0;  ///< loop filter
double w_o_AD = 0.0;//314.0;  ///< angular frequency for Master
double w_o_CD = 0.0;//314.0;  ///< angular frequency for Slave
double w_o_CD_ref = 0.0;//314.0;
double P_AD = 0.0;
double P_CD = 0.0;
double Q_AD = 0.0;
double Q_CD = 0.0;
double pf_AD = 0.0;
double pf_CD = 0.0;
double Kp_pf_reg = 0.0;
double Ki_pf_reg = 0.0;
double Theta_comp = 0.0;

Uint32 phase_shift_b = 0L;
double phase_shift_b_double = 0.0;
double prev_phase_shift_b_double = 0.0;
//Uint32 phase_shift_b_unlimited = 0LU;
//Uint32 phase_shift_b_limited = 0L;
double prop_pf_reg = 0.0;
double int_pf_reg = 0.0;
double pf_CD_err = 0L;
double pf_CD_ref = 0.0;
Appendix B. DSP Source Code: Experimental System

double S_AD_unlimited = 0.0;
double S_AD_limited = 0.0;
double S_CD_unlimited = 0.0;
double S_CD_limited = 0.0;
unsigned int toggle_bit1 = 0;
unsigned int toggle_bit2 = 0;
Uint16 grab_count = 0;
Uint32 TransientCounter = 0;
Uint16 run_pf_reg = 0;
int16 pf_reg_action = 0;
Uint16 pf_reg_counter = 0;
Uint16 pf_reg_disable = 0;
Uint16 EnablePFReg = 0;

/** @name State Machine Definitions */
//@{
void st_vsi_init(void), ///< The state initialisation function
st_vsi_cal(void),    ///< Calibration state
st_vsi_stop(void)    ///< Waiting for start trigger
}
st_vsi_ramp(void), ///< Ramps up the output voltage
st_vsi_run(void), ///< Maintaining target output voltage
st_vsi_fault(void); ///< Wait for faults to clear

State_Type
vsi_state =
{
  &st_vsi_init, // state function ptr
  1 // first state flag
};

#define PSIM_VERSION

DLL_DYNAMIC_LIBRARYFUNCTIONS

void vsi_state_machine(void)
{
  // state execution locked out of PSIM execution for now - 17/9/2017
  // directly command the required state to execute in PSIM support code instead
  if (!defined PSIM_VERSION)
    DO_STATE(vsi_state);
  #endif

  if (adc.flag_rms != 0)
  {
    // rms calculations here
    adc_rms_scale();
    adc.flag_rms = 0;
  }
  else if (adc.flag_dc != 0)
  {
    // dc calculations here
    adc.flag_dc = 0;
    adc_scale_slow();
  }
  else if (adc.flag_cal != 0)
  {
    // Perform calibration of analog inputs
    adc.flag_cal = 0;
    adc_calibrate();
  }
} /* end vsi_state_machine */

void vsi_state_machine(void)
{
  // state execution locked out of PSIM execution for now - 17/9/2017
  // directly command the required state to execute in PSIM support code instead
  if (!defined PSIM_VERSION)
    DO_STATE(vsi_state);
  #endif

  if (adc.flag_rms != 0)
  {
    // rms calculations here
    adc_rms_scale();
    adc.flag_rms = 0;
  }
  else if (adc.flag_dc != 0)
  {
    // dc calculations here
    adc.flag_dc = 0;
    adc_scale_slow();
  }
  else if (adc.flag_cal != 0)
  {
    // Perform calibration of analog inputs
    adc.flag_cal = 0;
    adc_calibrate();
  }
} /* end vsi_state_machine */

void vsi_state_machine(void)
{
  // state execution locked out of PSIM execution for now - 17/9/2017
  // directly command the required state to execute in PSIM support code instead
  if (!defined PSIM_VERSION)
    DO_STATE(vsi_state);
  #endif

  if (adc.flag_rms != 0)
  {
    // rms calculations here
    adc_rms_scale();
    adc.flag_rms = 0;
  }
  else if (adc.flag_dc != 0)
  {
    // dc calculations here
    adc.flag_dc = 0;
    adc_scale_slow();
  }
  else if (adc.flag_cal != 0)
  {
    // Perform calibration of analog inputs
    adc.flag_cal = 0;
    adc_calibrate();
  }
} /* end vsi_state_machine */

void vsi_state_machine(void)
{
  // state execution locked out of PSIM execution for now - 17/9/2017
  // directly command the required state to execute in PSIM support code instead
  if (!defined PSIM_VERSION)
    DO_STATE(vsi_state);
  #endif

  if (adc.flag_rms != 0)
  {
    // rms calculations here
    adc_rms_scale();
    adc.flag_rms = 0;
  }
  else if (adc.flag_dc != 0)
  {
    // dc calculations here
    adc.flag_dc = 0;
    adc_scale_slow();
  }
  else if (adc.flag_cal != 0)
  {
    // Perform calibration of analog inputs
    adc.flag_cal = 0;
    adc_calibrate();
  }
} /* end vsi_state_machine */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * * * * */

This function can be used to switch the VSI from the stopped state to a running state. It is useful in a manually controlled system, but in the stand-alone production version, the VSI should leave the stop state automatically.

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * * * * */

This function can be used to switch the VSI from the stopped state to a running state. It is useful in a manually controlled system, but in the stand-alone production version, the VSI should leave the stop state automatically.

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * * * * */

This function can be used to switch the VSI from the stopped state to a running state. It is useful in a manually controlled system, but in the stand-alone production version, the VSI should leave the stop state automatically.
Appendix B. DSP Source Code: Experimental System

```c
/*
void vsi_enable(void)
{
    if (detected_faults == 0)
    {
        is_vsi_switching = 1;
    }
} /* end vsi_enable */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/**
This function can be used to switch the VSI from the running state to a stop state. It is useful in a manually controlled system, but in the stand-alone production version, there is no need to stop except in fault conditions.

\author A. McIver
\par History:
\li 01/05/07 DGH - derived from ele2.5kva/code/latest/cfpp.c */
void vsi_disable(void)
{
    is_vsi_switching = 0;

    // reset AD voltage controller PR states
    vac_an.s0 = 0;
    vac_an.s1 = 0;
    vac_an.s2 = 0;
    vac_an.y0 = 0;

    // reset CD voltage controller PR states
    vac_bn.s0 = 0;
    vac_bn.s1 = 0;
    vac_bn.s2 = 0;
    vac_bn.y0 = 0;

    // reset A current controller PR states
    iac_a.s0 = 0;
    iac_a.s1 = 0;
    iac_a.s2 = 0;
    iac_a.y0 = 0;

    // reset C current controller PR states
    iac_c.s0 = 0;
    iac_c.s1 = 0;
    iac_c.s2 = 0;
    iac_c.y0 = 0;

    // reset PLL loop filter parameters
    Ic_al = 0.0;
    Ic_be = 0.0;
    Ia_al = 0.0;
    Ia_be = 0.0;
    phase_a = 0L;
    phase_b = 0L;
    phase_c = PHASE_180DEG;
    Iq_int_sum = 0.0;
    phase_shift_b = 0L;
    V_AD_al = 0.0;
    V_AD_be = 0.0;
    V_CD_al = 0.0;
    V_CD_be = 0.0;
    int_pf_reg = 0.0;
} /* end vsi_disable */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/**
Set the target modulation depth.

\author A. McIver
\par History:
*/
```

Appendix B. DSP Source Code: Experimental System

```c
#define 01/05/07 DGH - derived from ele2.5kva\code\latest\cfpp.c

\param[in] v Target output voltage in hundredths of a %

\begin{verbatim}
void vsi_set_mod(Uint16 v)
{
    mod_ref = v;
} /* end vsi_set_mod */
\end{verbatim}

\param[in] v Target output current in tenths of an amp

\begin{verbatim}
void vsi_set_i(Uint16 i)
{
    i_ref = i;
} /* end vsi_set_i */
\end{verbatim}

void vsi_set_v_AD(Uint16 v_AD)
{
    v_ref_AD = v_AD;
} /* end vsi_set_v_AD */

void vsi_set_v_CD(Uint16 v_CD)
{
    v_ref_CD = v_CD;
} /* end vsi_set_v_CD */

\begin{verbatim}
int16 vsi_get_status(void)
{
    if (detected_faults != 0)
    {
        return -1;
    }
    else
    {
        return is_vsi_switching;
    }
} /* end vsi_get_status */
\end{verbatim}

\param[in] mode The desired operating mode

\begin{verbatim}
void vsi_set_mode(Uint16 mode)
{
\end{verbatim}
```
Appendix B. DSP Source Code: Experimental System

if (is_vsi_switching == 0)
{
    op_mode_vsi = mode;
}
} /* end vsi_set_mode */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * *
This function tests for the presence of the software tested fault. Tested faults are:
\li FAULT_PDPINT
\author A.McIver
\par History:
\li 31/07/07 - initial creation
\li 02/05/08 AM - added Iout trip

\returns Any faults detected.
*/
Uint16 vsi_check_fault(void)
{
    Uint16 faults = 0;
    // check for gate fault
    if (EvaRegs.COMCONA.bit.PDPINTASTATUS == 0)
    {
        faults |= FAULT_PDPINT;
    }
    return faults;
} /* end vsi_check_fault */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * *
Retrieve present operating state of vsi state machine.
\returns operating state of vsi state machine
*/
Uint16 vsi_get_state(void)
{
    Uint16 state;
    if (IS_CURRENT_STATE(vsi_state,st_vsi_run))
    {
        state = VSI_STATE_RUN;
    }
    else if ( (IS_CURRENT_STATE(vsi_state,st_vsi_init))
    || (IS_CURRENT_STATE(vsi_state,st_vsi_cal))
    || (IS_CURRENT_STATE(vsi_state,st_vsi_stop)) )
    {
        state = VSI_STATE_STOP;
    }
    else if (IS_CURRENT_STATE(vsi_state,st_vsi_ramp))
    {
        state = VSI_STATE_RAMP;
    }
    else if (IS_CURRENT_STATE(vsi_state,st_vsi_fault))
    {
        state = VSI_STATE_FAULT;
    }
    else
    {
        state = VSI_STATE_ERROR;
    }
    return state;
} /* end vsi_get_state */

/*
*=====================================================================*/
*/

__Interrupts()

=====================================================================
Appendix B. DSP Source Code: Experimental System

/* brief Updates VSI and stores ADC results */

This interrupt is triggered by the completion of the ADC conversions. It then:

- stores the ADC results
- applies the ADC calibration factors
- sums the calibration measurements
- applies a fast decaying average filter to the analog signals
- checks for fault conditions
- performs low speed averaging and rms calculations
- DC bus compensation
- updates phase angle
- calculates switching times
- centers pulses in switching period
- compares registers with switching times
- sets up analogs for next interrupt

#ifndef BUILD_RAM
#pragma CODE_SECTION(isr_pwm, "ramfuncs");
#endif

interrupt void isr_pwm(void)
{

dbg_var4++;

if(toggle_bit1)
{ SET_TP11();
  toggle_bit1 = 0;
}
else
{ CLEAR_TP11();
  toggle_bit1 = 1;
}

/* Find out the direction which the timers are going */

// 1 = PWM carrier is currently rising (underflow interrupt)
// 0 = PWM carrier is currently falling (period interrupt)
timer1_dir = EvaRegs.GPTCONA.bit.T1STAT;
timer3_dir = EvbRegs.GPTCONB.bit.T3STAT;

/* */

--

isr_pwm_ADC_calculations()
--

//--

// iac_a (store ADC results from previous cycle)
adc.iac_a.raw = (AdcRegs.ADCRESULT0>>4); //Ia

// iac_a (gain correction factor)
adc.iac_a.raw = (int16) ((int32)adc.iac_a.raw*(int32)cal_gainA) >> 14
- cal_offsetA - ADC_IAC_OFFSET - cal_offset_iac_a_dc;

// iac_b (store ADC results from previous cycle)
adc.iac_b.raw = (AdcRegs.ADCRESULT2>>4);

// iac_b (gain correction factor)
adc.iac_b.raw = (int16) ((int32)adc.iac_b.raw*(int32)cal_gainA) >> 14
- cal_offsetA - ADC_IAC_OFFSET - cal_offset_iac_b_dc;

// iac_c (store ADC results from previous cycle)
adc.iac_c.raw = (AdcRegs.ADCRESULT4>>4);

// iac_c (gain correction factor)
adc.iac_c.raw = (int16) ((int32)adc.iac_c.raw*(int32)cal_gainA) >> 14
- cal_offsetA - ADC_IAC_OFFSET - cal_offset_iac_c_dc;

adc.idc.raw = (AdcRegs.ADCRESULT6>>4);

// idc (gain correction factor)
adc.idc.raw = (int16) ((int32)adc.idc.raw*(int32)cal_gainA) >> 14
- cal_offsetA - ADC_IDC_OFFSET - cal_offset_idc;
Appendix B. DSP Source Code: Experimental System

```c
// vac1 (store ADC results from previous cycle)
adc.hvdc1.raw = (AdcRegs.ADCRESULT7>>4);
// hvdc1 (gain correction factor)
adc.hvdc1.raw = (int16)((int32)adc.hvdc1.raw*(int32)cal_gainB) >> 14
   - cal_offsetB - ADC_VDC_OFFSET - cal_offset_hvdc1;

// hvdc2 (store ADC results from previous cycle)
adc.hvdc2.raw = (AdcRegs.ADCRESULT1>>4);
// hvdc2 (gain correction factor)
adc.hvdc2.raw = (int16)((int32)adc.hvdc2.raw*(int32)cal_gainB) >> 14
   - cal_offsetB - ADC_VDC_OFFSET - cal_offset_hvdc2;

// hvdc (store ADC results from previous cycle)
adc.hvdc.raw = (AdcRegs.ADCRESULT9>>4);
// hvdc (gain correction factor)
adc.hvdc.raw = (int16)((int32)adc.hvdc.raw*(int32)cal_gainB) >> 14
   - cal_offsetB - ADC_VDC_OFFSET - cal_offset_hvdc;

adc.vac_an.raw = (AdcRegs.ADCRESULT5>>4);
// vac_an (gain correction factor)
adc.vac_an.raw = (int16)((int32)adc.vac_an.raw*(int32)cal_gainB) >> 14
   - cal_offsetB - ADC_VAC_OFFSET - cal_offset_vac_an;

vAD = adc.vac_an.raw;
vCD = adc.vac_bn.raw;
vAB = adc.vac_an.raw - adc.vac_bn.raw; // vAB = vAC
```

```c
// calibration from references
adc.yHA.dc_sum += (Uint32)(AdcRegs.ADCRESULT12>>4);
adc.yLA.dc_sum += (Uint32)(AdcRegs.ADCRESULT14>>4);
adc.yHB.dc_sum += (Uint32)(AdcRegs.ADCRESULT13>>4);
adc.yLB.dc_sum += (Uint32)(AdcRegs.ADCRESULT15>>4);
adc.count_cal++;

if (adc.count_cal > ADC_COUNT_CAL)
{
    adc.count_cal = 0;
    adc.yHA.dc_sum_bak = adc.yHA.dc_sum;
    adc.yLA.dc_sum_bak = adc.yLA.dc_sum;
    adc.yHB dc_sum_bak =adc.yHB.dc_sum;
    adc.yLB.dc_sum_bak =adc.yLB.dc_sum;
    adc.yHA.dc_sum = 0;
    adc.yLA.dc_sum = 0;
    adc.yHB.dc_sum = 0;
    adc.yLB.dc_sum = 0;
    adc.flag_cal = 1;
}

// Reinitialize for next ADC sequence -
// AdcRegs.ADCCTRL2.bit.RST_SEQ1 = 1; // Reset SEQ1
```

```c
// fast filter ADC results)
adc.hvdc1.filt = (3*adc.hvdc1.filt + adc.hvdc1.raw + 2)>>2;
adc.hvdc2.filt = (3*adc.hvdc2.filt + adc.hvdc2.raw + 2)>>2;
adc.hvdc.filt = (3*adc.hvdc.filt + adc.hvdc.raw + 2)>>2;
adc.iac_b.filt = (3*adc.iac_b.filt + adc.iac_b.raw + 2)>>2;
adc.iac_c.filt = (3*adc.iac_c.filt + adc.iac_c.raw + 2)>>2;
adc.idc.filt = (3*adc.idc.filt + adc.idc.raw + 2)>>2;
adc.vac_an.filt = (3*adc.vac_an.filt + adc.vac_an.raw + 2)>>2;
adc.vac_bn.filt = (3*adc.vac_bn.filt + adc.vac_bn.raw + 2)>>2;
```
isr_pwm_low_speed_average()

// DC summation calculations
adc.iac_a_dc.dc_sum += (int32)adc.iac_a_dc.filt;
adc.iac_b_dc.dc_sum += (int32)adc.iac_b_dc.filt;
adc.iac_c_dc.dc_sum += (int32)adc.iac_c_dc.filt;
adc.vac_an_dc.dc_sum += (int32)adc.vac_an_dc.filt;
adc.vac_bn_dc.dc_sum += (int32)adc.vac_bn_dc.filt;
adc.hvdc1.dc_sum += (int32)adc.hvdc1.filt;
adc.hvdc2.dc_sum += (int32)adc.hvdc2.filt;
adc.hvdc.dc_sum += (int32)adc.hvdc.filt;
adc.idc.dc_sum += (int32)adc.idc.filt;
adc.count_dc++; 
if (adc.count_dc >= COUNT_DC_IN) {
    adc.iac_a_dc.dc_sum_bak = adc.iac_a_dc.dc_sum;
    adc.iac_a.dc_sum = 0;
    adc.iac_b_dc.dc_sum_bak = adc.iac_b_dc.dc_sum;
    adc.iac_b.dc_sum = 0;
    adc.iac_c_dc.dc_sum_bak = adc.iac_c_dc.dc_sum;
    adc.iac_c.dc_sum = 0;
    adc.vac_an_dc.dc_sum_bak = adc.vac_an_dc.dc_sum;
    adc.vac_an_dc.dc_sum = 0;
    adc.vac_bn_dc.dc_sum_bak = adc.vac_bn_dc.dc_sum;
    adc.vac_bn_dc.dc_sum = 0;
    adc.hvdc1.dc_sum_bak = adc.hvdc1.dc_sum;
    adc.hvdc1.dc_sum = 0;
    adc.hvdc2.dc_sum_bak = adc.hvdc2.dc_sum;
    adc.hvdc2.dc_sum = 0;
    adc.hvdc.dc_sum_bak = adc.hvdc.dc_sum;
    adc.hvdc.dc_sum = 0;
    adc.idc.dc_sum_bak = adc.idc.dc_sum;
    adc.idc.dc_sum = 0;
    adc.count_dc = 0;
    adc.flag_dc = 1;
} 

// RMS summation calculations
adc.iac_a.rms_sum += ((int32)(adc.iac_a.filt*adc.iac_a.filt)>>ADC_RMS_PS);
adc.iac_a.dc_sum += (int32)adc.iac_a.filt;
adc.iac_b.rms_sum += ((int32)(adc.iac_b.filt*adc.iac_b.filt)>>ADC_RMS_PS);
adc.iac_b.dc_sum += (int32)adc.iac_b.filt;
adc.iac_c.rms_sum += ((int32)(adc.iac_c.filt*adc.iac_c.filt)>>ADC_RMS_PS);
adc.iac_c.dc_sum += (int32)adc.iac_c.filt;
adc.vac_an.rms_sum += ((int32)(adc.vac_an.filt*adc.vac_an.filt)>>ADC_RMS_PS);
adc.vac_an.dc_sum += (int32)adc.vac_an.filt;
adc.vac_bn.rms_sum += ((int32)(adc.vac_bn.filt*adc.vac_bn.filt)>>ADC_RMS_PS);
adc.vac_bn.dc_sum += (int32)adc.vac_bn.filt;
adc.count_rms++; 
if (phase_a < phase_step_a) {
    adc.iac_a.rms_sum_bak = adc.iac_a.rms_sum;
    adc.iac_a.rms_sum = 0;
    adc.iac_b.rms_sum_bak = adc.iac_b.rms_sum;
    adc.iac_b.rms_sum = 0;
    adc.iac_c.rms_sum_bak = adc.iac_c.rms_sum;
    adc.iac_c.rms_sum = 0;
    adc.vac_an.rms_sum_bak = adc.vac_an.rms_sum;
    adc.vac_an.rms_sum = 0;
    adc.vac_bn.rms_sum_bak = adc.vac_bn.rms_sum;
    adc.vac_bn.rms_sum = 0;
} 

adc.count_rms++; 
// only update rms sum over full cycle 
if (phase_a < phase_step_a) {
    // store sums for background processing
    adc.iac_a.rms_sum_bak = adc.iac_a.rms_sum;
    adc.iac_a.rms_sum = 0;
    adc.iac_a.dc_sum_bak = adc.iac_a.dc_sum;
    adc.iac_a.dc_sum = 0;
    adc.iac_b.rms_sum_bak = adc.iac_b.rms_sum;
    adc.iac_b.rms_sum = 0;
    adc.iac_b.dc_sum_bak = adc.iac_b.dc_sum;
    adc.iac_b.dc_sum = 0;
    adc.iac_c.rms_sum_bak = adc.iac_c.rms_sum;
    adc.iac_c.rms_sum = 0;
    adc.iac_c.dc_sum_bak = adc.iac_c.dc_sum;
    adc.iac_c.dc_sum = 0;
    adc.vac_an.rms_sum_bak = adc.vac_an.rms_sum;
    adc.vac_an.rms_sum = 0;
    adc.vac_bn.rms_sum_bak = adc.vac_bn.rms_sum;
    adc.vac_bn.rms_sum = 0;
}
Appendix B. DSP Source Code: Experimental System

```c
adc.vac_an.dc_sum_bak = adc.vac_an.dc_sum;
adc.vac_an.dc_sum = 0L;
adc.vac_bn.rms_sum_bak = adc.vac_bn.rms_sum;
adc.vac_bn.rms_sum = 0L;
adc.vac_bn.dc_sum_bak = adc.vac_bn.dc_sum;
adc.vac_bn.dc_sum = 0L;
adc.count_rms_bak = adc.count_rms;
adc.count_rms = 0;
adc.flag_rms = 1;
}
/*
---------------------------------------------------------------------
==
void isr_pwm_code()
---------------------------------------------------------------------
*/
pwm_int_count++;
/* update phase angle */
phase_a += phase_step_a;
index_a = (phase_a>>22)|0x0001; // to access high word of 32 bit
sine table
phase_b += phase_step_b;
index_b = (phase_b>>22)|0x0001; // to access high word of 32 bit
sine table

// if(phase_c < phase_step_b)
if(phase_c < PHASE_180DEG < phase_step_b) // needs strategy to run
only once within a fundamental cycle
{
    if(pf_reg_disable == 0)
    {
        if(toggle_bit1) toggle_bit1 = 0; else toggle_bit1 = 1;
    }
    else if(pf_reg_action == 1)
    {  
        phase_c = phase_c + phase_step_b + phase_shift_b;
    }
    else if(pf_reg_action == -1)
    {  
        phase_c = phase_c + phase_step_b - phase_shift_b;
    }  
    else  
    {  
        phase_c = phase_c + phase_step_b;
    }  
    index_c = ((phase_c)>>22)|0x0001; // to access high word of 32
bit sine table
    run_pf_reg = 1;
}
else
{
    phase_c = phase_c + phase_step_b;
    index_c = ((phase_c)>>22)|0x0001; // to access high word of 32
bit sine table
    pf_reg_disable = 1;
}
else
{
    phase_c = phase_c + phase_step_b;
    index_c = ((phase_c)>>22)|0x0001; // to access high word of 32
bit sine table
}

// if(pf_reg_action == 1)
// phase_c = phase_c + phase_step_b + phase_shift_b;
// else if(pf_reg_action == -1)
// phase_c = phase_c + phase_step_b - phase_shift_b;
// else
// phase_c = phase_c + phase_step_b;
```
Appndix B. DSP Source Code: Experimental System

// index_c = (phase_c)>>22)|0x0001; // to access high word of 32 bit sine table
//
// else
//
// phase_c = phase_c + phase_step_b;
// index_c = (phase_c)>>22)|0x0001; // to access high word of 32 bit sine table
//
// change code below to operate on Ic instead of Ia

phase_offset = (phase_a&0x007F0000L)>>16;
// interpolate more accurate sin value for phase A
Ic_al_lo = sin_table[index_b];
Ic_al_diff = sin_table[index_b+2] - Ic_al_lo;
sin_Ic_al = Ic_al_lo +
(int16)(((int32)phase_offset*(int32)Ic_al_diff)>>7);

// interpolate cos value
index_b += TABLE_OFFSET_90;
Ic_al_lo = sin_table[index_b];
Ic_al_diff = sin_table[index_b+2] - Ic_al_lo;
cos_Ic_al = Ic_al_lo +
(int16)(((int32)phase_offset*(int32)Ic_al_diff)>>7);

// convert sine and cos values from integer to floating point
sin_Ic_al_f = (double)sin_Ic_al/16384.0;
cos_Ic_al_f = (double)cos_Ic_al/16384.0;
Ic = ADC_IAC_SC * (double)adc.iac_c.raw; //

// Quadrature Signal Generator (SOGI-QSG)
qsg_x_Ic = (Ic - Ic_al)*QSG_K; //
Ic_al += (qsg_x_Ic - Ic_be)*w_o_CD*T_S; //For now consider nominal frequency

Ic_be += Ic_al*w_o_CD*T_S;

// Parks transformation
// Ic_d = Ic_al*cos_Ic_al_f + Ic_be*sin_Ic_al_f;
// Ic_q = -Ic_al*sin_Ic_al_f + Ic_be*cos_Ic_al_f;

// Loop Filter (LF)
lf_prop = Ic_q*Kp_lf;
lf_int =Ic_q*Ki_lf;
lf_int_sum += lf_int;
lf_y = lf_prop + lf_int_sum;

// Alpha_Beta transformation for Power Calculation
// For Master DG current_Ia
Ia = ADC_IAC_SC * (double)adc.iac_a.raw;
// Quadrature Signal Generator (SOGI-QSG)
qsg_x_Ia = (Ia - Ia_al)*QSG_K; //
Ia_al += (qsg_x_Ia - Ia_be)*w_o_CD*T_S;
Ia_be += Ia_al*w_o_CD*T_S;

// For string voltage_VAD
V_AD = ADC_VAC_SC * (double)adc.vac_an.raw;
qsg_x_V_AD = (V_AD - V_AD_al)*QSG_K; //
V_AD_al += (qsg_x_V_AD - V_AD_be)*w_o_AD*T_S;
V_AD_be += V_AD_al*w_o_AD*T_S;

// For slave voltage_VCD
V_CD = ADC_VAC_SC * (double)adc.vac_bn.raw;
qsg_x_V_CD = (V_CD - V_CD_al)*QSG_K; //
V_CD_al += (qsg_x_V_CD - V_CD_be)*w_o_CD*T_S;
V_CD_be += V_CD_al*w_o_CD*T_S;

// Update frequency
phase_step_a = phase_step_a_new;

//w_o = W_N; // without PLL
\[ w_{o_CD} = W_N + f_y; \quad \text{// with PLL} \]
\[ w_{o_CD} = W_N + f_{int\_sum}; \]
\[ \text{phase\_step\_b\_new} = (\text{int32}) (w_{o_CD} * (1.0/2.0/\_\_\_PI*65536.0*65536.0/VSI\_FIN)); \]
\[ \text{phase\_step\_b} = \text{phase\_step\_b\_new}; \]

\[ \text{phase\_step\_b} = \text{phase\_step\_a\_new}; \quad \text{// slave takes freq from droop instead of PLL} \]

```c
w_o_CD = W_N + f_y; // with PLL
w_o_CD = W_N + f_int_sum;
phase_step_b_new = (int32) (w_o_CD * (1.0/2.0_/_/PI*65536.0*65536.0/VSI_FIN));
phase_step_b = phase_step_b_new;

//phase_step_b = phase_step_a_new; // slave takes freq from droop instead of PLL
```

/*
=======================================================================
==
Closed Loop Current Regulation Code
=======================================================================
=======*/

// H-bridge AB (Master)
// Voltage control loop AD
vac_an.targ_adc = (((long)sin_table[index_a]*(long)v_isr_AD)>>14);
vac_an.err = (int32)((vac_an.targ_adc - adc.vac_an.raw)<<ERR_SH_VAC_AD);
if(iref_ad >= I_POS_SAT_ADCCNT)
    debug_var1 = 1;
else
    debug_var1 = 0;
if(iref_ad <= I_NEG_SAT_ADCCNT)
    debug_var2 = 1;
else
    debug_var2 = 0;

// Anti-Windup for integrators
if ((iref_ad >= I_POS_SAT_ADCCNT) || (iref_ad <= I_NEG_SAT_ADCCNT))
    vac_an.err_res = 0;
else
    {
        vac_an.err_res = vac_an.err;
    }

vac_an.s2 += (vac_an.s1 + DELTA_R)>>DELTA_Q;
vac_an.s1 += (vac_an.s0 + DELTA_R)>>DELTA_Q;
vac_an.s0 = ((int16)((int32)vac_an.err_res -
    ((int32)((int32)vac_an.alpha1*(int32)vac_an.s1 +
    (int32)vac_an.alpha2*(int32)vac_an.s2
    + (int32)ALPHA0_R)>>ALPHA0_Q)));
vac_an.y0 = (int16)((int32)((int32)vac_an.s0*(int32)vac_an.beta0 +
    (int32)vac_an.s1*(int32)vac_an.beta1 + (int32)ALPHA0_R)>>ALPHA0_Q);

iref_ad = (int16)((int32)((int32)vac_an.err + (int32)vac_an.y0) *
    (int32)vac_an.Kp
    +(int32)P_ERR_R_VAC_AD) >> (P_SHIFT_VAC_AD+ERR_SH_VAC_AD);
if (op_mode_vsi == VSI_CURR)
    {
        iac_a.targ_adc = (((long)sin_table[index_a]*(long)i_isr)>>14);
    }
else if(op_mode_vsi == VSI_VOLT)
    {
        iac_a.targ_adc = iref_ad;
    }

iac_a.err = (int32)((iac_a.targ_adc - adc.iac_a.raw)<<ERR_SH_IAC_AB);

// Anti-Windup for integrators
if (V_Asat||V_Bsat)
    {
        iac_a.err_res = 0;
    }
else
    {
        iac_a.err_res = iac_a.err;
    }
```
Appendix B. DSP Source Code: Experimental System

iac_a.s2 += (iac_a.s1 + DELTA_R) >> DELTA_Q;
iac_a.s1 += (iac_a.s0 + DELTA_R) >> DELTA_Q;
iac_a.s0 = (int16)((int32)iac_a.err_res - ((int32)(iac_a.alpha1 * (int32)iac_a.s1 + (int32)iac_a.alpha2 * (int32)iac_a.s2 + (int32)ALPHA0_R) >> ALPHA0_Q));

iac_a.y0 = (int16)((int32)(int32)iac_a.s0 * (int32)iac_a.beta0 + (int32)iac_a.s1 * (int32)iac_a.beta1 + (int32)ALPHA0_R) >> ALPHA0_Q);
t_a_cnt = (int16)((int32)(iac_a.err + (int32)iac_a.y0) * (int32)iac_a.Kp + (int32)P_ERR_R_IAC_AB) >> (P_SHIFT_IAC_AB + ERR_SH_IAC_AB);
t_b_cnt = ~t_a_cnt;

// H-bridge CD (Slave)

// Voltage control loop CD
//vac_bn.targ_adc = (((long)sin_table[index_b] * (long)v_isr_CD) >> 14); // power factor regulator is disabled
vac_bn.targ_adc = (((long)sin_table[index_c] * (long)v_isr_CD) >> 14); // power factor regulator is enabled

vac_bn.err = (int32)((vac_bn.targ_adc - adc.vac_bn.raw) << ERR_SH_VAC_CD);

// Anti-Windup for integrators
if ((iref_cd >= I_POS_SAT_ADCCNT) || (iref_cd <= I_NEG_SAT_ADCCNT))
{
  vac_bn.err_res = 0;
}
else
{
  vac_bn.err_res = vac_bn.err;
}

vac_bn.s0 = (int16)((int32)vac_bn.err_res - ((int32)vac_bn.alphai * (int32)vac_bn.s1 + (int32)vac_bn.alphal * (int32)vac_bn.s2 + (int32)ALPHA0_R) >> ALPHA0_Q));
vac_bn.y0 = (int16)((int32)(vac_bn.s0 * (int32)vac_bn.beta0 + (int32)vac_bn.s1 * (int32)vac_bn.beta1 + (int32)ALPHA0_R) >> ALPHA0_Q);

iref_cd = (int16)((int32)vac_bn.err + (int32)vac_bn.y0) * (int32)vac_bn.Kp + (int32)P_ERR_R_VAC_CD) >> (P_SHIFT_VAC_CD + ERR_SH_VAC_CD);

if (op_mode_vsi == VSI_CURR)
{
  iac_c.targ_adc = (((long)sin_table[index_a] * (long)i_isr) >> 14);
}
else if (op_mode_vsi == VSI_VOLT)
{
  iac_c.targ_adc = iref_cd;
}

iac_c.err = (int32)((iac_c.targ_adc - adc.iac_c.raw) << ERR_SH_IAC_CD);

// Anti-Windup for integrators
if (V_Csat || V_Dsat)
{
  iac_c.err_res = 0;
}
else
{
  iac_c.err_res = iac_c.err;
}

iac_c.s2 += (iac_c.s1 + DELTA_R) >> DELTA_Q;
iac_c.s1 += (iac_c.s0 + DELTA_R) >> DELTA_Q;
iac_c.s0 = (int16)((int32)iac_c.err_res - ((int32)iac_c.alphai * (int32)iac_c.s1 + (int32)iac_c.alphal * (int32)iac_c.s2 + (int32)ALPHA0_R) >> ALPHA0_Q));
Appendix B. DSP Source Code: Experimental System

```c
+ (int32)ALPHA0_R)>>ALPHA0_Q));
iac_c.y0 = (int16)((int32)( (int32)iac_c.s0*(int32)iac_c.beta0 +
   (int32)iac_c.s1*(int32)iac_c.beta1 + (int32)ALPHA0_R)>>ALPHA0_Q);
t_c_cnt = (int16)(( (int32)iac_c.err + (int32)iac_c.y0 ) *
   (int32)iac_c.Kp
   +(int32)P_ERR_R_IAC_CD ) >> (P_SHIFT_IAC_AB+ERR_SH_IAC_CD) );
t_d_cnt = -t_c_cnt;

//Calculate scale factor for DC bus compensation of H-bridge AB
//DC bus compensation = mod_depth * nominal_DC_bus/real_DC_bus
//DC Bus compensation only applies above the minimum DC Bus voltage
if (adc.hvdc2.filt > MIN_ADC_VDC_AB_COMP)
{
   adc_vdc_comp_AB = adc.hvdc2.filt;
} else
{
   adc_vdc_comp_AB = MIN_ADC_VDC_AB_COMP;
}
// Scaled by DC_COMP_SHIFT to enable it to remain an integer
DCBUS_comp_AB = (int32)(VHI_AB_NOM_COUNT_SHFT/adc_vdc_comp_AB);
// Apply DC bus compensation to switching times calculated by current
// regulator
//t_a_cnt = (int16)(((int32)(t_a_cnt) * DCBUS_comp_AB) >>
//   AB_COMP_SHIFT);
//t_b_cnt = (int16)(((int32)(t_b_cnt) * DCBUS_comp_AB) >>
//   AB_COMP_SHIFT);

//Calculate scale factor for DC bus compensation of H-bridge CD
//DC bus compensation = mod_depth * nominal_DC_bus/real_DC_bus
//DC Bus compensation only applies above the minimum DC Bus voltage
if (adc.hvdc2.filt > MIN_ADC_VDC_CD_COMP)
{
   adc_vdc_comp_CD = adc.hvdc2.filt;
} else
{
   adc_vdc_comp_CD = MIN_ADC_VDC_CD_COMP;
}
// Scaled by DC_COMP_SHIFT to enable it to remain an integer
DCBUS_comp_CD = (int32)(VHI_CD_NOM_COUNT_SHFT/adc_vdc_comp_CD);
// Apply DC bus compensation to switching times calculated by current
// regulator
//t_c_cnt = (int16)(((int32)(t_c_cnt) * DCBUS_comp_CD) >>
//   CD_COMP_SHIFT);
//t_d_cnt = (int16)(((int32)(t_d_cnt) * DCBUS_comp_CD) >>
//   CD_COMP_SHIFT);
```

Select Modulation Source

```c
/*
====================================================================
Select Modulation Source
====================================================================*/
if (op_mode_vsi == VSI_OL)
{
   // use open loop modulation command
   t_A = ((int32)sin_table[index_a]*(int32)mod_isr)>>14;
   t_B = -t_A;
   t_C = ((int32)sin_table[index_a]*(int32)mod_isr)>>14;
   t_D = -t_C;
} else if ((op_mode_vsi == VSI_CURR) || (op_mode_vsi == VSI_VOLT))
{
   // use closed loop modulation command
   t_A = t_a_cnt;
   t_B = t_b_cnt;
   t_C = t_c_cnt;
   t_D = t_d_cnt;
}
```
Appendix B. DSP Source Code: Experimental System

```c
/* calculate t_C from t_A and t_B */
// t_C = -t_A - t_B;

ISR_pwm_desaturation()

/* Phase A - CMPR4  Phase C - CMPR1
   Phase B - CMPR5  Phase D - CMPR2 */

// Phase A
V_Asat_prev = V_Asat;
if (t_A >= MAX_TIME)
  { V_Asat = POS_SAT;
    EvbRegs.CMPR4 = 0;
  }
else if (t_A <= -MAX_TIME)
  { if((V_Asat==NEG_SAT) || (timer3_dir==0))
    { EvbRegs.CMPR4 = period;
    }
    else
    { EvbRegs.CMPR4 = period - 1;
      V_Asat = NEG_SAT;
    }
  }
else
  { V_Asat = NOT_SAT;
    EvbRegs.CMPR4 = period_2 - t_A;
  }

// B phase
V_Bsat_prev = V_Bsat;
if (t_B >= MAX_TIME)
  { V_Bsat = POS_SAT;
    EvbRegs.CMPR5 = 0;
  }
else if (t_B <= -MAX_TIME)
  { if((V_Bsat==NEG_SAT) || (timer3_dir==0))
    { EvbRegs.CMPR5 = period;
    }
    else
    { EvbRegs.CMPR5 = period - 1;
      V_Bsat = NEG_SAT;
    }
  }
else
  { V_Bsat = NOT_SAT;
    EvbRegs.CMPR5 = period_2 - t_B;
  }

//Phase C
V_Csat_prev = V_Csat;
if (t_C >= MAX_TIME)
  { V_Csat = POS_SAT;
    EvaRegs.CMPR1 = 0;
  }
else if (t_C <= -MAX_TIME)
  { if((V_Csat==NEG_SAT) || (timer1_dir==0))
    { EvaRegs.CMPR1 = period;
    }
    else
    { EvaRegs.CMPR1 = period - 1;
      V_Csat = NEG_SAT;
    }
  }
else
  { V_Csat = NOT_SAT;
    EvaRegs.CMPR1 = period_2 - t_C;
  }

// B phase
V_Bsat_prev = V_Bsat;
if (t_B >= MAX_TIME)
  { V_Bsat = POS_SAT;
    EvbRegs.CMPR5 = 0;
  }
else if (t_B <= -MAX_TIME)
  { if((V_Bsat==NEG_SAT) || (timer3_dir==0))
    { EvbRegs.CMPR5 = period;
    }
    else
    { EvbRegs.CMPR5 = period - 1;
      V_Bsat = NEG_SAT;
    }
  }
else
  { V_Bsat = NOT_SAT;
    EvbRegs.CMPR5 = period_2 - t_B;
  }
```

140
else
{
    V_Csat = NOT_SAT;
    EvaRegs.CMPR1 = period_2 - t_C;
}

// Phase D
V_Dsat_prev = V_Dsat;
if (t_D >= MAX_TIME)
{
    V_Dsat = POS_SAT;
    EvaRegs.CMPR2 = 0;
} else if (t_D <= -MAX_TIME)
{
    if((V_Dsat == NEG_SAT) || (timer3_dir == 0))
    {
        EvaRegs.CMPR2 = period;
    }
    else
    {
        EvaRegs.CMPR2 = period - 1;
    }
    V_Dsat = NEG_SAT;
} else
{
    V_Dsat = NOT_SAT;
    EvaRegs.CMPR2 = period_2 - t_D;
}

if(grab_count == GRAB_DEC)
{
    grab_count = 0;

    if (GrabRunning())
    {
        grab_dec = 0;
        // GrabStore(0,pf_AD);
        // GrabStore(1,pf_CD_ref);
        // GrabStore(2,pf_CD);
        // GrabStore(3,F_AD);
        // GrabStore(4,F_CD);
        // GrabStore(5,Q_AD);
        // GrabStore(6,Q_CD);
        // GrabStore(7,w_o_CD);
        GrabStore(0,pwm_int_count);
        GrabStore(1,pf_AD);
        GrabStore(2,pf_CD_ref);
        GrabStore(3,pf_CD);
        GrabStore(4,Theta_comp);
        GrabStore(5,(double)phase_shift_b);
        GrabStore(3,w_o_AD);
        GrabStore(4,Theta_comp);
        GrabStore(5,cal_offset_iac_a_dc);
        GrabStore(6,cal_offset_iac_c_dc);
        GrabStore(7,cal_offset_vac_an_dc);
        GrabStore(8,cal_offset_vac_bn_dc);
        GrabStep();
    }
}
#endif
// prepare for next interrupt
AxcRegs.ADCST.bit.INT_SEQ1_CLR = 1; // clear interrupt flag
PieCtrlRegs.PIEACK.all = PIEACK_GROUP1; // Acknowledge interrupt to PIE
/* end isr_pwm */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* * * * */
/** Handles the PDPINT interrupt caused by a gate fault. */

#ifndef BUILD_RAM
#pragma CODE_SECTION(isr_pdpint, "ramfuncs");
#endif
interrupt void isr_pdpint(void)
{
  VSI_FAST_STOP();
  is_vsi_switching = 0;
  detected_faults |= FAULT_PDPINT;

  EvaRegs.EVAIFRA.all = BIT0;  
  // Acknowledge this interrupt to receive more interrupts from group 1
  PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
} /* end isr_pdpint */

/**
 * _State_Functions()
 */

---

```c

// This function initialises the ADC, the PWM, and LEDs.
// It should
// -# reset the target output voltage to zero
// -# make sure that any soft charge relays and contactors are in the correct state
// -# initialize all variables that require resetting (such as RMS values).

void st_vsi_init(void)
{
  if (IS_FIRST_STATE(vsi_state))
  {
    DONE_FIRST_STATE(vsi_state);

    adc_init();
    pwm_init(); // initialises VSI
    VSI_FAST_STOP();
  }
  NEXT_STATE(vsi_state, st_vsi_cal);
} /* end st_vsi_init */

/**
 * The current inputs from LEMs can have significant offsets due to tolerance variations between the LEMs and the op amp inputs. This function provides a wait state where the user can perform zeroing calibration on any LEM inputs before the system is switching.

void st_vsi_cal(void)
{
  if (IS_FIRST_STATE(vsi_state))
  {
    DONE_FIRST_STATE(vsi_state);

    wd_timer[WD_CHARGE] = CAL_DELAY;
  }
  if (wd_timer[WD_CHARGE] == 0)
  {
    // Acknowledge this interrupt to receive more interrupts from group 1
    PicCtrlRegs.PIEACK.all = PIEACK_GROUP1;
  } /* end st_vsi_cal */
```

---
Appendix B. DSP Source Code: Experimental System

```c
{%
// cal_offset_vac_an_dc = (int16)(adc.vac_an_dc.real/ADC_VAC_SC);
// cal_offset_vac_bn_dc = (int16)(adc.vac_bn_dc.real/ADC_VAC_SC);
NEXT_STATE(vsi_state, st_vsi_stop);
%
} /* end st_vsi_cal */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/**
This is the state where the VSI is stopped. There is no switching. In its automatic production configuration, the operation waits in this state for 1 second after the DC bus volts reach the starting value before moving to the ramp up state. During the wait, this state checks whether any faults have been detected and if so, moves to the fault state.
*/

void st_vsi_stop(void)
{
    if (IS_FIRST_STATE(vsi_state))
    {
        DONE_FIRST_STATE(vsi_state);
        VSI_FAST_STOP(); // turn off outputs
        is_vsi_switching = 0;
        puts_COM0("s");
    }

    // check for faults
    if (detected_faults != 0)
    {
        NEXT_STATE(vsi_state, st_vsi_fault);
        return;
    }

    if (is_vsi_switching != 0)
    {
        NEXT_STATE(vsi_state, st_vsi_ramp);
    }
} /* end st_vsi_stop */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/**
This function starts the VSI switching and ramps the target output voltage from the initial output voltage up to the reference output voltage in constant voltage mode.

This ramp rate is determined by the step size STEP_VDC and the calling frequency of this state machine which is assumed to be 1msec.

If the VSI is stopped or the DC bus volts drop too low then the next state is the stop state. If a fault is detected the next state is the fault state.
Otherwise, once the target reaches the reference, the next state is the run state.
*/

void st_vsi_ramp(void)
{
    if (IS_FIRST_STATE(vsi_state))
    {
        DONE_FIRST_STATE(vsi_state);
        VSI_ENABLE();
        mod_targ = 0;
        puts_COM0("u");
```
// check for faults
if (detected_faults != 0)
{
    NEXT_STATE(vsi_state,st_vsi_fault);
    return;
}
// check for stop signal or inadequate dc bus voltage
if (is_vsi_switching == 0)
{
    NEXT_STATE(vsi_state,st_vsi_stop);
    return;
}
// check for target reached
if (mod_targ >= mod_ref)
{
    NEXT_STATE(vsi_state,st_vsi_run);
}
 else
{
    mod_targ += STEP_MOD_TARG;
    mod_isr = scale_mod(mod_targ);
    i_targ = i_ref;
    i_isr = scale_iref(i_targ);
    v_targ_AD = v_ref_AD;
    v_isr_AD = scale_vref_AD(v_targ_AD);
    v_targ_CD = v_ref_CD;
    v_isr_CD = scale_vref_CD(v_targ_CD);
}
} /* end st_vsi_ramp */

/**
 * In this state, the VSI is running and following the reference. If the
 * VSI is stopped or the DC bus volts drop too low then the next state is the
 * stop state. If a fault is detected the next state is the fault state.
 */

void st_vsi_run(void)
{
    if (IS_FIRST_STATE(vsi_state))
    {
        DONE_FIRST_STATE(vsi_state);
        puts_COM0("r");
    }
    // check for faults
    if (detected_faults != 0)
    {
        NEXT_STATE(vsi_state,st_vsi_fault);
        return;
    }
    // check for stop signal or inadequate dc bus voltage
    if (is_vsi_switching == 0)
    {
        NEXT_STATE(vsi_state,st_vsi_stop);
        return;
    }
    // follow changes in reference
    mod_targ = mod_ref;
    mod_isr = scale_mod(mod_targ);
    i_targ = i_ref;
    i_isr = scale_iref(i_targ);
    v_targ_AD = v_ref_AD;
    v_isr_AD = scale_vref_AD(v_targ_AD);
    v_targ_CD = v_ref_CD;
    v_isr_CD = scale_vref_CD(v_targ_CD);
}
} /* end st_vsi_run */
/**
 * This state is entered when a fault condition has been detected. It is
 * not left
 * until all the faults have been cleared by the background fault
 * handling
 * system.
 */

void st_vsi_fault(void)
{
    if (IS_FIRST_STATE(vsi_state))
    {
        DONE_FIRST_STATE(vsi_state);
        VSI_FAST_STOP(); // turn off outputs
        puts_COM0("f");
    }

    if (detected_faults == 0)
    {
        NEXT_STATE(vsi_state,st_vsi_init);
    }
} /* end st_vsi_fault */

/*
 __Local_Functions()
 ========================================================================
======= */

/// Scale mod depth from 0.01% to suit ISR use
int16 scale_mod(Uint16 m)
{
    int16 mod;
    mod = (int16)( (double)m*PERIOD_2/MOD_100);
    return mod;
} /* end scale_mod */

// Scale current reference from 0.1A to suit ISR use
int16 scale_iref(Uint16 i)
{
    int16 iref;
    iref = (int16)( (double)i/ADC_IAC_SC/IREF_10);
    return iref;
} /* end scale_mod */

// Scale voltage reference to suit ISR use
int16 scale_vref_AD(Uint16 v_AD)
{
    int16 vref_AD;
    vref_AD = (int16)( (double)v_AD/ADC_VAC_SC/VREF_10);
    return vref_AD;
} /* end scale_mod */

// Scale voltage reference to suit ISR use
int16 scale_vref_CD(Uint16 v_CD)
{
    int16 vref_CD;
    vref_CD = (int16)( (double)v_CD/ADC_VAC_SC/VREF_10);
    return vref_CD;
} /* end scale_mod */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

/**
 * This function initialises the VSI switching and hardware interrupts.
 * It:
 * \li Sets the output pins on the DSP to PWM mode
 */

void st_vsi_init(void)
{
    //...
void pwm_init(void)
{
    // Disable CPU interrupts
    DINT;
    // Establishes stable initial condition for event manager config registers
    // EVA
    EvaRegs.ACTRA.all = 0x0000; // force PWM1-PWM6 outputs to low level
    EvaRegs.EVAIMRA.all = 0x0000; // Disable interrupts
    EvaRegs.EVAIMRB.all = 0x0000;
    EvaRegs.EVAIMRC.all = 0x0000;
    EvaRegs.EVAIMRA.all = 0x0000;
    EvaRegs.EVAIFRA.all = BIT0;
    EvaRegs.COMCONA.all = 0x0000;
    EvaRegs.DBTCONA.all = 0x0000;
    EvaRegs.T1CON.all = 0x0000;
    EvaRegs.T2CON.all = 0x0000;
    EvaRegs.T1CNT = 0x0000;
    EvaRegs.T2CNT = 0x0000;
    EvaRegs.CAPCONA.all = 0x0000;
    EvaRegs.CAPFIFOA.all = 0x0000;
    // EVB
    EvbRegs.ACTRB.all = 0x0000; // force PWM7-PWM12 outputs to low level
    EvbRegs.EVBIMRA.all = 0x0000; // Disable interrupts
    EvbRegs.EVBIMRB.all = 0x0000;
    EvbRegs.EVBIMRC.all = 0x0000;
    EvbRegs.EVBIPRA.all = BIT0;
    EvbRegs.COMCONB.all = 0x0000;
    EvbRegs.DBTCONB.all = 0x0000;
    EvbRegs.T3CON.all = 0x0000;
    EvbRegs.T4CON.all = 0x0000;
    EvbRegs.T3CNT = 0x0000;
    EvbRegs.T4CNT = 0x0000;
    // Establishes stable initial condition for event manager config registers
    EALLOW; // Enables writing to protected registers

    // -> Setup PWM output pins
    /* EVA - PWM1=GCU, PWM2=GCL, PWM3=GDU, PWM4=GDG
       EVB - PWM7=GAU, PWM8=GAL, PWM9=GBU, PWM10=GBL */

    // EVA
    GpioMuxRegs.GPAMUX.bit.PWM1_GPIOA0 = 1; // enable PWM1 pin
    GpioMuxRegs.GPAMUX.bit.PWM2_GPIOA1 = 1; // enable PWM2 pin
    GpioMuxRegs.GPAMUX.bit.PWM3_GPIOA2 = 1; // enable PWM3 pin
    GpioMuxRegs.GPAMUX.bit.PWM4_GPIOA3 = 1; // enable PWM4 pin

    // EVB
    GpioMuxRegs.GPBMUX.bit.PWM7_GPIOB0 = 1; // enable PWM7 pin
    GpioMuxRegs.GPBMUX.bit.PWM8_GPIOB1 = 1; // enable PWM8 pin
    GpioMuxRegs.GPBMUX.bit.PWM9_GPIOB2 = 1; // enable PWM9 pin
    GpioMuxRegs.GPBMUX.bit.PWM10_GPIOB3 = 1; // enable PWM10 pin

    // Qualification period
    GpioMuxRegs.GPQUAL.bit.QUALPRD = 15; // 1000ns qualification period

 DSP Source Code: Experimental System

// Set up PWM outputs
EDIS; // Disables writing to protected registers

// Setup timers
// Setup deadband time - 1.2us
/* DBT DBTPS time
 9 2 0.24
 9 3 0.6
 9 4 1.2
12 3 0.8 */

// Configure Timer1 for sampling/switching on EVA
EvaRegs.T1CON.all = 0;
EvaRegs.T1CON.bit.FREE = 0; // timer stops emu suspend
EvaRegs.T1CON.bit.SOFT = 0;
EvaRegs.T1CON.bit.TMODE = 1; // continuous up/down count mode
EvaRegs.T1CON.bit.TECMPR = 1; // enable timer compare
EvaRegs.T1CON.bit.TPS = 0; // input clock prescaler
EvaRegs.T1CON.bit.TPS = 4; // input clock prescaler (for testing purposes)
EvaRegs.T1CON.bit.TCLD10 = 1; // Reload timer compare on period or 0
EvaRegs.T1PR = PERIOD;
EvaRegs.T1CNT = 0;
EvaRegs.T1CMPR = PERIOD;

// Configure Timer4 in sync with Timer3 on EVB
EvbRegs.T4CON.all = 0;
EvbRegs.T4CON.bit.FREE = 0; // timer stops emu suspend
EvbRegs.T4CON.bit.SOFT = 0;
EvbRegs.T4CON.bit.TMODE = 0; // disabled
EvbRegs.T4CON.bit.TECMPR = 0; // disable timer compare
EvbRegs.T4CON.bit.TPS = 0; // input clock prescaler
EvbRegs.T4CON.bit.TPS = 4; // input clock prescaler (for testing purposes)
EvbRegs.T4CON.bit.TCLD10 = 1; // Reload timer compare on period or 0
EvbRegs.T4PR = PERIOD;
EvbRegs.T4CNT = PERIOD;
EvbRegs.T4CMPR = PERIOD; // doesn't matter, because timer compare is disabled

// Start timers
EvaRegs.T1CON.bit.TENABLE = 1; // Enables Timer 1 and Timer 2 simultaneously
EvbRegs.T3CON.bit.TENABLE = 1; // Enables Timer 3 and Timer 4
simultaneously
// <- Setup timers

// -> Initialise CMPRx
// EVA
EvaRegs.CMPR1 = PERIOD_2;
EvaRegs.CMPR2 = PERIOD_2;
EvaRegs.CMPR3 = PERIOD_2;
// EVB
EvbRegs.CMPR4 = PERIOD_2;
EvbRegs.CMPR5 = PERIOD_2;
EvbRegs.CMPR6 = PERIOD_2;

// -> Initialise CMPRx
// <- Initialise CMPRx

// Setup timers
/* DBT DBTPS time
9 2 0.24
9 3 0.48
9 4 0.96
9 5 1.92
12 3 0.64
12 5 2.56
15 5 3.2
deadtime = 2^DBTPS * DBT / clock freq (150M) */

// Setup deadband
data = 1.2us
// EVA
EvaRegs.DBTCONA.bit.DBT = 9;
EvaRegs.DBTCONA.bit.DBTPS = 4;
EvaRegs.DBTCONA.bit.EDBT1 = 1;
EvaRegs.DBTCONA.bit.EDBT2 = 1;
EvaRegs.DBTCONA.bit.EDBT3 = 1;
// EVB
EvbRegs.DBTCONB.bit.DBT = 9;
EvbRegs.DBTCONB.bit.DBTPS = 4;
EvbRegs.DBTCONB.bit.EDBT1 = 1;
EvbRegs.DBTCONB.bit.EDBT2 = 1;
EvbRegs.DBTCONB.bit.EDBT3 = 1;

// Setup action control registers
// EVA
EvaRegs.ACTRA.all = 0x0000;
// EVB
EvbRegs.ACTRB.all = 0x0000;
// <- Setup action control registers

// -> Set compare control registers
// EVA
EvaRegs.COMCONA.all = 0;
EvaRegs.COMCONA.bit.ACTRLD = 2; // reload ACTRA immediately
EvaRegs.COMCONA.bit.SVENABLE = 0; // disable space vector PWM
EvaRegs.COMCONA.bit.CLD = 1; // reload CMPRA on underflow and period
EvaRegs.COMCONA.bit.FCOMPOE = 1; // full compare enable
EvaRegs.COMCONA.bit.CENABLE = 1; // enable compare operation

// EVB
EvbRegs.COMCONB.all = 0;
EvbRegs.COMCONB.bit.ACTRLD = 2; // reload ACTRB immediately
EvbRegs.COMCONB.bit.SVENABLE = 0; // disable space vector PWM
EvbRegs.COMCONB.bit.CLD = 1; // reload CMPRB on underflow and period
EvbRegs.COMCONB.bit.FCOMPOE = 1; // full compare enable
EvbRegs.COMCONB.bit.CENABLE = 1; // enable compare operation
// <- Set compare control registers

// -> Set how ADC is triggered
EvaRegs.GPTCONA.all = 0;
EvaRegs.GPTCONA.bit.T2TOADC = 2; // Start ADC on Timer 2
underflow
// <- Set how ADC is triggered

// -> Map interrupt vectors to ISR functions
EALLOW; // Enables writing to protected registers
Appendix B. DSP Source Code: Experimental System

PieVectTable.ADCINT = &isr_pwm; // ADC int triggers isr_pwm
PieVectTable.PDPINTA = &isr_pdpint; // PDPINTA
PieVectTable.PDPINTB = &isr_pdpint; // PDPINTB

EDIS; // Disables writing to protected registers
// <= Map interrupt vectors to ISR functions

// --> Configure interrupts
EvaRegs.EVAIMRA.bit.PDPINTA = 1; // enable interrupt on PDPINTA
EvaRegs.EVAIFRA.bit.PDPINTA = 1; // clear PDPINTA interrupt flag
EvbRegs.EVBIMRA.bit.PDPINTB = 1; // enable interrupt on PDPINTB
EvbRegs.EVBIFRA.bit.PDPINTB = 1; // clear PDPINTB interrupt flag

// Enable PDPINTA in PIE: Group 1 interrupt 1
PieCtrlRegs.PIEIER1.bit.INTx1 = 1;

// Enable PDPINTB in PIE: Group 1 interrupt 2
PieCtrlRegs.PIEIER1.bit.INTx2 = 1;

// Enable ADC interrupt in PIE: Group 1 interrupt 6
PieCtrlRegs.PIEIER1.bit.INTx6 = 1;

AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1; // clear interrupt flag
PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;

EVA |= M_INT1; // Enable CPU Interrupt 1

EINT; // enable all maskable interrupts
// <= Configure interrupts

// Initial compare values to 50%
EvaRegs.CMPR1 = period_2;
EvaRegs.CMPR2 = period_2;
EvbRegs.CMPR4 = period_2;
EvbRegs.CMPR5 = period_2;

// enable timer 1, 2 and 3
EvaRegs.T1CON.bit.TENABLE = 1;
EvbRegs.T3CON.bit.TENABLE = 1;

ENABLE_GATES();       ///
/* * * * * * * * * * * * * * * * * * * * * * * * * * * 
** * * * */

This function initialises the ADC unit to:

li Trigger a conversion sequence from timer 2 underflow
li Convert the appropriate ADC channels

Result registers as follows:

<table>
<thead>
<tr>
<th>Result register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADCRESULT0</td>
<td>ADCINA0 CPT-E13 IAC1 input</td>
</tr>
<tr>
<td>ADCRESULT1</td>
<td>ADCINB0 CPT-E13 VDC3 input</td>
</tr>
<tr>
<td>ADCRESULT2</td>
<td>ADCINB1 CPT-E13 IAC2 input</td>
</tr>
<tr>
<td>ADCRESULT3</td>
<td>ADCINB2 CPT-E13 VAC2 input</td>
</tr>
<tr>
<td>ADCRESULT4</td>
<td>ADCINA2 CPT-E13 IAC3 input</td>
</tr>
<tr>
<td>ADCRESULT5</td>
<td>ADCINB3 CPT-E13 VAC2 input</td>
</tr>
<tr>
<td>ADCRESULT6</td>
<td>ADCINA3 CPT-E13 IDC1 input</td>
</tr>
<tr>
<td>ADCRESULT7</td>
<td>ADCINB4 CPT-E13 IDC2 input</td>
</tr>
<tr>
<td>ADCRESULT8</td>
<td>ADCINB5 CPT-E13 VDC1 input</td>
</tr>
<tr>
<td>ADCRESULT9</td>
<td>ADCINA5 not used</td>
</tr>
<tr>
<td>ADCRESULT10</td>
<td>ADCINB6 yHA</td>
</tr>
<tr>
<td>ADCRESULT11</td>
<td>ADCINA6 yHA</td>
</tr>
<tr>
<td>ADCRESULT12</td>
<td>ADCINB7 yLB</td>
</tr>
<tr>
<td>ADCRESULT13</td>
<td>ADCINA7 yLB</td>
</tr>
<tr>
<td>ADCRESULT14</td>
<td>ADCINB8 yLB</td>
</tr>
</tbody>
</table>

void adc_init(void)
{
    AdcRegs.ADCMAXCONV.all = 0x0007; // Setup 8 conv's on SEQ1, SEQ2
    AdcRegs.ADCCHSELSQ1.bit.COMV00 = 0x0; // Setup ADCINA/B0 as 1st conv.
    AdcRegs.ADCCHSELSQ1.bit.COMV01 = 0x1; // Setup ADCINA/B1 as 2nd conv.
Appendix B. DSP Source Code: Experimental System

```c
AdcRegs.ADCCHSELSEQ1.bit.CONV02 = 0x2; // Setup ADCINA/B2 as 3rd conv.
AdcRegs.ADCCHSELSEQ1.bit.CONV03 = 0x3; // Setup ADCINA/B3 as 4th conv.
AdcRegs.ADCCHSELSEQ2.bit.CONV04 = 0x4; // Setup ADCINA/B4 as 5th conv.
AdcRegs.ADCCHSELSEQ2.bit.CONV05 = 0x5; // Setup ADCINA/B5 as 6th conv.
AdcRegs.ADCCHSELSEQ2.bit.CONV06 = 0x6; // Setup ADCINA/B6 as 7th conv.
AdcRegs.ADCCHSELSEQ2.bit.CONV07 = 0x7; // Setup ADCINA/B7 as 8th conv.
AdcRegs.ADCTRL1.bit.ACQ_PS = 1; // lengthen acq window size
AdcRegs.ADCTRL1.bit.SEQ_CASC = 1; // cascaded sequencer mode
AdcRegs.ADCTRL2.bit.EVA_SOC_SEQ1 = 1; // EV manager start
AdcRegs.ADCTRL2.bit.INT_ENA_SEQ1 = 1; // enable interrupt
AdcRegs.ADCTRL2.bit.INT_MOD_SEQ1 = 0; // int at end of every SEQ1
AdcRegs.ADCTRL3.bit.SMODE_SEL = 1; // simultaneous sampling mode
AdcRegs.ADCTRL3.bit.ADCCLKPS = 0x04; // ADCLK = HSPCLK/8 (9.375MHz)
} /* end adc_init */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/**
See spra989a.pdf for calibration details
*/

void adc_calibrate(void)
{
  // char
  // str[60];
  double
  yHA = 0.0,
  yLH,
  yLB;
  yHA = (double)adc.yHA.dc_sum_bak/(double)ADC_COUNT_CAL;
  yLB = (double)adc.yLB.dc_sum_bak/(double)ADC_COUNT_CAL;
  cal_gain_A = (xH - xL)/(yHA - yLA);
  cal_offset_A = yLA * cal_gain_A - xL;
  cal_gain_B = (xH - xL)/(yHB - yLB);
  cal_offset_B = yLB * cal_gain_B - xL;
  // sanity check on gains
  if (   ( (cal_gain_A > 0.95) && (cal_gain_A < 1.05) )
    && ( (cal_gain_B > 0.95) && (cal_gain_B < 1.05) )
    && ( (cal_offset_A > -80.0) && (cal_offset_A < 80.0) )
    && ( (cal_offset_B > -80.0) && (cal_offset_B < 80.0) ) )
  {
    cal_gainA = (int16)(cal_gain_A*(double)(1<<14));
    cal_gainB = (int16)(cal_gain_B*(double)(1<<14));
    cal_offsetA = (int16)cal_offset_A;
    cal_offsetB = (int16)cal_offset_B;
  }
  // sprintf(str,"cal:gA=%.3f,oA=%5.1f, gB=%.3f,oB=%5.1f\n",cal_gain_A,
  // cal_offsetA,cal_gain_B,cal_offset_B);
  // puts_COM0(str);
} /* end adc_calibrate */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/**
\fn void adc_scale(void)
\brief Scales the filtered ADC quantities.

This function is called every 20ms to perform the RMS calculations and scale
the analog quantities to Volts and Amps for use in the background.
It also calculates the Vac feed forward compensation factor to
improve the output voltage load regulation.
*/
```

150
Appendix B. DSP Source Code: Experimental System

```c
void adc_rms_scale(void)
{
    double val;
    // calculate Iac RMS quantity
    val = (double)adc.iac_a.dc_sum_bak/(double)adc.count_rms_bak;
    val = (double)adc.iac_a.rms_sum_bak*(double)(1<<ADC_RMS_PS)
    / (double)adc.count_rms_bak - val*val;
    if (val < 0.0) val = 0.0;
    adc.iac_a.real = ADC_IAC_SC * sqrt(val);

    val = (double)adc.iac_b.dc_sum_bak/(double)adc.count_rms_bak;
    val = (double)adc.iac_b.rms_sum_bak*(double)(1<<ADC_RMS_PS)
    / (double)adc.count_rms_bak - val*val;
    if (val < 0.0) val = 0.0;
    adc.iac_b.real = ADC_IAC_SC * sqrt(val);

    val = (double)adc.iac_c.dc_sum_bak/(double)adc.count_rms_bak;
    val = (double)adc.iac_c.rms_sum_bak*(double)(1<<ADC_RMS_PS)
    / (double)adc.count_rms_bak - val*val;
    if (val < 0.0) val = 0.0;
    adc.iac_c.real = ADC_IAC_SC * sqrt(val);

    // calculate Vac RMS quantity
    val = (double)adc.vac_an.dc_sum_bak/(double)adc.count_rms_bak;
    val = (double)adc.vac_an.rms_sum_bak*(double)(1<<ADC_RMS_PS)
    / (double)adc.count_rms_bak - val*val;
    if (val < 0.0) val = 0.0;
    adc.vac_an.real = ADC_VAC_SC * sqrt(val);

    val = (double)adc.vac_bn.dc_sum_bak/(double)adc.count_rms_bak;
    val = (double)adc.vac_bn.rms_sum_bak*(double)(1<<ADC_RMS_PS)
    / (double)adc.count_rms_bak - val*val;
    if (val < 0.0) val = 0.0;
    adc.vac_bn.real = ADC_VAC_SC * sqrt(val);
}

/* end adc_rms_scale */

/**
This function is called every 200ms to perform the RMS calculations on
the input AC voltage signals and average the DC signals. The analog
quantities are scaled to Volts and Amps for use in the background.
*/

void adc_scale_slow(void)
{
    // calculate filtered DC values
    adc.iac_a_dc.real = ADC_IAC_SC * (double)adc.iac_a_dc.dc_sum_bak
    / (double)COUNT_DC_IN;
    adc.iac_b_dc.real = ADC_IAC_SC * (double)adc.iac_b_dc.dc_sum_bak
    / (double)COUNT_DC_IN;
    adc.iac_c_dc.real = ADC_IAC_SC * (double)adc.iac_c_dc.dc_sum_bak
    / (double)COUNT_DC_IN;
    adc.vac_an_dc.real = ADC_VAC_SC * (double)adc.vac_an_dc.dc_sum_bak
    / (double)COUNT_DC_IN;
    adc.vac_bn_dc.real = ADC_VAC_SC * (double)adc.vac_bn_dc.dc_sum_bak
    / (double)COUNT_DC_IN;
    adc.hvdc.real = ADC_VDC_SC * (double)adc.hvdc.dc_sum_bak
    / (double)COUNT_DC_IN;
    adc.hvdc1.real = ADC_VDC2_SC * (double)adc.hvdc1.dc_sum_bak
    / (double)COUNT_DC_IN;
    adc.hvdc2.real = ADC_VDC3_SC * (double)adc.hvdc2.dc_sum_bak
    / (double)COUNT_DC_IN;
    adc.idc.real = ADC_IDC_SC * (double)adc.idc.dc_sum_bak
    / (double)COUNT_DC_IN;
}

/* end adc_scale_slow */

// droop control
```
/*
This function runs every 1 ms
*/

void droop_control(void)
{
    debug_var3++;

    // Master Code
    P_AD=(V_AD_al*Ia_al+V_AD_be*Ia_be)*0.50;
    Q_AD=(V_AD_be*Ia_al-V_AD_al*Ia_be)*0.50;
    S_AD_unlimited=P_AD*P_AD+Q_AD*Q_AD;
    if(S_AD_unlimited >= UPPER_LIMIT_AD)
        S_AD_limited = UPPER_LIMIT_AD;
    else if(S_AD_unlimited <= LOWER_LIMIT_AD)
        S_AD_limited = LOWER_LIMIT_AD;
    else
        S_AD_limited = S_AD_unlimited;

    S_AD_limited = sqrt(S_AD_limited);
    pf_AD=(P_AD/S_AD_limited);
    if(Q_AD < 0.0)
        pf_AD = -pf_AD;

    // Slave Code
    P_CD=(V_CD_al*Ic_al+V_CD_be*Ic_be)*0.50;
    Q_CD=(V_CD_be*Ic_al-V_CD_al*Ic_be)*0.50;
    S_CD_unlimited=P_CD*P_CD+Q_CD*Q_CD;
    if(S_CD_unlimited >= UPPER_LIMIT_CD)
        S_CD_limited = UPPER_LIMIT_CD;
    else if(S_CD_unlimited <= LOWER_LIMIT_CD)
        S_CD_limited = S_CD_unlimited;
    else
        S_CD_limited = S_CD_unlimited;

    S_CD_limited = sqrt(S_CD_limited);
    pf_CD=(P_CD/S_CD_limited);
    if(Q_CD < 0.0)
        pf_CD = -pf_CD;

    w_o_CD_ref = W_N + lf_int_sum;
    pf_CD_ref=((w_max-w_o_CD)/Mp);  //pf regulator implementation
    if(EnablePFReg == 1)
    {
        if(run_pf_reg == 1)
        {
            if(pf_reg_counter == 15)
            {
                pf_reg_disable = 0;
                pf_reg_counter = 0;
            }
            else
            {
                pf_reg_counter++;
                pf_reg_disable = 1;
            }
        }
    }
if((run_pf_reg == 1) && (pf_reg_disable == 0))
{
    if(toggle_bit2) toggle_bit2 = 0; else toggle_bit2 = 1;
    run_pf_reg = 0;

    Kp_pf_reg = (0.05); // (0.1);
    Ki_pf_reg = (500.0)*(1.0/50.0); // multiplied by fundamental period

    pf_CD_err = (pf_CD_ref - pf_CD);
    prop_pf_reg = (pf_CD_err*Kp_pf_reg);
    if(pf_reg_action != 0) // freeze integrator if pf_reg_action = 0
        int_pf_reg = int_pf_reg + (pf_CD_err*Ki_pf_reg);
    Theta_comp = (prop_pf_reg + int_pf_reg);
    phase_shift_b_double = ((Theta_comp)/360.0)*65536.0*65536.0;
    if((phase_shift_b_double - prev_phase_shift_b_double) < 0.0)
        phase_shift_b = (Uint32)(-phase_shift_b_double - prev_phase_shift_b_double);
    else
        phase_shift_b = (Uint32)(phase_shift_b_double - prev_phase_shift_b_double);
    prev_phase_shift_b_double = phase_shift_b_double;

    if(pf_CD_err < -0.01)
        pf_reg_action = 1; // use positive phase_shift
    else if(pf_CD_err > 0.01)
        pf_reg_action = -1; // use negative phase_shift
    else
        pf_reg_action = 0; // don't phase-shift

    // w_o_AD = W_N; // droop is disabled
    w_o_AD = (w_max - Mp*pf_AD);
    phase_step_a_new = (int32)(w_o_AD*(1.0/2.0/PI*65536.0*65536.0/VSI_FINT));
    // step routine
    if (TransientCounter != 0)
        TransientCounter = 0;
    if (TransientCounter == 1)
    {
        SET_TP10();
        v_ref_CD = 450;
        v_isr_CD = scale_vref_CD(v_ref_CD);
    }
    if (TransientCounter == 5000)
    {
        v_ref_CD = 650;
        v_isr_CD = scale_vref_CD(v_ref_CD);
    }
    if (TransientCounter == 10000)
    {
        v_ref_CD = 850;
        v_isr_CD = scale_vref_CD(v_ref_CD);
    }
    if (TransientCounter == 15000)
    {
Appendix B. DSP Source Code: Experimental System

v_ref_CD = 450;
v_isr_CD = scale_vref_CD(v_ref_CD);
TransientCounter = 0;
CLEAR_TP10();
}
else
TransientCounter++;}
}

// end vsi.c
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * */

C.5. Grab Routine – Header File

// Grab.h
*
=====================================================================
__Grab_Code_Definitions()
=====================================================================
/* * */

//ifdef GRAB_SETTINGS_DEFINED_ON_LOCALFOLDER

#undef GRAB_LOCAL_H

/** @name Grab Code Definitions */
//@{
///grab type
#define GRAB_SHORT
#define GRAB_LONG 1
#define GRAB_DOUBLE 1

// grab array size
#define GRAB_LENGTH 200 ///< Length of Grab Code Array
#define GRAB_WIDTH 6   ///< Width of Grab Code Array
//@}
#endif
#define GRAB_DEC 10 ///< Decimals - not used

// modes
#define GRAB_GO 0 ///< logging data and waiting for trigger to stop
#define GRAB_IDLE 1 ///< waiting for a start signal
#define GRAB_TRIG 2 ///< waiting for a trigger point
#define GRAB_STOPPED 3 ///< finished logging data
#define GRAB_SHOW 4 ///< showing logged data

// macros
/// starts waiting for a trigger
#define GrabStart() { grab_index = 0; grab_mode = GRAB_TRIG; }
/// goes from triggered to running
#define GrabRun() { grab_index = 0; grab_mode = GRAB_GO; }
/// forces a running grab to stop
#define GrabStop() grab_mode = GRAB_STOPPED;

/// Clear grab contents - ready to overwrite
#define GrabClear() { grab_index = 0; grab_mode = GRAB_IDLE; }

/// Check if Grab has been Triggered
#define GrabTriggered() (grab_mode == GRAB_TRIGGER)
/// Check if Grab is Running
#define GrabRunning() (grab_mode == GRAB_GO)
/// Check if Grab has stopped
#define GrabStopped() (grab_mode == GRAB_STOPPED)
/// Check if Grab data is available
#define GrabAvail() (grab_mode >= GRAB_STOPPED)
/// Check if Grab data can be triggered
#define GrabShowTrigger() (grab_mode == GRAB_SHOW)
Appendix B. DSP Source Code: Experimental System

```c
/// log data macro
#define GrabStore(_loc_,_data_)
    grab_array[grab_index][_loc_] = _data_

/// wrap around storage
#define GrabStep() { grab_index++;
    if (grab_index >= GRAB_LENGTH) 
        grab_mode = GRAB_STOPPED; 
    }

/*
==================================================================
====
====
__Exported_Variables()
==================================================================
======= */
extern int16
    grab_mode, ///< Current Grab Mode
    grab_index, ///< Index into Grab array
    grab_dec; ///< grab decimation counter
#ifdef GRAB_SHORT
extern short
    grab_array[GRAB_LENGTH][GRAB_WIDTH]; ///< Grab Array
#endif
#ifdef GRAB_LONG
extern long
    grab_array[GRAB_LENGTH][GRAB_WIDTH]; ///< Grab Array
#endif
#ifdef GRAB_DOUBLE
extern double
    grab_array[GRAB_LENGTH][GRAB_WIDTH]; ///< Grab Array
#endif
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * */

--- */

// functions
void GrabDisplay(void); ///< Display the grabbed data for file storage
void GrabInit(void); ///< Initialise the grab data storage array
/* */
/* end grab.h
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

--- */

// functions
void GrabDisplay(void); ///< Display the grabbed data for file storage
void GrabInit(void); ///< Initialise the grab data storage array
/* */
/* end grab.h
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
References


References


References


