HYSTERESIS CURRENT REGULATION OF VOLTAGE SOURCE INVERTERS WITH CONSTANT SWITCHING FREQUENCY

A thesis submitted in accordance with the regulations to RMIT University in fulfilment of the requirements for the degree of
Doctor of Philosophy

by

Reza Davoodnezhad
B.E. (Hons) Monash University 2008

School of Electrical and Computer Engineering

RMIT University Australia

January 2014

Copyright Notice

Table of Contents

List of Figures

List of Tables

ABSTRACT

This thesis presents an integrated approach for constant frequency closed-loop hysteresis current control of a VSI. The proposed approach uses the fact that the switching frequency of a fixed-band hysteresis-controlled inverter varies according to the average load voltage. A novel technique is used to measure this average voltage by capturing transition times of the phase leg switching events, to avoid the frequency roll-off effects of using a low pass filter. The average voltage is then used to vary the hysteresis band in order to keep the VSI switching frequency constant. Next, a refinement is applied to the variable band to synchronise the zero-crossing of the current error to a fixed reference clock. The zero-crossing time is calculated by linearly interpolating between the captured switching transition times, to avoid the need for direct measurement of the zero-crossing event. This achieves a more robust and accurate synchronization process compared to current state-of-the-art time-based and deadbeat hysteresis controllers. For a three-phase VSI, the common mode current is subtracted from the phase leg current error that is calculated from three-phase leg gate signals. Further enhancements are then presented to extend the linear modulation range by 15%, to maintain excellent switching stability during excursions into overmodulation and to replace the third phase regulator with a fixed frequency directly modulated phase leg. The overall result is a two-level hysteresis current control approach with a harmonic performance that is similar to open-loop CSVM.

The average voltage calculation of the phase leg switched voltage then allows the integrated control concept to be applied to three-level three-phase multilevel topologies. Firstly, it allows the development of a three-level variable hysteresis band with band refinement similar to a two-level inverter. Secondly, it facilitates the development of a logic decoder based on the identification of the average voltage zero-crossings, so that only one hysteresis comparator is required per phase leg. A finite state machine is then developed to utilize the redundant switching states of the VSI similar to open-loop phase-shifted PWM. The control concept is then extended to suit a three-phase system by compensating for the common mode interacting current in a similar way as was done for two-level inverters. Further developments are then presented to synchronize the three-phase current errors to a fixed reference clock. The overall result is a three-level hysteresis current control approach that achieves a performance similar to open-loop phase disposition (PD) pulse width modulated three-phase multilevel inverter.

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.

Reza Davoodnezhad

Acknowledgements

I wish to express my sincere appreciation to those who have contributed to this thesis and supported me in one way or the other during this amazing journey.

First of all, I am extremely grateful to my supervisors, Professor Grahame Holmes and Dr. Brendan McGrath, for their guidance and all the useful discussions and brainstorming sessions, especially during the difficult first year (development stage) and the last six months (writing stage). Their deep insights helped me at various stages of my research.

Very special thanks to the Power Electronic Research Group Dr. Wang Kong, Dr. Dinesh Segaran , Mr Stewart Parker, Mr Carlos Teixeira and in particular and Mr Zaki Mohzani for their feedback and support.

Words cannot express the feelings I have for my parents Alireza and Mahnaz and parents in laws Kazem and Shahla for their constant unconditional support - both emotionally and financially.

Finally, I would like to acknowledge the most important person in my life – my wife Sahar. She has been a constant source of strength and inspiration. There were times during the past four years when everything seemed hopeless. I can honestly say that it was only her determination and constant encouragement (and sometimes a kick on my backside when I needed one) that ultimately made it possible for me to see this project through to the end.

Glossary of Terms

AC	Alternating Current
ADC	Analog Digital Converter
APOD	Alternative Phase Opposition Disposition
BJT	Bipolar Junction Transistor
CHB	Cascaded H-Bridge
CHCC	Conventional Hysteresis Current Controller
CPLD	Complex Programmable Logic Device
CPT-DA2810	Creative Power Technology DSP Card
CPT-GIIB	Creative Power Technology Inverter Card
CPT-IIB	Creative Power Technology Inverter Card
CPT-MINI2810	Creative Power Technology Controller Card
CSI	Current Source Inverter
CSVM	Centered Space Vector Modulation
DAC	Digital to Analog Converter
DC	Direct Current
DLL	Dynamic Link Library
dq	Direct Quadrature
DSP	Digital Signal Processor
EMF	Electromotive Force
EPLD	Erasable Programmable Logic Device
FC	Flying Capacitor
FPGA	Field-Programmable Gate Array
FSM	Finite State Machine
GTO	Gate Turn Off Thyristor
HB	Hysteresis Band
HCC	Hysteresis Current Controller
HIGH	Logic State ONE
HMI	Human Machine Interface
IGBT	Insulated Gate Bipolar Transistor
ISR	Interrupt Service Routine
LEM	Current Transducer
LOW	Logic State ZERO
LSC	Level Shifted Carrier
LS-PWM	Level Shifted Pulse Width Modulation

MBHCC	Multiple Band Hysteresis Current Controller
MOHCC	Multiple Offset Hysteresis Current Controller
MUX	Multiplexer
NPC	Neutral Point Clamped
OFF	Conducting State
ON	Non-Conducting State
PD	Phase Disposition
PI	Proportional Integrator
PLL	Phase Locked Loop
POD	Phase opposition Disposition
PR	Proportional Resonant
PSC	Phase Shifted Carrier
PSIM	Powersim Inc
PWM	Pulse Width Modulation
RL	resistive Inductive
RS232	Serial Interface
SPI	Serial Peripheral Interface
STATCOM	Static Compensator
SVHCC	Space Vector Hysteresis Current Controller
SVM	Space Vector Modulation
THD	Total Harmonic Distortion
UPS	Uninterruptable Power Supply
VSI	Voltage Source Inverter
WTHD	Weighted Total Harmonic Distortion
ZACE	Zero Average Current Error
APOD	Alternative Phase Opposition Disposition
AC	Alternating Current
ADC	Analog to Digital Converter
CAP	DSP Capture Port
CPLD	Complex Programmable Logic Device

List of Symbols

$\delta t_x \ , \ x \in A,B,C$	Three-phase time differences between the actual and expected current error zero-crossings
$\delta I_{x,h} \ , \ x \in A,B,C$	Three-phase calculated band offsets
$d_x \ , \ d_y \ , \ d_z$	Duty cycles of the three nearest space vectors
$e_\alpha(t), e_\beta(t)$	Stationary frame current error components
$e_d(t), e_q(t)$	Synchronous frame current error components
$E_x(t) \ , \ x \in A,B,C$	Three-phase back-emf
$e_x(t) \ , \ x \in A,B,C$	Three-phase current errors
f_o	Fundamental frequency (Hz)
f_{sw}	Switching frequency (Hz)
$f_{sw,+}, f_{sw,-}$	Switching frequency during the positive and negative average load voltage (Hz)
$\gamma'(t)$	Modified common mode interacting current for two-level inverter (including third harmonic current)
$\gamma(t)$	Common mode interacting current for two-level inverter
$\gamma'_{3L}(t)$	Modified common mode interacting current for three-level inverter (including neutral point current)
$\gamma_{3L}(t)$	Common mode interacting current for three-level inverter
$I_{3rd}(t)$	Third harmonic current component
$I_x^*(t) \ , \ x \in A,B,C$	Three-phase reference currents
$I_{h,x}(t) \ , \ x \in A,B,C$	Three-phase hysteresis bands
$I_{h,+}(t) \ , \ I_{h,-}(t)$	Hysteresis bands during positive and negative average voltage
I_α, I_β	Stationary frame components of the output current
I_d, I_q	Synchronous frame components of the output current
$I_f(t)$	Fundamental component of the output current

$I_r(t)$	Ripple component of the output current
$I_{NP}(t)$	Common mode neutral point current
$I_x^s(t)$, $x \in A,B,C$	Non-interacting component of the output current for two-level inverter
$I_{x,f}^s(t)$, $x \in A,B,C$	Fundamental component of the non-interacting output current for two-level inverter
$I_{x,r}^s(t)$, $x \in A,B,C$	Ripple component of the non-interacting output current for two-level inverter
$I_{x,3L}^s(t)$, $x \in A,B,C$	Non-interacting component of the output current for three-level inverter
$I_{x,3L,f}^s(t)$, $x \in A,B,C$	Fundamental component of the non-interacting output current for three-level inverter
$I_{x,3L,r}^s(t)$, $x \in A,B,C$	Ripple component of the non-interacting output current for three-level inverter
$I_{x,h,old}$, $x \in A,B,C$	Three-phase hysteresis bands before compensation of the band offsets
$I_{x,h,new}$, $x \in A,B,C$	Three-phase hysteresis bands after compensation of the band offsets
L_x , $x \in A,B,C$	Three-phase output inductors
M	Modulation command
R_x , $x \in A,B,C$	Three-phase output resistor
$S\overline{V}_x$, $S\overline{V}_y$, $S\overline{V}_z$	Three nearest space vectors
$S_x(t), \overline{S}_x(t)$ $x \in A,B,C$	Three phase switching commands and their complementary
$S_{PD,1x}(t), \overline{S}_{PD,1x}(t), S_{PD,2x}(t), \overline{S}_{PD,2x}(t)$ $x \in A,B,C$	Three-phase PD equivalent switching signals and their complementary
$S_{PS,1x}(t), \overline{S}_{PS,1x}(t), S_{PS,2x}(t), \overline{S}_{PS,2x}(t)$ $x \in A,B,C$	Three-phase PS equivalent switching signals and their complementary
$S_{x,OR}(t)$, $x \in A,B,C$	Three-phase modified hysteresis switching signals
$S_{x,OR,avg}(t)$, $x \in A,B,C$	Fundamental components of the three-phase feedback OR gate switching signals
$S_{x,tot}(t)$, $x \in A,B,C$	Three-phase hysteresis switching signals
$S_{x,tot,avg}(t)$, $x \in A,B,C$	Fundamental components of the three-phase hysteresis switching signals
θ	Angle of the applied voltage vector
t_{1+} , t_{1-}	ON time of the gate switching signal during positive and negative average voltage for three-level inverter
T_+ , T_-	Period of the gate switching signal during positive and negative average voltage for three-level inverter

T_1, T_2, T_3	Switching time events of the gate signal
t_{db}	Amount of the dead-band
t_{on}, t_{off}	ON time and OFF time of the gate switching signal for two-level inverter
$T_{OFF\ ,pred}$	Predicted OFF time of the next switching event
$T_{OFF\ ,prev}$	OFF time of the previous switching event
T_{prev}	Period of the previous switching event
$U_0(t)$	Three-phase load neutral point voltage
V^*	Applied voltage vector
$+V_{DC}, -V_{DC}$	Positive and negative DC bus voltage
$V_{x,avg}(t)\ ,\ x \in A,B,C$	Three-phase measured average voltages of the inverter
$V_x(t)\ ,\ x \in A,B,C$	Three-phase switched phase voltages of the inverter
V_{offset}	Injected common mode third harmonic voltage
$V_{NP}(t)$	Neutral point voltage of the NPC inverter
$V_L(t)$	Inductor voltage
Z_1, Z_2, Z_3	Switching time events of the current error zero-crossings

Chapter 1

Introduction

1.1 Background

Power electronic converters are an essential technology to transform electrical energy from available levels of voltage, current and operating frequency to a suitable form for various load applications, using power semiconductor devices such as IGBTs, Power MOSFETs, BJTs and GTOs [5]. Voltage source inverters (VSI) are a particular form of power electronic converter that convert direct current (DC) to alternating current (AC) at any required voltage level and operating frequency. A significant emerging application of VSIs is distributed grid-connected solar generation systems where the DC energy produced by the photovoltaic system is converted to AC form for household and industrial use. Depending on the application and the required output power, VSIs can be categorized as a single-phase leg (consists of one phase leg), single-phase (consists of two phase legs) and three-phase (consists of three phase legs). Furthermore, using series/parallel combinations of the power devices per phase leg, the output DC voltage levels can also be increased (from two-level to multilevel) to achieve higher power delivery and lower spectral distortion in the output switched waveform [7].

The fundamental requirement of converters used for DC/AC energy transformation is to switch the operating status of the power devices between one of two states - fully ON or fully OFF. The process of controlling the operating status of the individual devices is called *modulation*, with the most prevalent strategy being pulse width modulation (PWM) [11]. This strategy controls the amount of power delivered from the input supply to the output load by varying the duty cycle (or mark-space ratio) of the power devices at a high switching frequency, to achieve a target average low frequency output current/voltage. Broadly speaking, the approaches that have been investigated over the last several decades to address this need fall into one of two categories – open-loop modulation and closed-loop control.

Open-loop modulation of a VSI can be generally categorized as carrier-based and carrier-less modulation strategies [11][12]. Carrier-based modulation is implemented by comparing a low frequency fundamental reference command against a high frequency triangular carrier waveform to generate the required switching pulses for each power device. For a multilevel VSI, this strategy is extended either by phase shifting (PS-PWM) or level shifting (LS-PWM) the triangular carriers to generate the gate signals for each individual power device [24]. In contrast, carrier-less modulation strategies such as space vector (SV) modulation directly calculate the duty cycle from a space-vector representation of the reference command and the inverter switching states, with the placement of the switched pulses providing an additional degree of freedom. Previous work [15] has shown that carrier-based and SV modulation of VSIs generate identical switching pulses and hence produce identical space vector sequences, despite their divergent implementation strategies.

Closed-loop control of a VSI uses an internal current feedback loop and an open-loop modulator to regulate the instantaneous output current of the inverter with high accuracy. Most of the closed loop inverter current control strategies [3] that have been investigated over the last several decades fall into one of two categories – linear and non-linear.

Linear systems [3][4][14][53] such as proportional-integral (PI) or proportional-resonant (PR), separate the current error compensation and open-loop modulation processes. They measure an average current error at each calculation cycle, and then calculate an average inverter output voltage that corrects this error. This voltage command is passed to an open-loop pulse width modulator, which determines the inverter switching sequences that achieve this commanded volt-second output over the next modulation period. Linear regulation systems offer the particular benefits of a constant output switching frequency, full DC bus utilization and they only need two current regulators for a three-phase system. However they often have a limited high frequency dynamic response, must be matched to their load characteristics to achieve the best possible performance and do not provide instantaneous over-current protection.

Non-linear systems [3][51][67][78][84], such as hysteresis current control (HCC), zero average current error control (ZACE), space vector based hysteresis current control (SVHCC) and predictive deadbeat control, combine the two tasks of current control and modulation of the power devices. In the simplest form of hysteresis current control, the switching state of the VSI is defined by instantaneous comparisons between the required

reference currents and the measured currents, using a fixed set of hysteresis bands. These systems have the advantages of a fast "deadbeat" transient response, simplicity of implementation, insensitivity to load parameter variations, and inherent direct overcurrent protection. However, they suffer from a variable inverter switching frequency and (for three-phase systems) adverse interactions between the three-phase leg controllers [14][51][66][67] that can degrade the current regulation performance. Various strategies have been proposed in the literature to overcome the limitations of hysteresis current regulators for two-level inverters. However, they typically suffer from complexity of implementation and both their regulation and harmonic performance is not comparable to that achieved by a linear closed-loop system. For multilevel voltage source inverters, various hysteresis strategies [96][100][102][109][110][115] such as multiple band (MB), multiple offset (MO), SV based and time-based HCC have been previously reported. These strategies also suffer from variable switching frequency, DC steady state tracking error and limited dynamic response.

Hence, further research is needed into the development of an integrated approach for constant frequency hysteresis current regulation of voltage source inverters to solve these shortcomings.

1.2 Aim of the Research

The aim of the research presented in this thesis is to develop an integrated approach for constant frequency closed-loop hysteresis current control of VSIs. The work is presented in two stages.

Stage one develops a new approach for the hysteresis current regulation of a two-level VSI, with the particular benefits of:

1. Maintaining a constant switching frequency independently of the derivative of the current error.
2. Ensuring that a three-phase VSI selects the "three nearest space vectors" within one switching cycle, to achieve a harmonic performance that is as close as possible to centered space vector open-loop modulation.
3. Compensating for interactions between phase legs for a three-phase system, without requiring direct measurement of the three phase leg switched voltages.
4. Full utilization of the available DC bus voltage.

5. Maintaining current control and switching stability during excursions into the overmodulation region.
6. Avoiding current unbalance because of gain mismatch between the current transducers.

Stage two extends the new two-level hysteresis approach to three-level topologies such as a full bridge and the neutral point clamped (NPC) and flying capacitor (FC) multilevel inverters, with the particular benefits of:

1. Reliably detecting the instant of output voltage level change without requiring multiple hysteresis bands.
2. Maintaining a constant switching frequency independently of the current error zero-crossing information.
3. Achieving a line to line voltage harmonic performance that is similar to open-loop phase disposition PWM for a three-phase multilevel system.
4. Compensating for the interactions between phase legs for a three-phase multilevel system.
5. Maintaining a balanced intermediate neutral point voltage for a three-phase NPC inverter.
6. Maintaining balanced flying capacitor voltages for a three-phase FC inverter.

All the theoretical work presented has been verified using both simulation and experimental investigations under a wide variety of operating conditions.

1.3 Structure of the Thesis

The thesis structure is divided into four main sections. The first section presents a review of the existing literature for two-level and three-level VSIs, with their open-loop modulation and closed-loop control techniques. The second section presents the development of the new integrated hysteresis current control approach for two-level VSIs. The third section applies this development to three-level VSIs. Throughout each chapter, selected simulation and experimental results are provided to support the investigation of the research work. Each chapter is then concluded by presenting extensive consolidated simulation and experimental results. The last section of the thesis describes the simulation and experimental systems used to validate this research work.

The thesis chapters are arranged as follows:

Chapter 1 (this chapter) provides a context and overview for the work performed. It also identifies and outlines the thesis structure.

Chapter 2 provides an overview of the current literature for two-level VSIs and their existing linear open-loop and closed-loop current control and modulation strategies. A review of two-level hysteresis current control is provided for two-level single-phase and three-phase voltage source inverters. The literature review is then extended to the three-level multilevel topologies of NPC and FC inverters, together with multilevel hysteresis strategies for these types of inverters.

Chapter 3 begins with a detailed analysis to identify the major driver of variable switching frequency for two-level VSIs. Next, a new technique is proposed to measure and compensate for this influence, allowing an improved approach to be developed to maintain a constant frequency HCC. Additional refinements are then presented to further improve the switching frequency regulation of the converter and its associated harmonic performance.

Chapter 4 applies the new two-level single-phase hysteresis approach to three-phase VSIs. A strategy is presented to identify and compensate for the common mode interacting current. It is then shown how to implement the proposed hysteresis approach with only two hysteresis regulators and how to operate into the overmodulation region to achieve full utilization of the available DC bus voltage.

Chapter 5 presents the application of the new constant frequency hysteresis approach to multilevel VSIs. The approach presented in chapter 3 is extended to three-level VSI topologies with particular focus on a Full Bridge and the Neutral Point Clamped and the Flying Capacitor multilevel inverters. Further refinements are made to the variable hysteresis band calculation to improve the switching frequency regulation. A novel logic decoder is developed to allow the use of only a single hysteresis comparator per phase leg. The result is a switching process that is similar to the open-loop PWM strategies commonly used to modulate these inverters.

Chapter 6 extends the concepts of chapter 5 to three-phase multilevel inverters. A novel strategy is again developed to calculate and compensate for the effect of common mode current. Next, active balancing strategies are developed to maintain a balanced neutral point voltage for the NPC VSI and a balanced flying capacitor voltage for the FC

VSI. Finally, it is shown how to operate in into the overmodulation region while maintaining regulator stability.

Chapter 7 provides details of the simulation and experimental systems used for developing and implementing the new constant frequency hysteresis current control approach for both two-level and multilevel systems.

Chapter 8 concludes the thesis and offers recommendations for future work in this field of research.

1.4 Identification of Original Contribution

Hysteresis current regulation of VSIs has been a major research area in power electronics for many years. The work in this thesis develops a novel alternative approach for constant frequency hysteresis current regulation of VSIs. For clarity, it is useful to identify the original contributions of the thesis relating to this field of research.

The first contribution is a novel strategy to synchronously measure the average inverter output voltage of a hysteresis current regulated inverter by capturing the transition times of the phase leg switching events without requiring low pass filtering.

The second contribution is the development of a new approach for two-level constant frequency closed-loop hysteresis current control of a two-level single phase VSI, using the synchronously measured average inverter output voltage. A novel technique is then used to synchronise the current error zero-crossings to a fixed reference clock without requiring direct measurement of the zero-crossing times. The resulting strategy achieves a significantly more constant switching frequency than present state-of-the-art time-based and deadbeat hysteresis strategies.

The third contribution is to apply the new constant frequency hysteresis approach to a two-level three-phase VSI, with the objective of achieving a harmonic performance that is similar to open-loop CSVM. A new strategy is developed to compensate for common mode interacting current without requiring direct measurement of the three-switched phase voltages. The outcome fully utilizes the available DC bus voltage, achieves excellent current control in the overmodulation region and allows replacement of the third phase leg regulator with a fixed-frequency directly pulse width modulated controller.

The fourth contribution is the extension of the two-level hysteresis approach to three-level VSIs such as a full bridge, and NPC and FC inverters, requiring only one hysteresis comparator per phase leg while avoiding the creation of a DC steady state

tracking error. Additional strategies using combinational logic and finite state machine are included in this new algorithm to maintain the internal DC bus voltage balance for these inverters as part of the control process.

The fifth contribution extends the new three-level HCC strategy to three-phase NPC and FC inverters with the objective of achieving a harmonic performance that is similar to the harmonically superior open-loop phase disposition (PD) PWM. A new strategy is also presented to compensate for the common mode interacting current.

All of the theoretical work presented in this thesis has been confirmed in simulation and experiments.

1.5 Publications

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

Conferences

- Holmes, D.G.; Davoodnezhad, R.; McGrath, B.P.; "An improved three phase variable band hysteresis current regulator," *Power Electronics and ECCE Asia (ICPE & ECCE), 2011 IEEE 8th International Conference on*, vol., no., pp.2274-2281, May 30 2011-June 3 2011
- Davoodnezhad, R.; Holmes, D. G.; McGrath, B. P.;" Constant frequency three-phase hysteresis current regulator with extended modulation depth," *Power Electronics and Motion Control Conference (IPEMC), 2012 7th International*, vol.1, no., pp.263-268, 2-5 June 2012
- Davoodnezhad, R.; Holmes, D. G.; McGrath, B. P.; "Hysteresis current regulation of three phase flying capacitor inverter with balanced capacitor voltages," *Power Electronics and Motion Control Conference (IPEMC), 2012 7th International*, vol.1, no., pp.47-52, 2-5 June 2012
- Davoodnezhad, R.; Holmes, D. G.; McGrath, B. P.; "Three-Level Hysteresis Current Regulation for a Three Phase Neutral Point Clamped Inverter" *Power Electronics and Motion Control Conference (ECCE-EPE PEMC2012), 2012 7th International*, vol.1, no., pp.47-52, 4-6 Sep. 2012

- Davoodnezhad, R.; Holmes, D. G.; McGrath, B. P.; " A Three-Level Self-Synchronizing Hysteresis Current Regulator with Constant Switching Frequency " *Energy Conversion Congress and Conference (ECCEAsia DownUnder 2013), 2013 5th International*, vol.1, no., pp.38-44, 4-6 June. 2013

Journals

- Holmes, D.G.; Davoodnezhad, R.; McGrath, B.P.; "An Improved Three-Phase Variable-Band Hysteresis Current Regulator," *Power Electronics, IEEE Transactions on*, vol.28, no.1, pp.441-450, Jan. 2013.
- Davoodnezhad, R.; Holmes, D. G.; McGrath, B. P., "A Novel Three-Level Hysteresis Current Regulation of Three-Phase Multilevel Inverters. (accepted for publication, IEEE Transactions on Power Electronics)

Chapter 2

Literature Review

The background material presented in this chapter is divided into three main sections. The first section starts with a topological overview of major power electronic inverters and their open-loop modulation techniques. These topologies include two-level VSIs and three-level Neutral Point Clamped and Flying Capacitor VSIs. This review of current-state-of-the-art two-level and multilevel open-loop modulation is important since it identifies the goal of a current control strategy to achieve the same switching and harmonic performance.

The second section presents a background overview of the existing closed-loop linear current regulation strategies for these types of topologies. This acts as a baseline for comparison against the performance of the proposed non-linear controlled system in terms of the quality of the current control.

The third section is divided into two subsections with a focus on (1) hysteresis current regulation of single-phase and three-phase two-level VSIs, and (2) multilevel hysteresis current regulation of single-phase and three-phase multilevel VSIs. From this review, it can be seen how the performance of existing hysteresis strategies is constrained because of the variable switching frequency, dependence on the current error derivative and direct measurement of the current error zero-crossing, multiple hysteresis band arrangements and improper compensation of the common mode interacting current.

2.1 Two-Level Voltage Source Inverters and Their Open-Loop Modulation

2.1.1 Topological Overview of Two-level Inverters

Figure 2.1 shows the structure of three conventional two-level voltage source inverters. Figure 2.1(a) shows the structure of a single-phase leg voltage source inverter (also known as half-bridge VSI) [11] consisting of a DC bus voltage of $+2V_{DC}$, two power switches $(S_a, \bar{S}_a)$ and two anti-parallel diodes. The single-phase leg inverter generates output voltage values of $+V_{DC}$ and $-V_{DC}$ with respect to the input source neutral point (point Z). Figure 2.1(b) shows the structure of a single-phase voltage source inverter (also known as a H-bridge or full-bridge) constructed by combining two half-bridge VSIs connected across a common input DC bus

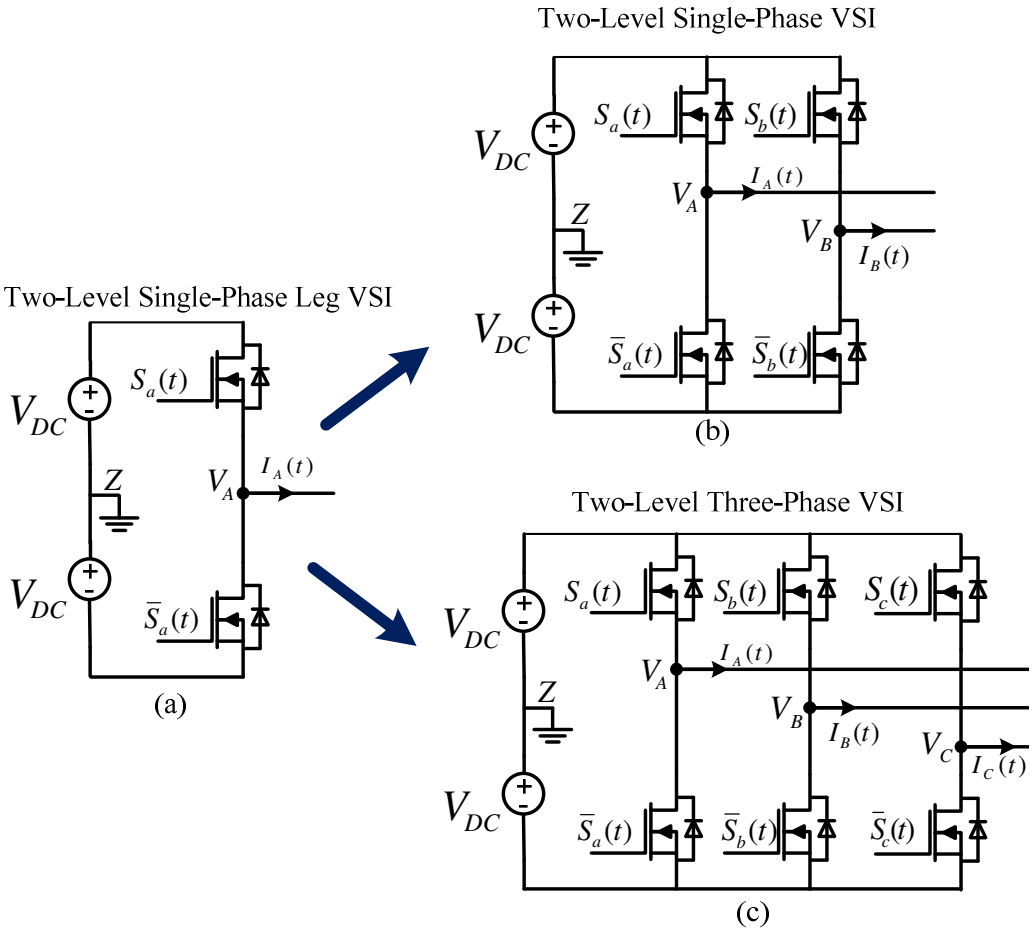


Figure 2.1: Structure of a two-level (a) single phase leg (b) single phase (H-bridge) (c) three-phase voltage source inverter

voltage. In this configuration, the maximum allowable output voltage is twice the output voltage of the half-bridge VSI since the entire DC voltage can be impressed across the output load, rather than only one half as is the situation for the single-phase leg inverter. Figure 2.1 (c) extends this idea to a three-phase voltage source inverter where three half-bridge VSIs are connected in parallel across a common DC bus input voltage.

Note that the topologies shown in Figure 2.1 are constructed by a progressive parallel combination of a half-bridge VSI that is capable of transferring power bidirectionally. If the power is transferred from the DC input side to the output AC side, the converter is considered an inverter while if the power is transferred from the AC side to the DC side the converter is considered a rectifier.

2.1.2 Open-Loop Modulation of Two-Level Inverters

Pulse width modulation of voltage source inverters has been a topic of intensive research over the last several decades as the primary strategy to control the AC output (either voltage or current) of a power electronic converter [11][12]. The fundamental concept of PWM is to control the duty cycle of each switch (mark-space ratio or ON time) at a high switching frequency to generate a train of switched pulses that have the required low frequency fundamental component. The second objective of these strategies [11] is to arrange the switching pulses in such a way as to minimize unwanted harmonic distortion, switching losses or to manage secondary performance criteria (e.g. common mode voltage).

The most common modulation strategy for a two-level voltage source inverter is naturally sampled sine-triangle pulse width modulation, shown in Figure 2.2. In this strategy, a low frequency sinusoidal reference signal (the modulation command) is compared against a high frequency triangular carrier. When the reference signal is greater than the carrier signal, the VSI phase leg upper switch turns ON, the lower switch is turned OFF, and the inverter phase leg output switches to the upper DC bus voltage. When the reference signal is smaller than the carrier signal, the VSI phase leg lower switch turns ON, the upper switch turns OFF, and the inverter phase leg output switches to the lower DC bus voltage. For a voltage source inverter, the switches of each phase leg cannot be turned ON simultaneously, to prevent a short circuit across the input voltage source and consequential shoot through currents through the power switches. For this reason, the phase leg switches are always

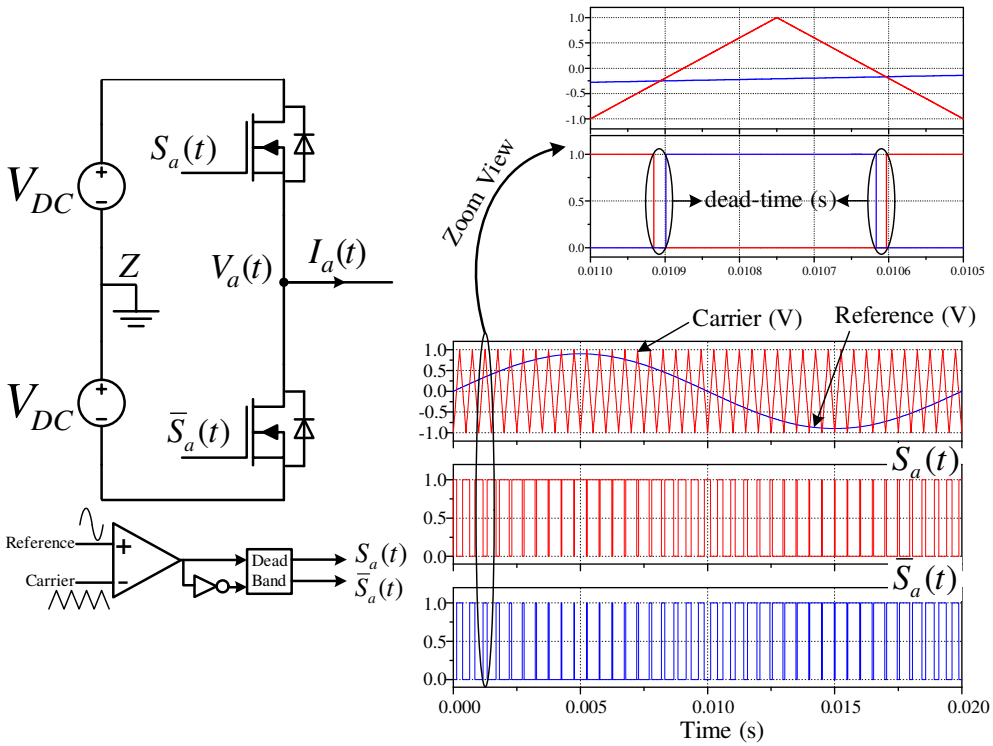


Figure 2.2: Open-loop modulation of two-level single phase leg voltage source inverter with dead-time

operated complementary ($S_a(t), \overline{S}_a(t)$ as shown in Figure 2.2) and a short null period is introduced between the turn OFF and turn ON events of the phase leg switches. This short period (usually a few percent of the switching period) is called dead-time and is essential for any VSI modulation strategy.

Once the modulation of a two-level half-bridge VSI is established, the concept can be easily extended to the full-bridge single-phase and three-phase VSIs. Figure 2.3 shows open-loop naturally sampled sine-triangle PWM for a single-phase VSI and the switching arrangement to achieve a two-level output voltage. In [11], it has been shown that the optimum harmonic performance of a single-phase H-bridge is achieved using a three-level double-edge asymmetrically sampled PWM. In this way, each phase leg uses a common triangle carrier where their sinusoidal reference commands are displaced by 180°. This ensures the cancellation of the main carrier harmonics, which substantially reduces the WTHD.

Figure 2.4 shows open-loop sine-triangle PWM of a two-level three-phase VSI with a separate modulation command (sinusoidal reference signal) per phase, to form a complete set of three-phase modulation commands that are now displaced by 120°. Figure 2.4 also shows the switched phase *a* voltage and the three-level switched *ab* line to line voltage.

Another common form of PWM reported for a three-phase VSI is space-vector modulation (SVM) [13][14]. The primary advantage of SVM over sinusoidal PWM is the explicit identification of the pulse placement within each carrier period as an extra degree of freedom, that has been shown to improve harmonic performance of the inverter [11][13]. The principle of SVM is based on the fact that for a two-level three-phase VSI, there are only eight possible switching states. Two of the switching states are null states ($\overline{SV}_0$ and $\overline{SV}_7$) and create a short circuit (zero voltage) across the output load. The other six switching states are the active states where the output load is connected to the DC link. The output switching states and these voltage vectors are shown in Figure 2.5. This figure also shows an arbitrary reference target voltage $\overline{V}^*$.

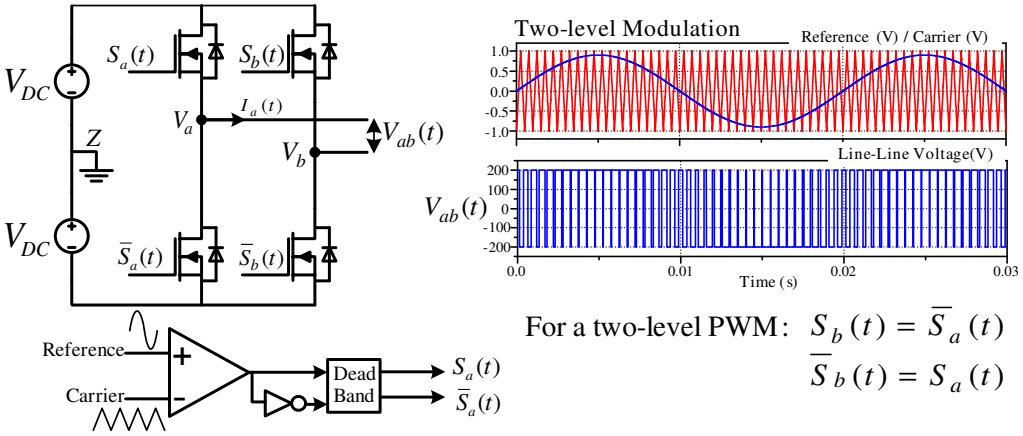


Figure 2.3: Open-loop modulation of full-bridge VSI using two-level sine-triangle PWM

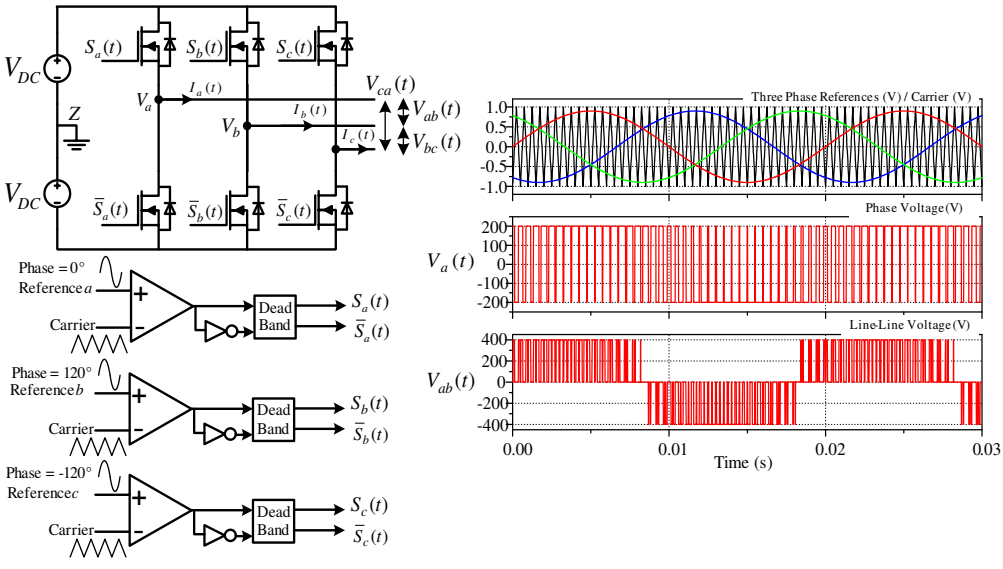


Figure 2.4: Open-loop modulation of two-level three-phase VSI using sine-triangle PWM

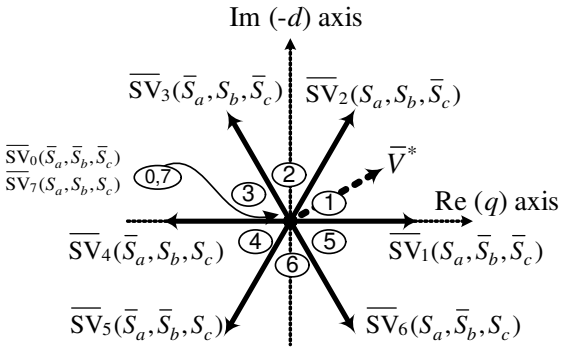


Figure 2.5: Space vector diagram for 2-level VSI inverter with the eight possible switching combinations

In order to achieve the target output voltage vector, SVM must meet the following criteria [11]:

- Select the three nearest space vectors adjacent to the reference vector.
- The switching cycle consists of four successive switching states between these space vectors.

In this way, the output voltage vector can be expressed as:

$$V^* = d_x S\bar{V}_x + d_y S\bar{V}_y + d_z S\bar{V}_z \tag{2.1}$$

where $\overline{SV}_x$, $\overline{SV}_y$ and $\overline{SV}_z$ are the three nearest voltage vectors to be applied and d_x, d_y and d_z are the duty cycles for these vectors. Once the three nearest space vectors are selected, optimum harmonic performance can be achieved by centring the active pulses of the voltage source inverter within one switching cycle, known as centred space-vector modulation (CSVM) [15].

To achieve the same level of harmonic performance with carrier-based PWM, an appropriate common mode offset can be injected into the three-phase modulation commands in a three-phase system [11][16]. The common mode offset is readily calculated [11] from the three-phase sinusoidal modulation commands shown in Figure 2.4, as follows:

$$V_{offset} = \frac{\max(V_a + V_b + V_{c,}) + \min(V_a + V_b + V_c)}{2} \tag{2.2}$$

2.2 Multilevel Voltage Source Inverters and Their Open-Loop Modulation

2.2.1 Topological Overview of Multilevel Inverters

Multilevel inverters are constructed by cascading power switches in a series or parallel combination to create converter topologies principally for high power applications. Over the last three decades, many multilevel topologies have been reported in the literature [6][7][8][18]. The three major emerging topological families include: flying capacitor (FC), neutral point clamped (NPC) and cascaded H-bridge (CHB) multilevel inverters. One should also note that the five-level cascaded H-bridge is constructed by a series combination of two three-level H-bridges [6][7]. However, in this thesis, the topology of a five-level cascaded H-bridge topology is not considered, and so the remainder of this section will review the three-level FC and NPC topologies only.

Figure 2.6(a) shows the structure of a three-level single-phase leg neutral point clamped inverter proposed by Nabae et al. [19]. Each phase leg is the combination of two two-level half bridges stacked on top of each other. The outputs of each half bridge are constrained by two series clamping diodes. The neutral point (N) is formed by connecting the mid-point of the clamping diodes to the middle of the bus capacitors. In this way, the switched output voltage has three distinct levels as it generates the output voltage values of $+V_{DC}$,$-V_{DC}$ and $0V$. It should be noted that the zero voltage level is only achieved by the switching combination of $(S_{1a}, S_{2a}) = (0,1)$, since the switching combination $(S_{1a}, S_{2a}) = (1,0)$ is not allowed because it generates a high impedance state [7][11][15][19].

Figure 2.6(b) extends this idea to a three-level three-phase NPC VSI where the three phase legs are connected in parallel across a common DC bus input voltage and bus capacitors. Also, the mid-points of the three-phase series clamping diodes and the input capacitors are connected together. In this configuration, the VSI generates three-level switched phase and five-level line-line voltages. The major advantages of this topology are simple construction and low bus capacitor count. However, this topology requires balancing of the neutral point voltage to prevent it drifting away from the zero voltage [28][29][30]. Additionally, as the number of the output voltage levels increases, the number of the neutral points and the clamping diodes

significantly increase, which limits the industrial applications for higher level diode clamped VSIs [7][8].

Figure 2.7(a) shows the structure of a three-level single-phase leg flying capacitor inverter proposed by Meynard et al. [20]. This topology is similar to a three-level NPC VSI except that that the clamping diodes are now replaced by the flying capacitor. The output zero voltage level is generated by connecting the load to the positive or negative DC bus voltage by means of the flying capacitor with the opposite polarity with respect to the DC bus voltage. In contrast to the NPC VSI, the zero voltage level is achieved by the two switching combinations of $(S_{1a}, S_{2a}) =$

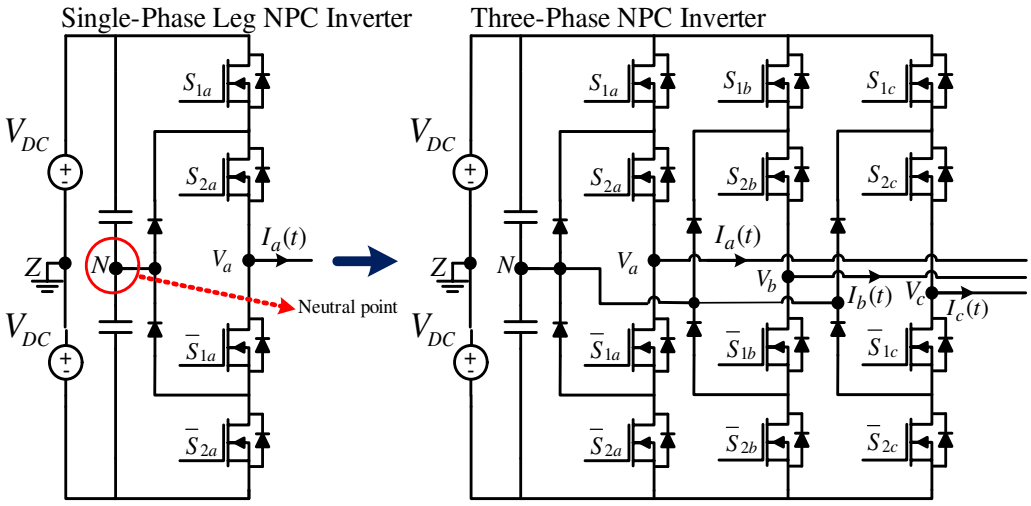


Figure 2.6: Topology of a three-level (a) single-phase leg NPC inverter (b) three-phase NPC inverter

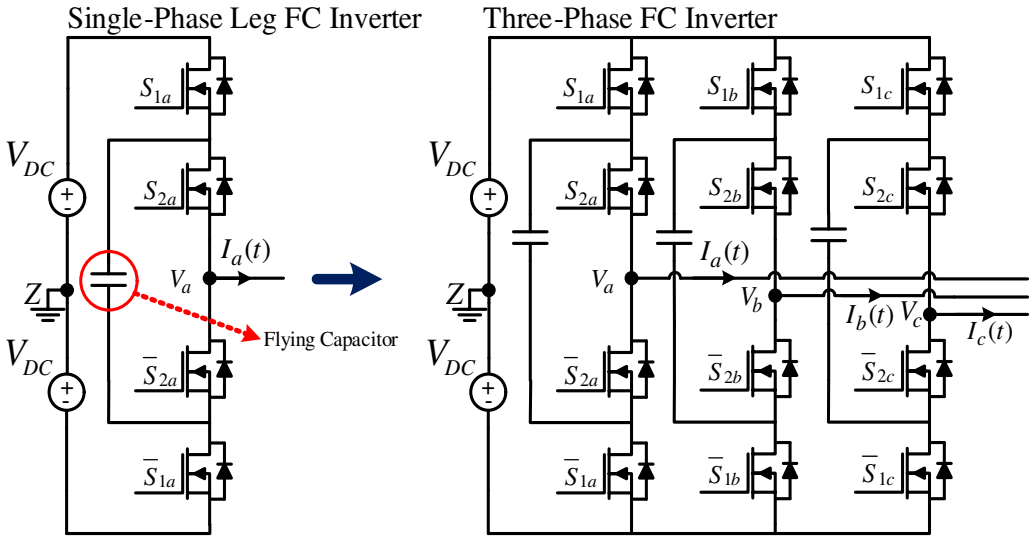


Figure 2.7: Topology of a three-level (a) single-phase leg FC inverter (b) three-phase leg FC inverter

$(0,1)$ and $(S_{1a}, S_{2a}) = (1,0)$, known as voltage level redundancies [7][20][23]. This voltage level redundancy is of particular advantage for modulation of a FC inverter. McGrath et al. [36] has shown that the round robin assignment of the inverter zero states can be used to balance the flying capacitor voltage without using additional voltage sensors.

Figure 2.7(b) extends this idea to a three-level three-phase FC VSI where the three phase legs are connected in parallel across a common DC bus input voltage. The major advantages of this topology are its modular structure and the ability to work without an isolation transformer [6][7][8][11].

2.2.2 Open-Loop Modulation of Multilevel Inverters

The next step in reviewing multilevel inverters is to consider their open-loop modulation. The identification of the optimal open-loop modulation performance for a multilevel inverter sets the goal for extending two-level hysteresis current regulation to multilevel inverters while still achieving a harmonic performance similar to open-loop modulation. Similar to two-level inverters, a multilevel inverter [8][11] can be modulated using either sine-triangle PWM or space vector based PWM. The most common sine-triangle PWM strategies for multilevel inverters [7][8][11][24] are: (a) phase shifted carriers (PSC) (b) level shifted carriers (LSC).

As shown in Figure 2.8(a), PSC PWM is a natural extension of two-level sine-triangle PWM of two-level inverters. It is a common modulation technique for a flying capacitor VSI due to the modularity of this topology [7][8]. The three-level single-phase leg FC inverter is made up of two power cells (also known as FC cells). The number of triangle carriers is dependent on the number of FC cells using the relationship $360/k$ where k is the number of cells.

Similar to a two-level three-phase VSI, when this strategy is applied to a three-phase FC VSI, three separate modulation commands displaced by 120°, are used with the same carrier arrangements for each phase leg. Previous work [24] has shown that for a three-phase system, its line to line voltage harmonic performance is suboptimal in comparison to phase disposition PD LSC PWM. However the advantage of PSC PWM when applied to FC inverters [22][23] is natural balancing of the flying capacitor voltages since this ensures equal power loss sharing between the power cells.

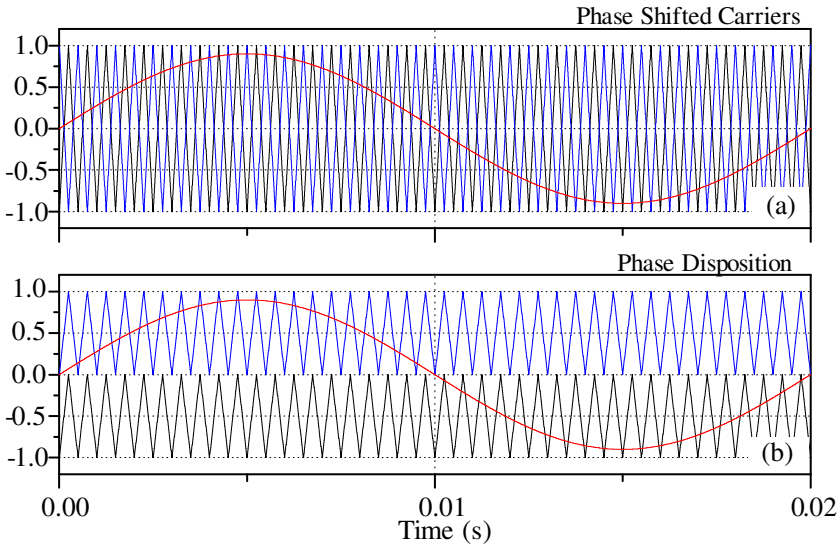


Figure 2.8: Open-loop sine-triangle modulation of multilevel inverter
(a) PS PWM (b) PD PWM

The alternative carrier-based open-loop modulation strategy for multilevel inverters is level-shifted PWM (LSPWM), first proposed by Carrara et. al. [27]. Depending on the number of output voltage levels *(n)*, there are *n-1* triangular carriers stacked on top of each other. Each carrier is shifted equally to form a continuous band within the full linear modulation range, with each of these carriers belonging to particular voltage levels.

Figure 2.8(b) shows the carrier arrangements for phase disposition (PD) LS PWM, where all carriers are in phase across all bands. Previous works [11][24][27] [33][34] have shown that PD-LSPWM achieves the optimal line-line harmonic spectrum for three-phase multilevel inverters in preference to POD and APOD PWM. PD-LSPWM is specifically advantageous for the NPC inverter since each of the carriers belongs to two power switches of the VSI [7][8]. Additionally, PD PWM guarantees to facilitate the natural balancing of the neutral point voltage for NPC inverter [28][31][32].

Figure 2.9 shows the switched phase and line to line voltages for PD PWM and PSC PWM of a three-level VSI. From this figure, PD PWM achieves a clear five-level line to line voltage level transition when compared to the PSC PWM. Also, the work by McGrath et. al. [35][36] has shown that the "round robin" assignment of the FC ZERO switching redundancy states using the PD-LSPWM guarantees balancing of the FC voltage while maintaining the harmonic benefits of PD PWM.

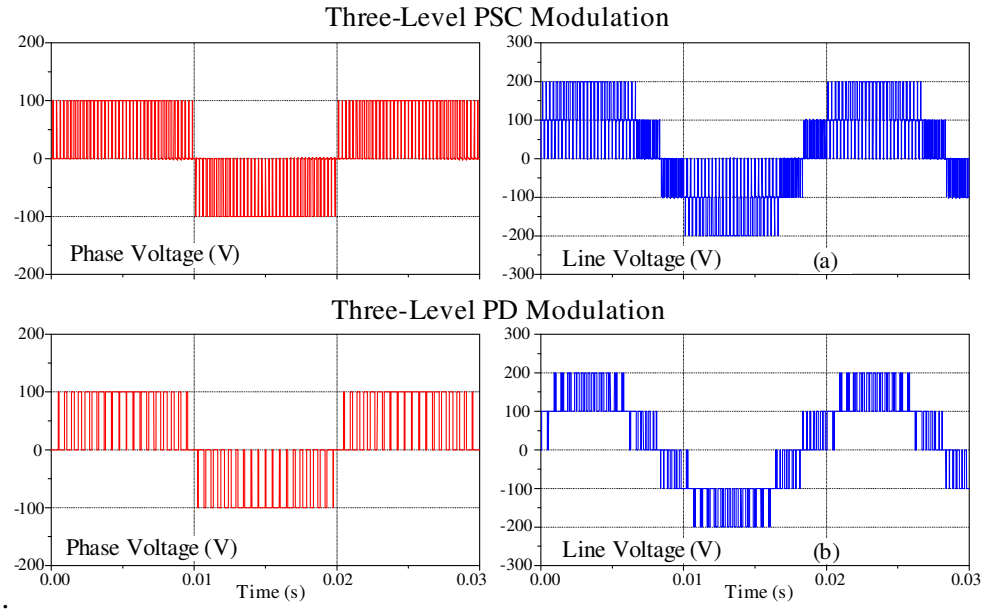


Figure 2.9: (a) Phase voltage and line voltage of three-level three-phase FC inverter under PSCPWM (b) Phase voltage and line voltage of three-level three-phase NPC inverter under PD-LSPWM (switching frequency = 2kHz)

The concept of space-vector modulation of a two-level inverters can also be extended to multilevel inverters [37][38][39]. McGrath et. al [15] extended the method of injecting the common mode offset for carrier-based PWM to multilevel inverters. This further improves the harmonic performance of the VSI while it reduces the computational time performed by the digital signal processor (DSP).

From this review, it is concluded that the new multilevel hysteresis current regulator must achieve line to line harmonic performance similar to open-loop PD PWM, if it is to be a useful and comparable alternative approach.

2.3 Current Regulation of Voltage Source Inverters

For many applications of voltage source inverters, it is important to regulate the inverter AC output currents using a closed loop controller. These applications include grid connected inverters [45][46], motor drive systems [47][48][49][50], uninterruptable power supplies (UPS) and AC power supplies [40][41]. For many of these applications, the quality of energy conversion highly depends on the quality of the current regulation employed, to regulate the AC output current. The current regulation approach for any power electronic inverter should have the following advantages [3][12][14][110]:

- Control of instantaneous input or output current with high accuracy
- Over-current protection
- Overload rejection
- Compensation of second order effects and robustness to variation of output load parameters
- Fast dynamic response
- Compensation of dead-time effect
- Compensation of DC-link voltage changes

Broadly speaking, the approaches that have been investigated over the last several decades to address this need fall into one of two categories [3][12][51][52][55] – linear and non-linear.

2.4 Linear Current Regulation of Voltage Source Inverters

Linear current regulation of inverters operates in conjunction with open-loop PWM. Figure 2.10 shows a generalized approach to linear current regulation that suits most of the previously mentioned applications. From this figure, the regulator separates the tasks of current control and modulation, so that the operation of the controller is independent of the inverter topology. However, depending on the inverter topology (either two-level or multilevel), the appropriate open-loop modulation strategy must be selected. The primary advantage of linear current regulators is their fixed switching frequency [3][4][51] which leads to superior

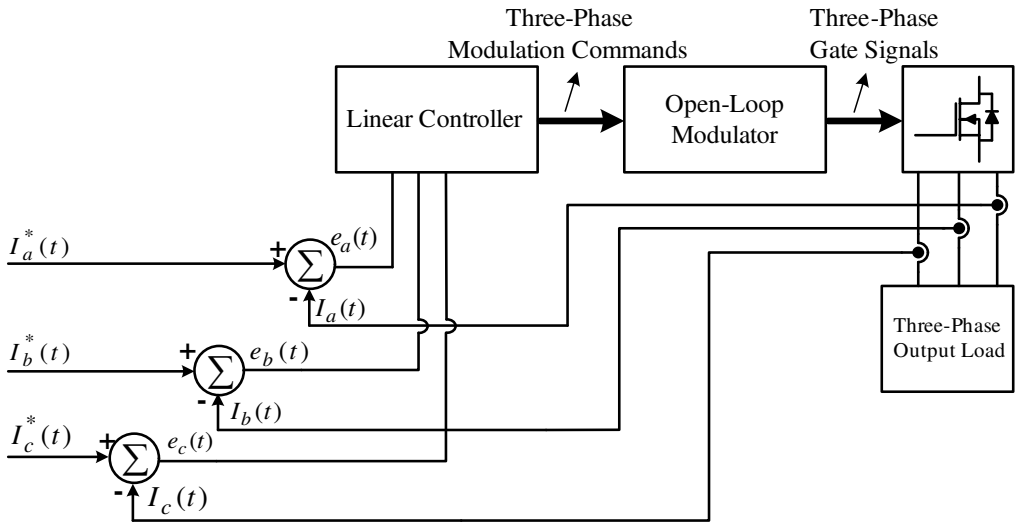


Figure 2.10: Basic block diagram of linear current-controlled PWM inverter

harmonic performance. Additionally when applied to a balanced three-phase system, they require only two regulators since an isolated neutral load is only a two-degree-of-freedom problem [52][53][54].

2.4.1 Stationary Frame Current Control

The most common type of stationary frame linear current regulators are the proportional + integral (PI) and proportional + Resonant (P+Resonant) controllers [52][53][54][55]. These types of current regulators can be either implemented in a direct stationary *abc* or *α-β* frame of reference [43].

PI current regulators consist of a proportional term and an integral term where the proportional term manages the high frequency system response and the integral term minimizes the steady-state error [44]. Due to the sampling and transport delay of the digital PI regulator, the controller gains cannot be set arbitrarily high which consequently causes a steady state tracking error [43][52]. Many approaches have been proposed to address this shortcoming by either including an auxiliary signal proportional to the load back-emf or an additional phase lead compensation in the modulation command [52][56][58]. However, they usually require reasonably precise knowledge of the output load parameters.

Another approach to solve the steady-state tracking error is to use a P+Resonant regulator. The integral term of the PI controller is replaced with a band pass resonant filter. This ensures an almost infinite gain at the resonant frequency which is essential to eliminate steady state tracking error [53][54][55]. The transient response

of this type of regulator is similar to that offered by a conventional PI regulator [52][53].

2.4.2 Synchronous Frame Current Control

DC tracking errors and phase delay may not be acceptable in many industrial applications since they can cause unstable system operation [51]. To overcome this problem, current control techniques based on the synchronous d-q frame were proposed [55][60][62]. In such regulators, the stationary reference frame of the three-phase currents is transformed into DC rotating synchronous frame d-q components [61][62]. Synchronous frame linear regulators can use either a PI or P+R controller and apply it to these components to reduce the errors of the fundamental components. Previous work by Zmood et. al. [54] has shown that the frequency response of a stationary frame P+R controller is essentially identical to a synchronous d-q frame PI controller. However synchronous frame regulators are rather more complex as they require transformation of the AC measured signal to a rotating DC frame and then transformation of the control action signals back into the stationary AC frame for the modulation action. Also, this strategy does not compensate for the coupling between the d and q axes where a step reference change in the d axis affects the q axis or vice versa during the transient [55].

2.5 Non-Linear Current Regulation of Voltage Source Inverters

Non-linear current control of VSIs using techniques such as hysteresis current regulation combine the task of current control and the modulation as shown in Figure 2.11. The appropriate gate signals are then directly generated by the instantaneous comparison of the phase current error against a hysteresis band, or alternatively by using time information of the current error zero-crossings. Hysteresis current regulation of VSIs offers several advantages compared to linear current regulation as follows [3][12][51][55][110]:

- Inherent over-current protection
- Robustness to load/filter parameter variation
- Very rapid dynamic response
- Direct compensation of second order inverter distortion effects

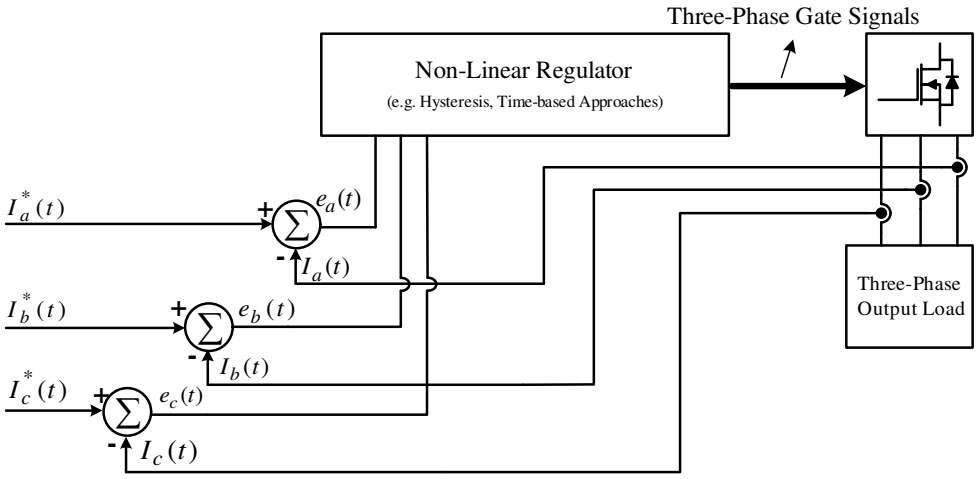


Figure 2.11: Basic block diagram of a non-linear current-controlled PWM inverter

However, it does suffer from two major drawbacks [3][14][51][67]:

- The switching frequency of the inverter varies due to the influence of the load average voltage.
- For three phase systems, there is an adverse interaction between phase legs that causes the current error to occasionally exceed the target limits.

Many variations of hysteresis current regulation have been proposed in the literature to solve these issues. The application of hysteresis approaches to two-level inverters is reasonably well established. However, their harmonic performance is not comparable to those offered by CSVM linear regulation. Additionally, these approaches cannot be easily extended to multilevel strategies and their extension to three-phase multilevel inverters is still a challenge.

2.5.1 Hysteresis Current Control of Two-Level Single-Phase VSI

A conventional fixed band hysteresis current controller (HCC) [3][51] for a single-phase VSIs is shown in Figure 2.12. This type of HCC (also known as a bang bang controller) is one of the simplest and most robust current regulators available. In this strategy, the physical output current of the VSI is measured using a current sensor (LEM) and is then subtracted from the desired reference current to create an instantaneous current error. The current error is then compared against fixed hysteresis bands using a comparator, to generate the switching signals $S_a(t)$ and $\bar{S}_a(t)$ as shown in Figure 2.12. If the current error becomes more positive than the

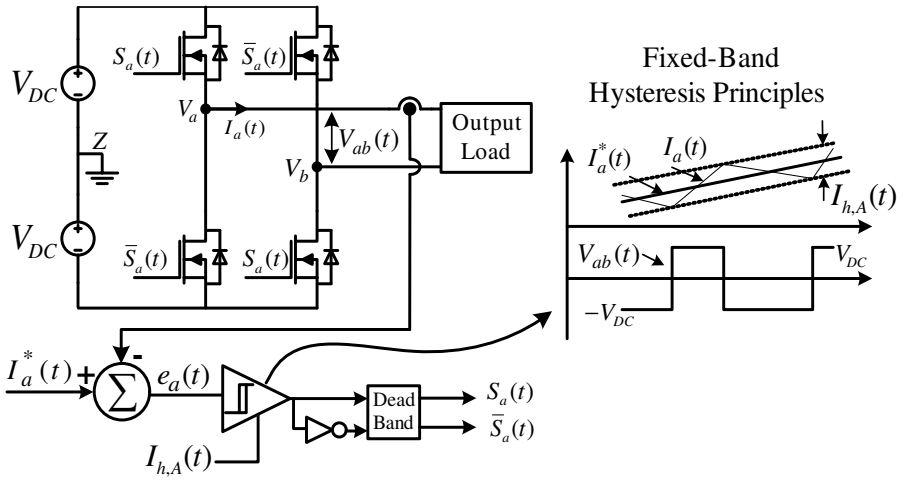


Figure 2.12: Conventional two-level hysteresis current-controlled single-phase VSI

upper band, the output voltage switches to the negative DC bus voltage to force the inductor current to ramp down. If the current error becomes more negative than the lower hysteresis band, the output voltage switches to the positive DC bus voltage to force the inductor current to ramp up. In this way, the output current tracks the reference current by direct switching of the VSI while maintaining the current error within the hysteresis bands.

The main disadvantage of this type of controller is the inverter variable switching frequency, which is largely dependent on the variation of the output load back-emf [3][12]. This is an important factor when designing output filters or calculating the switching losses since the controller creates an irregular spread harmonic spectrum [3][51].

2.5.2 Hysteresis Current Control of Two-Level Three-Phase VSI

The controller in Figure 2.12 can be easily extended to a three-phase VSI using an independent hysteresis controller for each phase leg as shown in Figure 2.13. Brod. et. al. [51] first identified that for an isolated load such as a grid-connected or a motor load, where the load neutral point is not connected to the mid-point of the inverter DC supply voltage (point Z), there is an adverse interaction between the three phase legs. This interaction forces the current error to exceed as much as twice the fixed hysteresis band which causes irregular switching of the inverter [66][67]. Hence, HCC of three-phase VSIs must be able to compensate for this interaction.

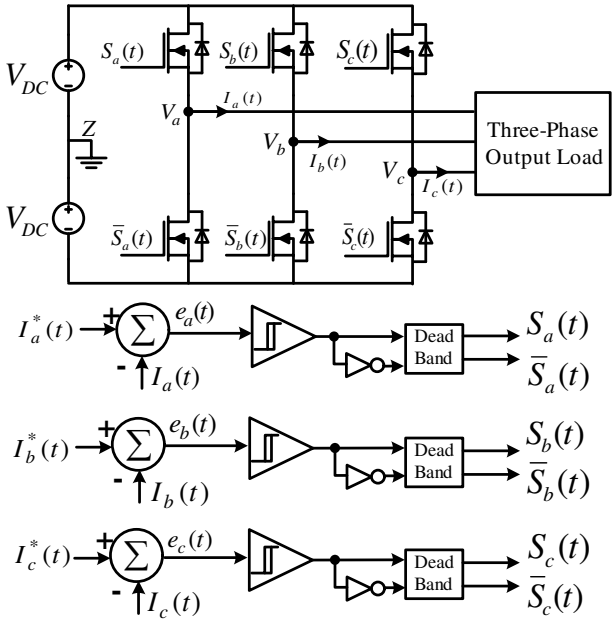


Figure 2.13: Conventional two-level hysteresis current-controlled three-phase VSI

2.5.2.1 Ramp Comparison Hysteresis Current Control

Figure 2.14 shows the block diagram of a ramp comparison HCC. In this arrangement, the regulator operation is similar to linear sine-triangle PWM [51] with a ramp comparison controller that is the same as a conventional HCC and the fixed hysteresis band is replaced with the triangle carriers. The current error is then compared against a constant frequency triangular carrier using hysteresis comparator to generate the appropriate gate signal. The primary advantage of ramp HCC compared with conventional HCC is its constant switching frequency. However this is at the expense of output current amplitude and phase error [69] unless the operating switching frequency of the carriers are chosen carefully [70]. Additional hysteresis bands can also be placed around the triangular carriers to prevent over-crossings of the current error with respect to the hysteresis bands [68][69].

Previous work [70] has shown that the addition of this band will not solve the problem of over-crossing and it may also cause under-crossings. Additionally, the placement of zero voltage vectors by this controller isolates the output load from the inverter at several instants over a fundamental cycle [68] which results in suboptimal harmonic performance [11][16].

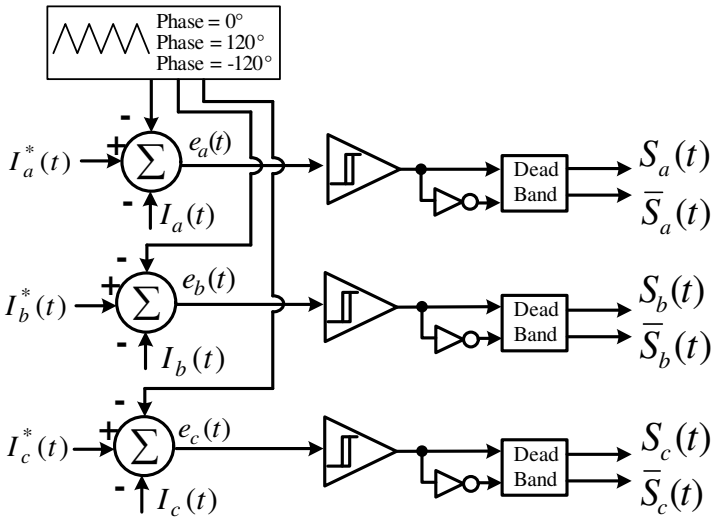


Figure 2.14: Block diagram of a ramp comparison hysteresis current controller

2.5.2.2 Variable Band Hysteresis Current Control

Constant operating frequency of the inverter using HCC can be achieved by varying the fixed hysteresis band in a sinusoidal form, according to the change in the load back-emf [67][71]-[76].

Yao. et. al. [67] has presented variable band (VB) HCC to vary the hysteresis band using the derivative of the current error as shown in Figure 2.15. The proposed strategy can be considered as an exemplar representation of the state-of-the-art for VB hysteresis strategies. It is mathematically expressed as [67]:

$$I_{h,a} = I_{h,\max}\left[1-\left(\alpha V_a(t)+\beta\frac{d\zeta_a(t)}{dt}\right)^2\right] \tag{2.3}$$

where $\alpha=\frac{1}{V_{DC}}$, $\beta=\frac{L}{V_{DC}}$, $I_{h,\max}=\frac{V_{DC}}{4Lf_{sw}}$

Equation (2.3) is a function of the input supply voltage V_{DC}, phase leg switched voltage $V_a(t)$, load inductance L and the derivative of the current error $\frac{d\zeta_a(t)}{dt}$. The variable hysteresis band calculation requires the current error derivative, which is known to be susceptible to noise [101][109], particularly at higher switching frequencies. Additionally, this strategy is highly dependent on the load inductance to correctly estimate the load back-emf for the band calculation. Finally, this strategy requires complex analog experimental implementation for the parameter measurement [73].

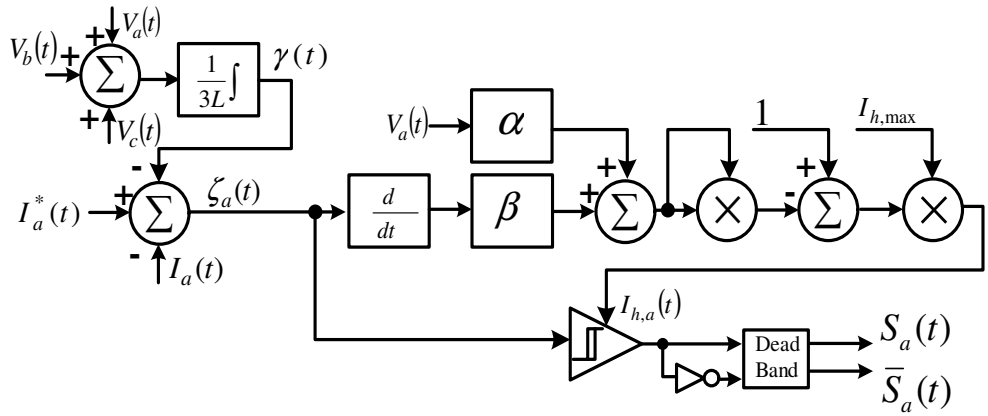


Figure 2.15: Block diagram of a per phase variable band hysteresis current controller for a two-level three-phase VSI

Work by Malesani et. al [77][78] presented a VB HCC using a self-adjusting analog prediction of the hysteresis band and a phase-locked-loop (PLL). This ensures better switching frequency regulation by synchronising the three-phase active voltages to a fixed reference clock. However synchronisation in this way is still not particularly accurate due to the bandwidth limitation of the PLL [78].

Another form of VB HCC is a zero average current error (ZACE) regulator proposed by Borle et. al [79][80]. For this strategy, the area of the triangular current error within one switching cycle is divided into positive and negative excursions and the controller must ensure that the previous and the present excursion areas are equal. This ensures a constant switching frequency irrespective of the reference current waveform. However in a three-phase system, the controller only regulates two currents out of the three during any sextant, which is equivalent to 60° discontinuous open-loop modulation. This is known to have a suboptimal harmonic performance in general in comparison to continuous modulation techniques [11][16]. Additionally, commutation from one sextant to the other causes a disturbance that appears as a spike in the output current [79][80]. At this point, the controller loses the current error zero-crossing which is necessary to ensure ZACE operation during each switching cycle [81].

The second major issue with conventional HCC of three-phase system is the interaction between the phase legs caused by the load neutral point voltage. To solve this issue, previous works [66][67] have shown that for an isolated load system, the phase leg current error can be split into two components, the non-interacting current error and the interacting current. The interacting current is a common mode across all

three phase legs and is calculated by the integration of the summation of the three-switched phase voltages. This requires voltage sensors in addition to the current sensors to directly measure the switched three-phase voltages. This current is then subtracted from the per phase current error before making the final hysteresis switching decision. Another strategy is to use the delta equivalent current errors to control the three-phase output star-connected load currents [79][80][81]. However, for a delta equivalent HCC, two of the three-phase currents must be regulated during any sextant [79][80] and the controller must ensure to select the correct two delta currents since its operation becomes identical to the ZACE HCC discussed previously.

2.5.2.3 Time-Based Hysteresis Current Control

Another approach to achieve constant switching frequency is time-based hysteresis current regulation [81]-[85]. The principle of time-based (TB) HCC is to determine the zero-crossing time instant of the phase leg current error and calculate the next target switching time. Previous work by Bode et. al. [81] has shown that there is no difference between time-based HCC and variable band HCC and the selection of either of these approaches depends mainly on the available hardware and software. However, the primary advantage of TB HCC is achieving centred three-phase voltages that are synchronized to a fixed reference clock. This better aligns the phase leg switching processes so that it only uses the "three nearest space vectors" within each switching cycle [81][84]. This achieves a similar harmonic performance to CSVPWM [11]. Another form of TB HCC [84] uses the current error zero-crossing time information to predict the next hysteresis band required for the next switching event. This approach is quite effective at maintaining a constant switching frequency but its performance is affected by dead-time [81]. Also if this delay becomes more than a few percent of the switching cycle, it requires extra compensation by adjusting the hysteresis band or the switching time events [81][84]. Another issues with TB HCC is the frequency jitter caused by the analog circuitry that is used to detect the current error zero-crossings and the sampling process associated with the digital implementation [85]. In [85], a fully digital TB HCC is presented for a two-level single phase VSI. The controller predicts the next switching transition time using the time information of the previous cycle on time and off time while incorporating the system nonidealities such as sampling delay and dead-time.

However, this approach requires high computation speed and signal processing capability, which makes it suitable for an implementation in FPGA.

Additionally, for an isolated three-phase system, improper compensation of the common mode current still prevents the controller achieving accurate synchronisation of the three-phase legs [66].

2.5.2.4 Space-Vector Based Hysteresis Current Control

The general block diagram of a simple SV-based HCC [86]-[93] is shown in Figure 2.16 [89]. The SV-based HCC includes four main functional blocks as follows [93]:

- Set of summing elements to calculate the derivative of the current error
- Set of hysteresis comparators (usually two comparators per phase leg)
- Switching table that is dependent on the level of switching
- Coordinate transformation of the three-phase currents

The primary advantage of SV-based hysteresis current control is the regulation of two currents out of the three-phase currents in a similar way as a linear regulators [53]. This achieves a better performance than a conventional hysteresis controller that uses three independent controllers [51]. Additionally, SV-based HCCs are claimed to remove the three-phase interactions caused by load neutral point voltage while reducing the inverter switching frequency variation [87][92][93] (this claim will be disproven later in this thesis).

The SV-based hysteresis strategies presented in [86][87][88] use the derivative of the current error to estimate the output load back-emf. From this estimation and a

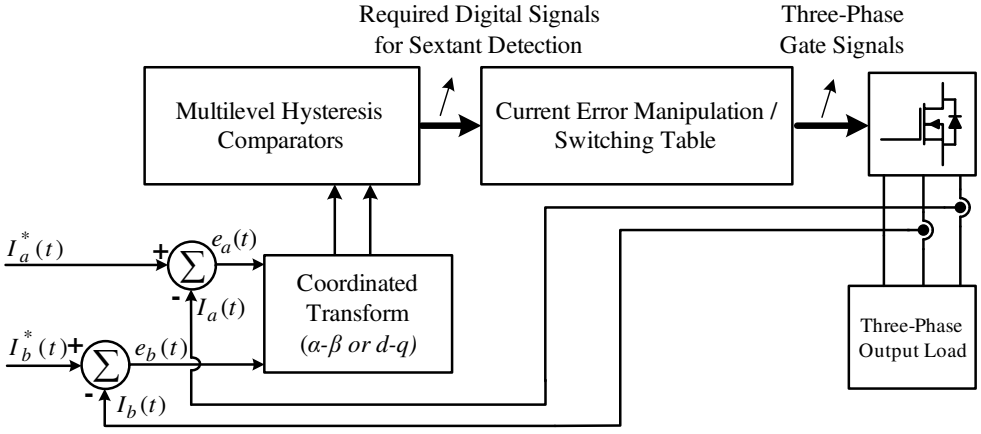


Figure 2.16: Block diagram of a SV-based hysteresis current control

switching table, the appropriate voltage vector is selected to minimize the current error vector. In [87], a stationary frame and a synchronous frame SV-based HCC is proposed. For both of these cases, each of the current error components use a three-level HCC that uses two hysteresis comparators per phase leg. The space vector plane is divided into 24 sections and the controller chooses the nearest voltage vector based on the digital outputs of the hysteresis comparators and the switching table. Another strategy [90] uses the stationary frame current errors and a fixed band single hysteresis comparator to create the α-β switching signals. These signals and the sign of the current error derivatives determine the operating region within the SV plane. Although these controllers are able to maintain the current error within the tolerance region, the zero voltage vectors are not symmetrically utilized and the presence of redundant states causes a suboptimal switching pattern. In [91] a region detector is proposed which does not require the information of the load back-emf. This strategy uses the direct *abc* phase currents and two hysteresis comparators per phase leg to achieve the advantage of both the conventional hysteresis controller and the space-vector HCC. However the controller does not maintain a constant switching frequency which consequently does not guarantee a symmetrical distribution of the zero voltage vectors.

Recent work by Mohseni. et. al. [93] proposes a new stationary frame SV-based HCC that uses four three-level hysteresis comparators per the α and β components of the current error. The comparator switching state allows the controller to choose the closest voltage vector from the two available non-zero voltage vectors and the two zero vectors to allow a systematic voltage selection process. This reduces the switching frequency variation of the inverter while keeping the current error within the tolerance region. However, the operation of this strategy is only verified in simulation due to the implementation complexity of multilevel hysteresis comparators.

2.5.2.5 Flux Modulator Based on Hysteresis Current Control

Carrier-less flux modulation of VSIs (also known as the fish method) was first introduced by Veltman et. al. [94]. This strategy can be considered as a SV based HCC where instead of direct voltage control, it uses flux equivalence as the integration of the voltage. In this strategy, the three-phase fluxes are transformed into the synchronous frame using a rotating fixed-band bounding box and a simple set of

switching rules are used to apply the nearest space vector and the null vectors in each sextant. However, this strategy still suffers from the presence of a common mode interacting current. The flux modulator approach has been extended to a multidimensional flux vector modulator for a four-phase-leg inverter [96][97] and its performance has been verified under various operating conditions such as unbalanced and non-linear loads. In [95] a flux modulator for a four-switch three phase inverter is proposed. The proposed controller reduces the harmonic torque ripple and the switching frequency variation, but the application of zero voltage vectors is not possible in the four-switch configuration and hence its voltage harmonic performance is suboptimal compared with CSVM [11]. Previous works by Loh. et. al [96][98] proposed a common mathematical basis for the flux modulation of a VSI and charge modulation of CSI. In [98] a variable band bounding box flux modulator and a set of optimized switching rules is presented to improve the harmonic performance of the proposed controller.

Another variation of the flux modulator is the flux-based deadbeat HCC proposed in [99]. This type of controller combines a flux modulator and a deadbeat hysteresis current controller that is capable of controlling the positive and negative currents. The deadbeat controller predicts the amounts of flux change required over a sampling interval to further reduce the additional flux error and hence keep the flux within the tolerance region.

2.5.3 Hysteresis Current Control of Multilevel Inverters

The target objectives of the previously reported multilevel hysteresis current regulators are as follows [100][106][110][115]:

- Maintain constant switching frequency.
- Compensate for the common mode current.
- Differentiate the adjacent switched phase voltage levels in order to bring the current error toward zero once it exceeds the hysteresis bands.
- Generate the switching signals similar to open-loop PD PWM.

The most well-known multilevel hysteresis strategies that have been reported in the literature can be categorized as follows

- Multiple band (MB) and multiple offset (MO) HCC [100][102][107][110] for single-phase leg and single-phase multilevel inverters.

- Time-based (TB) HCC [100][109] for single-phase leg and single-phase multilevel inverters.
- SV-based hysteresis current control for three-phase multilevel inverters [113][117][114][119]

2.5.3.1 Multiple Band and Multiple Offset Multilevel Hysteresis Strategies

Multiple band (MB) HCC, first proposed by Marchesoni et. al. [100] and later in [101]-[106], uses *N-1* sets of hysteresis bands for a *N* level inverter. Figure 2.17 shows the arrangement of hysteresis bands for this approach. The inner set of hysteresis bands are responsible for switching between the adjacent voltage levels and the outer hysteresis bands are responsible to detect the necessary level change. When the phase current error exceeds the inner hysteresis bands, the controller changes the inverter phase voltage level to increase or decrease by one step. If this action is not sufficient to bring the current error within limits, it will continue to hit the next hysteresis band until the error changes its direction. In [101][102][103] a MB hysteresis approach is presented for the three-level single-phase leg NPC and a five-level single-phase leg cascaded H-bridge. The strategy uses the derivative of the current error and sequential logic to determine whether the applied voltage level is sufficient to bring the current error back to zero. However, this strategy does not maintain constant switching frequency which is necessary to facilitate the natural balancing of the NPC VSI [28]. In [105], a variable MB HCC for the FC inverter is proposed to keep the switching frequency constant. The controller uses additional voltage sensors to implement an active strategy to maintain a balanced FC voltage. However this controller requires a complex experimental implementation and its operation is only verified in simulation.

Figure 2.17 shows the arrangement of the hysteresis bands for a Multiple offset (MO) HCC [107][110][111]. Similar to MB strategy, it uses *N-1* offset bands for an *N* level inverter where any set of offset hysteresis bands is responsible for adjacent voltage levels. Whenever the phase current error crosses an outer hysteresis band, the inverter changes its output voltage level by one to change the direction of the current toward zero. However, since the strategy works based on the offset placement around zero current error, it introduces steady-state tracking error, which then needs proper DC offset compensation depending on the output voltage level [110]. This makes the implementation of this approach more complex especially at higher voltage levels.

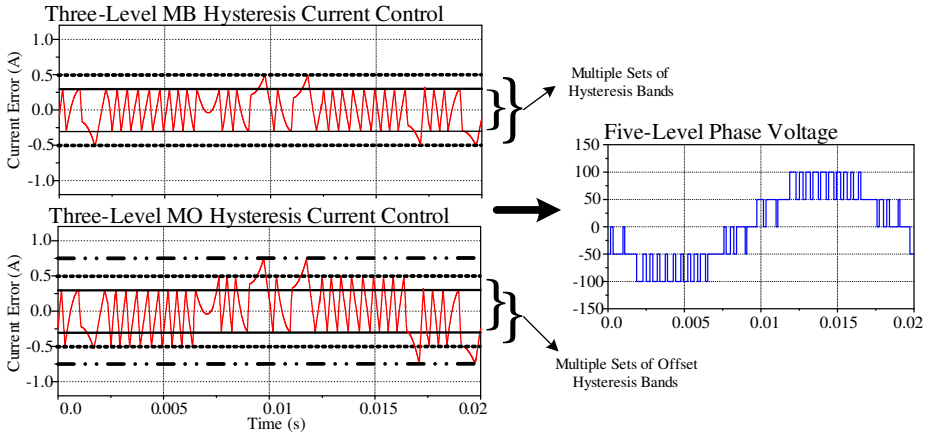


Figure 2.17: Current error and phase voltage of a five-level inverter using MB and MO hysteresis current control

In [110][111], a modified MO strategy is presented to solve this issue by remembering the previous applied voltage levels and hence locking the current error within the correct hysteresis boundaries. Unfortunately, the load back-emf for this experiment is negligible to maintain a constant switching frequency, which is not the case for most practical applications.

2.5.3.2 Time-Based Multilevel Hysteresis Strategies

The complexity of MB and MO HCCs leads to the extension of two-level TB HCC to multilevel TB hysteresis strategies as shown in Figure 2.18 [100][101][108]-[111]. The primary advantages of this approach are as follows[110][111]:

- It only uses one hysteresis comparator for multilevel inverters and extra digital logic that is implemented within the programmable logic devices (EPLD, CPLD, FPGA)
- It avoids DC offset compensation, which consequently avoids steady-state tracking error.

Time-based lock out hysteresis strategies for multilevel inverters were first proposed by Marchesoni. et. al. [100]. The strategy detects an out of band phase current error using digital logic which then adjusts the appropriate voltage level accordingly. However, this strategy is more prone to instability during transient and level change region due to the current error freewheeling as shown in Figure 2.18. In [108] a time-based hysteresis controller is proposed for a three-level H-bridge and later applied to the single-phase leg multilevel inverter [109] to improve its transient response and level selection by adding extra set of outer hysteresis bands. This

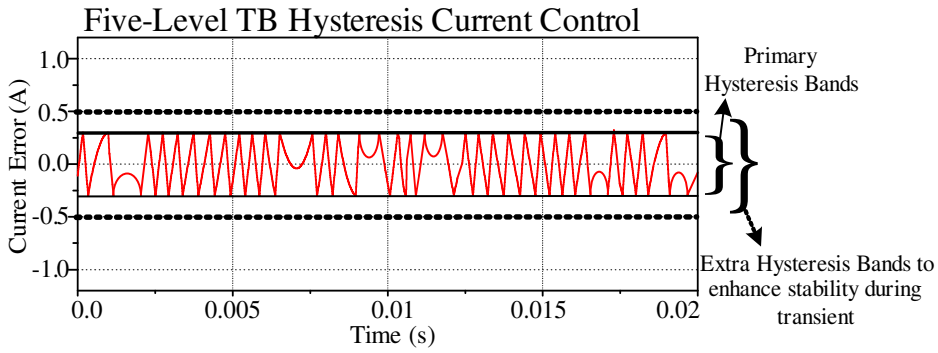


Figure 2.18: Current error of a five-level inverter using TB hysteresis current control

strategy uses the slope of the current error and the outputs of the two hysteresis comparators as the inputs to the sequential logic decoder to generate the VSI switching signals. The controller robustness is highly dependent on the correct slope detection of the current error particularly during the level change process [110][111]. In [110][111], a modified time-based HCC is presented which uses *N-2* hysteresis comparators for a *N* level inverter. The slope detection algorithm presented in [108][109] is replaced by a sign detection of the current error to improve the controller stability. Moreover, the strategy combines the horizontal and the vertical movement of the current error associated with MB hysteresis strategy to further improve the operation of the TB HCC of multilevel inverters.

2.5.3.3 SV-Based Hysteresis Current Control of Three-Phase Multilevel Inverters

Previous works have extended the concept of two-level SV-based hysteresis to multilevel inverters either in the form of current control [112]-[114] or flux modulator based HCC [119]. The controller in [112] estimates the load back-emf from the current error derivative to determine the correct voltage vector. However the controller lacks the ability to detect the correct sector in the SV plane and hence prevents the controller precisely tracking the output current. The problem is solved in [113] by avoiding the need for estimation of the load back-emf and choosing the appropriate adjacent voltage vector using a lookup table. In [114], a variable band SV-based HCC is presented for a five-level inverter formed by a dual three-level inverter. The operation of the controller has been verified for a low fundamental frequency ranging from 10Hz to 50Hz. At a low fundamental frequency, the controller keeps the switching frequency constant, but as the fundamental frequency increases, the switching frequency is not constant due to the poor estimation of the

stator back-emf. This prevents the controller selecting the three nearest space vectors within one switching cycle and hence its line-line harmonic performance is not comparable to that achieved by open-loop PD modulation.

In [117], a SV-based HCC is proposed for a three-level three-phase NPC inverter along with an active NP balancing strategy. The controller operation consists of four main modules: current error calculation, area detection, sector detection and voltage vector selection. Similar to a two-level SV-based HCC, the output of the multilevel hysteresis comparators select the appropriate area in the SV plane, and then from the angle of the current error vector, the correct voltage sector is selected. From this selection and the measurement of the neutral point current, an appropriate voltage vector is applied to ensure both current control and the balancing of the neutral point voltage. While the proposed controller is more robust than [114][118] in terms of sector selection, it still does not exactly achieve constant switching frequency operation and consequently has a harmonic performance that is inferior to open-loop PD PWM [11][15][34].

In [119], a new flux modulation approach for closed-loop voltage regulation of multilevel VSIs is developed that uses a HCC with a rectangular bounding box in the synchronous d-q frame. The controller extends the idea of a two-level flux modulation HCC [98] and applies it to multilevel inverters. The space vector plane for the multilevel inverters is divided into all possible two-level sub triangles and then it selects the nearest three space vectors to achieve an optimal space-vector sequence within one switching cycle. This work presents a superior harmonic performance in comparison to the previous multilevel SV-based HCCs. However, the common mode current is not properly compensated which may result in irregular switching (double switching) near the hysteresis bands.

2.6 Conclusion

This chapter has reviewed a topological overview of the two-level and multilevel single-phase and three-phase VSIs together with their common existing open-loop modulation strategies. From this review, it has been identified that the target switching objective of any current regulator for two-level VSI is to achieve harmonic performance similar to CSVPWM, and for a multilevel VSI similar to PD PWM.

Hysteresis current regulation of voltage source inverters offers particular advantages compared with linear current regulators as follows

- Inherent over-current protection
- Robustness to load/filter parameter variation
- Very rapid dynamic response

However, the existing HCC strategies suffer from the following major drawbacks:

- The switching frequency of the inverter varies due to the influence of the load average voltage.
- They require current error derivative and/or direct measurement of the current error zero-crossing to maintain constant switching frequency.
- For three phase systems, there is an adverse interaction between phase legs that causes the current error to occasionally exceed the target limits.
- They require estimation of the load back emf to apply the correct reference voltage vector.
- Additionally, for multilevel inverters, the output current has DC steady state tracking error due to the multiple hysteresis band arrangement to detect the phase voltage level change. In addition, they require a current error derivative or a sign detection algorithm to correctly detect the point of the inverter level change, which causes a poor dynamic response.

From this review, it is concluded that there is scope for the development of an integrated approach for constant switching frequency closed-loop hysteresis current regulation of VSIs to solve these shortcomings. The new hysteresis approach must achieve performance similar to open loop modulators and its fundamental concepts should be applicable to multilevel inverters with minimal modification.

Chapter 3

Hysteresis Current Regulation of a Two-Level Single-Phase Inverter

This chapter[1] presents a new approach for hysteresis current regulation of a single-phase two-level VSI. The chapter begins by showing how the average inverter output voltage is the major driver of variable switching frequency operation for a conventional HCC. A novel approach is then presented to synchronously measure the average output voltage of the VSI using the transition times of the inverter output switching events. This technique avoids both the susceptibility of derivative action on the measurement process and/or any requirement for analog or digital filtering. Next, this measured average voltage is used to develop a variable band hysteresis current regulator. The approach presented directly incorporates the influence of the average inverter output voltage into the variable band calculation while being both independent of current error derivative, and not requiring time information of the zero-crossings of the current error.

Current error zero-crossings are then synchronized to a reference clock to further improve the frequency regulation of the HCC, achieving with this combined strategy a harmonic performance that is very close to asymmetrical regular sampled PWM. However, the new approach calculates the zero-crossing time information from the switching time events to avoid the need for additional zero-crossing measurement circuitry. The synchronization technique is also further improved by compensating for the effect of dead-time in the inverter switching process.

Finally, the operation of the new HCC is extended into the overmodulation region by clamping the variable hysteresis band while maintaining switching stability.

Simulation and experimental results are presented at the end of this chapter.

---

[1] **Materials in this chapter were first published as**

1. Holmes, D.G.; Davoodnezhad, R.; McGrath, B.P.; "An improved three phase variable band hysteresis current regulator," *ECCE Asia (ICPE & ECCE), 2011 IEEE 8th International Conference*.
2. Holmes, D.G.; Davoodnezhad, R.; McGrath, B.P.; "An Improved Three-Phase Variable-Band Hysteresis Current Regulator," *Power Electronics, IEEE Transactions on*, vol.28, no.1, pp.441-450, Jan. 2013.

3.1 A Review of Variable Switching Frequency Operation

Figure 3.1(a) shows the topology of a two-level single-phase leg voltage source inverter. This combination of power switches and inverse parallel diodes is the simplest topology that can achieve a fundamental AC output voltage. Note also that the topology can be considered as an equivalent to a two-level single-phase H-bridge since the center tap point of the input voltages is not available in most practical applications. The inverter feeds into a series resistive and inductive load with an AC back-EMF, which is an effective representation of applications ranging from simple passive loads to grid-connected PV systems.

From Figure 3.1(a), the inverter load relationship can be written as:

$$V(t) = RI(t) + L\frac{dI(t)}{dt} + E(t) \tag{3.1}$$

where:

$V(t)$ is the phase-leg switched voltage.

$I(t)$ is the output current.

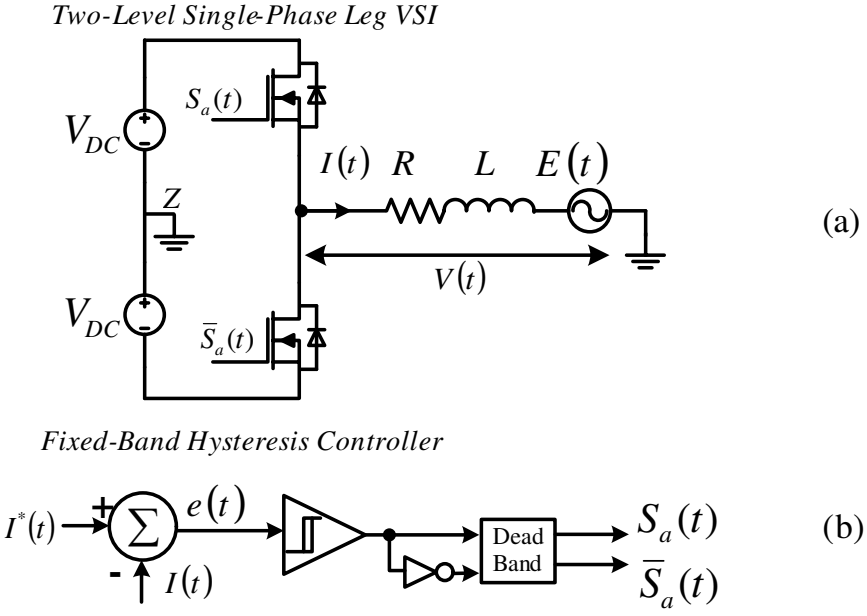


Figure 3.1: (a) Single-phase leg two-level voltage source inverter feeding a back-emf type load with series resistance and inductance (b) Block diagram of a two-level conventional fixed-band hysteresis controller

R is the load resistance.

L is the load inductance and,

$E(t)$ is the output load AC back-emf.

Assuming that the voltage across the series resistance is small compared to both the back-emf and the voltage across the series inductor, equation (3.1) simplifies to:

$$V(t) = L\frac{dI(t)}{dt} + E(t) \tag{3.2}$$

This equation is the starting point for almost all hysteresis current regulation strategies [67]-[78].

From Figure 3.1(b), it can be seen how the output load current $I(t)$ is subtracted from the reference current $I^*(t)$ to generate the phase current error $e(t)$. This error is then compared against a set of (fixed) hysteresis bands using a comparator to generate the appropriate switching signals. These switching signals feed into a dead-time unit (either a DSP or FPGA) to generate the complementary output switching signals $S_a(t), \overline{S}_a(t)$. These two switching signals in turn modulate the power switches and hence control the output current. The switched phase output voltage $V(t)$ can have the values of $\pm V_{DC}$.

Figure 3.2 shows the basic hysteresis switching process over one switching period for a two-level switched system. When the output current error exceeds the upper hysteresis band, the switching signal $S_a(t)$ becomes low, $\overline{S}_a(t)$ becomes high, and the phase voltage switches to the lower DC bus to reverse the current error direction towards zero. When the output current error exceeds the lower hysteresis band, the switching signal $S_a(t)$ becomes high, $\overline{S}_a(t)$ becomes low, and the phase voltage switches to the upper DC bus to drive the current error direction towards zero. As this process continues, it forces the output current to track a reference current independent of any second order effects such as fluctuation in the DC bus voltage and variation of output load parameters [51].

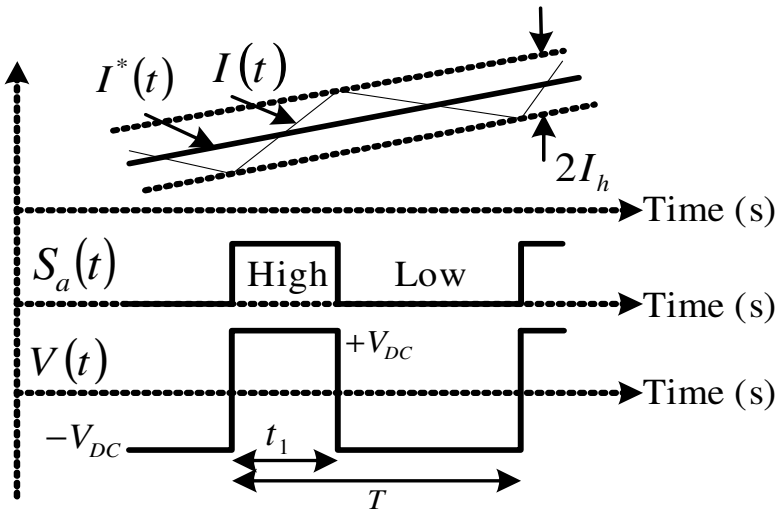


Figure 3.2: Basic principles of hysteresis current regulation process

Figure 3.3 shows the output current of a single-phase two-level VSI employing this conventional two-level hysteresis current control strategy. Careful examination of Figure 3.3 identifies that the output load current of the VSI, $I(t)$ can be separated into two components, as follows:

1. **Fundamental target component ($I_f(t)$):** This is the low frequency sinusoidal component of the output current without any switching ripple. The frequency (f_o) of this component is equivalent to the fundamental component frequency of the phase-leg switched voltage.
2. **Switching ripple component ($I_r(t)$):** This is the current ripple that is imposed on the fundamental component of the output current. The frequency (f_{sw}) of this component is equivalent to the switching frequency of the phase-leg switched voltage.

Using these two components, the output current can be expressed as:

$$I(t) = I_f(t) + I_r(t) \tag{3.3}$$

Substituting equation (3.3) into the load equation (3.2) gives:

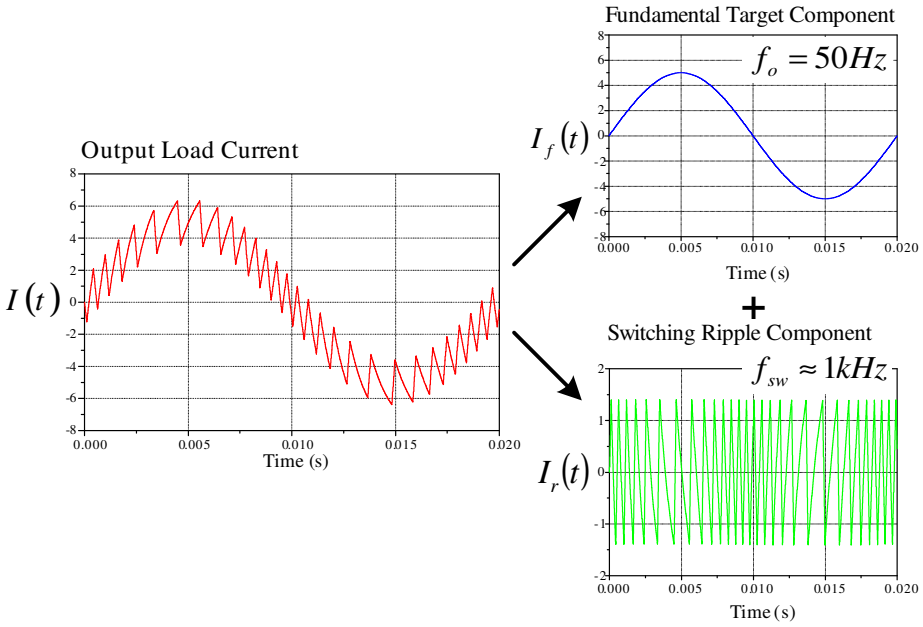


Figure 3.3: Output load current is split into two components being the fundamental target component and the switching ripple component.

$$\begin{aligned} V(t) &= L\frac{d\left(I_f(t)+I_r(t)\right)}{dt} + E(t) \\ &= L\frac{dI_f(t)}{dt} + L\frac{dI_r(t)}{dt} + E(t) \\ &= V_L(t) + L\frac{dI_r(t)}{dt} + E(t) \end{aligned} \tag{3.4}$$

In equation (3.4), $V_L(t)$ is the voltage drop across the load reactance caused by the fundamental target component of the output load current. Hence, equation (3.4) can be further simplified to:

$$V(t) = V_{avg}(t) + L\frac{dI_r(t)}{dt} \tag{3.5}$$

where $V_{avg}(t) = E(t) + V_L(t)$ and $V_L(t) = L\frac{dI_f(t)}{dt}$ (3.6)

In this form, equation (3.5) is the summation of the average fundamental voltage $V_{avg}(t)$ and the instantaneous voltage drop across the load caused by the switching ripple component of the output current. The average voltage is the summation of the load AC back-emf and the fundamental voltage drop across the load reactance and is

essentially equivalent to the fundamental voltage component of the inverter switched output voltage.

In order to identify the primary cause of variable switching frequency for a conventional hysteresis controller, the behaviour of the switching ripple component of the output current must be examined over one switching period. From equation (3.5) and Figure 3.2, the rate of change of ripple component of the current over one switching period can be written as: $\frac{dI_r(t)}{dt} = \frac{V(t) - V_{avg}(t)}{L}$

$$\frac{dI_r(t)}{dt} = \frac{V(t) - V_{avg}(t)}{L} \tag{3.7}$$

From Figure 3.2, the behaviour of the switching ripple can be separated into two components over one switching period, as follows:

1. **During the switching period ON-time** (t_1): The current ramps from the lower hysteresis band $-I_h$ to the upper hysteresis band $+I_h$ and the switched phase leg voltage is $+V_{DC}$.
2. **During the switching period OFF-time** ($T - t_1$): The current ramps from the upper hysteresis band $+I_h$ to the lower hysteresis band $-I_h$ and the switched phase leg voltage is $-V_{DC}$.

Expressing these switching behaviours mathematically and assuming that the average voltage component of the phase leg switched voltage is constant over one switching cycle, i.e. $V_{avg}(t) = V_{avg}$, the switching time intervals (t_1, $T - t_1$) can be expressed as:

During the switching period ON-time (t_1):

$$\frac{2I_h}{t_1} = \frac{V_{DC} - V_{avg}}{L} \tag{3.8}$$

$$\Rightarrow t_1 = \frac{2LI_h}{V_{DC} - V_{avg}}$$

During the switching period OFF-time ***($T - t_1$):***

$$\frac{-2I_h}{T-t_1} = \frac{-V_{DC} - V_{avg}}{L} \tag{3.9}$$
$$\Rightarrow T - t_1 = \frac{2LI_h}{-V_{DC} - V_{avg}}$$

Continuing the assumption that the average voltage V_{avg} is constant over one switching period, combining equation (3.8) and equation (3.9), and solving for the switching period T gives:

$$\begin{aligned} T - t_1 &= \frac{2LI_h}{-V_{DC} - V_{avg}} \\ T &= \frac{2LI_h}{-V_{DC} - V_{avg}} + t_1 \\ T &= \frac{4V_{DC}LI_h}{\left(V_{DC}^2 - V_{avg}^2\right)} + \frac{2LI_h}{V_{DC} - V_{avg}} \\ \Rightarrow T &= \frac{4V_{DC}LI_h}{\left(V_{DC}^2 - V_{avg}^2\right)} \end{aligned} \tag{3.10}$$

Finally, inverting equation (3.10) achieves an expression for switching frequency of:

$$f_{sw} = \frac{\left(V_{DC}^2 - V_{avg}^2\right)}{4V_{DC}LI_h} \tag{3.11}$$

where:

f_{sw} is the switching frequency.

V_{DC} is half of the total bus voltage.

L is the output load inductor.

V_{avg} is the average component of the inverter switched output voltage .

I_h is the hysteresis band.

For a conventional fixed-band hysteresis controller, the parameters of equation (3.11) including the filter/load inductance, the hysteresis band and the bus voltage are constant, except for the average inverter output voltage (which varies over each

fundamental cycle). Hence equation (3.11) identifies how the switching frequency of the inverter will vary as this fundamental component changes. Furthermore, from the mathematical analysis presented above, it can be seen how the switching frequency depends on the average load voltage. From equation (3.6), it can be seen that in the case of a small load inductance, the voltage drop across the filter reactance is small and the inverter average voltage is essentially equivalent to the output load AC back-emf. Hence, the switching frequency is primarily affected by the load back-emf, as is often conventionally assumed. In contrast, as the load inductance becomes more significant, the switching frequency will be affected by both this reactance and the AC back-emf.

This theoretical understanding has been verified in simulation using the simple hysteresis controller shown in Figure 3.1(b). Figure 3.4 shows the variation of the switching frequency over a fundamental cycle (50Hz), together with the average load voltage, AC back-emf and the load inductance voltage. This figure confirms the theoretical expectations, showing how the switching frequency varies in accordance with the average inverter output voltage variations. Note also how the local minima in Figure 3.4 (a) occurs exactly at the time where the load average voltage is at its

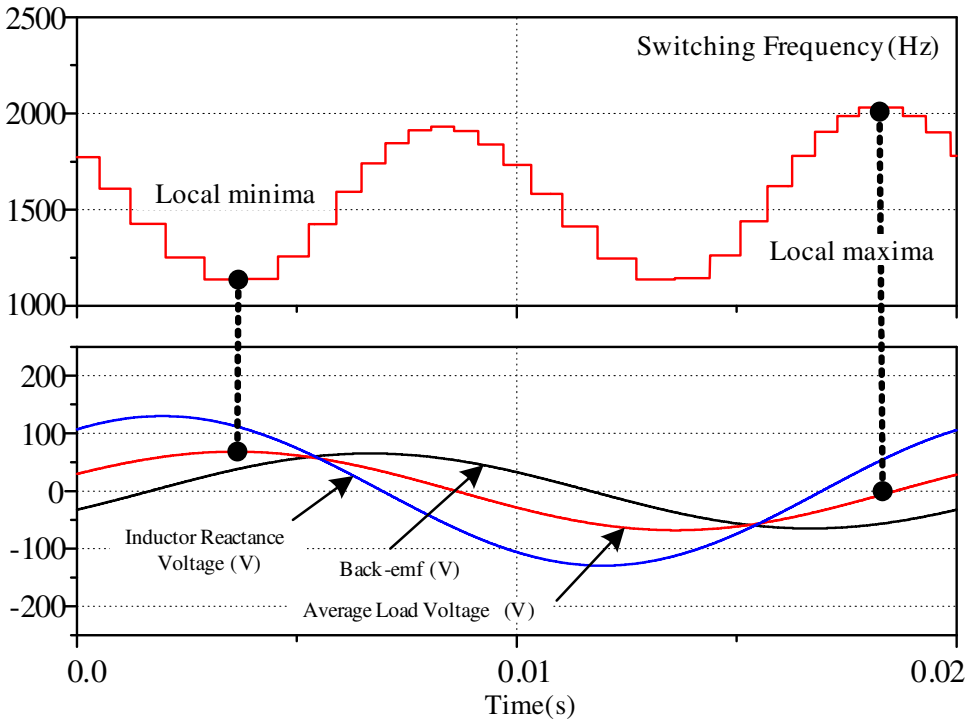


Figure 3.4: (Top) Variation in the switching frequency for a two-level VSI (Bottom) average load voltage, ac load back-emf and the inductor reactance voltage

maximum magnitude, while the local maxima in Figure 3.4 occurs exactly at the time where the load average voltage is at zero. This is also confirmed by setting $V_{avg} = 0$ in equation (3.11), to get a maximum switching frequency of:

$$\max(f_{sw}) = \frac{V_{DC}}{4LI_h} \tag{3.12}$$

3.2 Two-level Constant Frequency Hysteresis Current Regulation Using Average Voltage

In the previous section, a detailed mathematical analysis was presented to identify the primary cause of variable switching frequency for hysteresis regulation of a single-phase VSI. This section uses this understanding to develop a new approach for hysteresis current regulation of a two-level VSI to maintain a constant switching frequency.

3.2.1 Two-Level Hysteresis Band Variation Using Average Inverter Voltage

Section 3.1 has shown how the variable switching frequency is caused by the variation in the average load voltage. Equation (3.11) defines this switching frequency in terms of the average inverter output voltage and the constant system parameters including the load/filter inductance and the bus voltage. From this equation, it can be identified how the hysteresis band I_h can be varied in response to the average inverter output voltage in order to achieve a constant switching frequency. Rearranging equation (3.11) and solving it for a time-varying hysteresis band gives the following equation:

$$I_h(t) = \frac{\left(V_{DC}^2 - V_{avg}^2(t)\right)}{4Lf_{sw}} \tag{3.13}$$

Equation (3.13) can be rearranged to:

$$I_h(t) = I_{h,\max}\left(1 - \left(V_{avg}(t)/V_{DC}\right)^2\right) \tag{3.14}$$

where $I_{h,\max} = \frac{V_{DC}}{4Lf_{sw}}$ is the maximum allowable hysteresis band. (3.15)

Equation (3.14) defines how the hysteresis band should be varied to maintain a constant switching frequency as a function of the normalised average inverter output voltage. The value $I_{h,\max}$ can be set to achieve the required current ripple magnitude and the desired inverter switching frequency for a particular system. From equation (3.15), to achieve a higher switching frequency under fixed system parameters; the

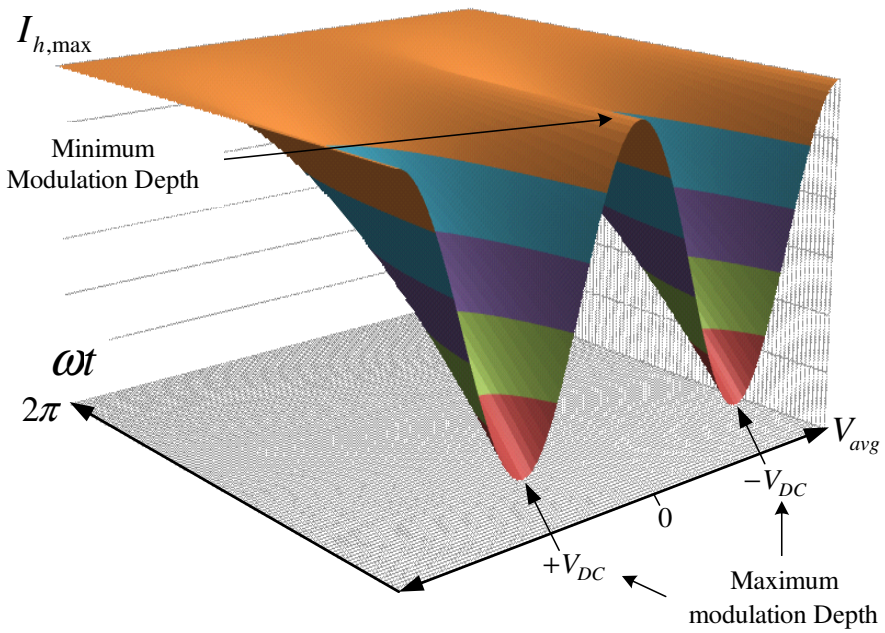


Figure 3.5: Three-dimensional diagram of variable hysteresis band with respect to the variation in average inverter output voltage and time

maximum allowable hysteresis band must be made smaller and for a lower switching frequency, the maximum allowable hysteresis band must be made larger.

Figure 3.5 shows the relationships between the variable hysteresis band, the (normalised) average inverter output voltage and the target fundamental reference period. From this figure, when the average voltage is near zero, the hysteresis band remains constant, much like conventional fixed band hysteresis regulation. When the average inverter output voltage magnitude increases, the hysteresis band changes significantly with maximum variation at the full modulation depth.

This approach for variable band hysteresis current regulation has the following advantages:

- The operation of the variable hysteresis band is independent of the output load/filter parameters such as load resistance and inductance.
- The variable hysteresis band calculation is independent of the current error. Hence, the current error derivative or the time information of the current error zero-crossing is not required to vary the hysteresis band.
- The variable hysteresis band directly incorporates the effect of the (normalised) average inverter output voltage. This means there is no need to further predict the load back-emf provided that the average voltage is

known (determination of the normalised average inverter output voltage will be discussed in the next section).

- The inverter switching frequency remains constant throughout the reference target fundamental cycle.

Figure 3.6 shows a simplified block diagram of the new variable band hysteresis current regulation approach applied to a two-level VSI feeding a back-emf type load through a series RL impedance. Figure 3.7 compares the performance of the new approach using the average inverter output voltage with that of a conventional fixed-band hysteresis current regulator, using the controller arrangements shown in Figure 3.1(b) and Figure 3.6(b). To conduct this comparison, a sinusoidal waveform is used as a current reference, which is synchronised with the phase average output voltage. As expected, using a fixed hysteresis band imposes a variation of 40% on the desired average switching frequency of 2500 Hz while employing a variable hysteresis band that varies according to the average output voltage significantly reduces the variation to less than 1% of the target switching frequency.

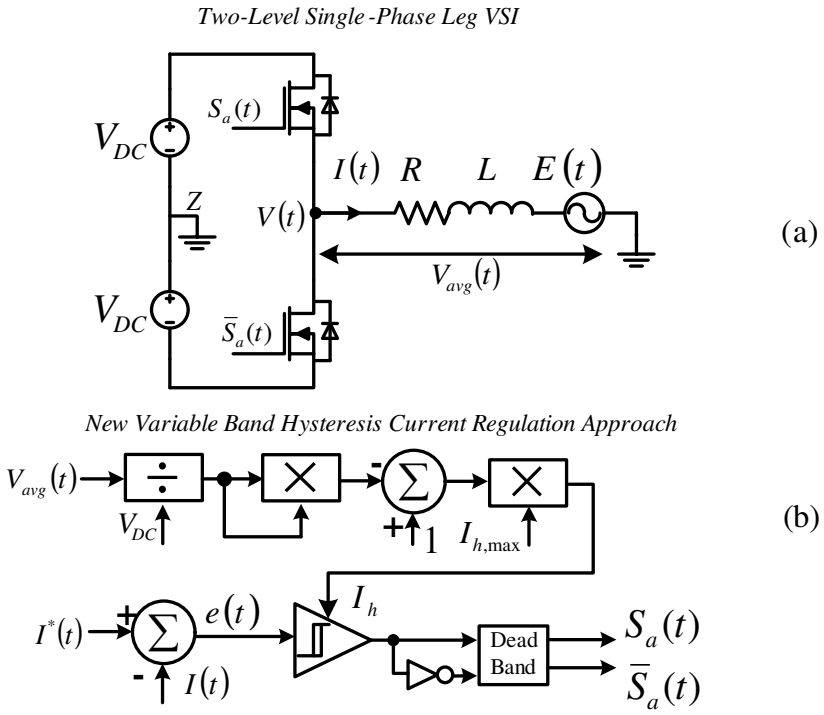


Figure 3.6: Simplified block diagram of the new variable band hysteresis controller for a two-level single-phase leg inverter feeding a back-emf type load with series RL load

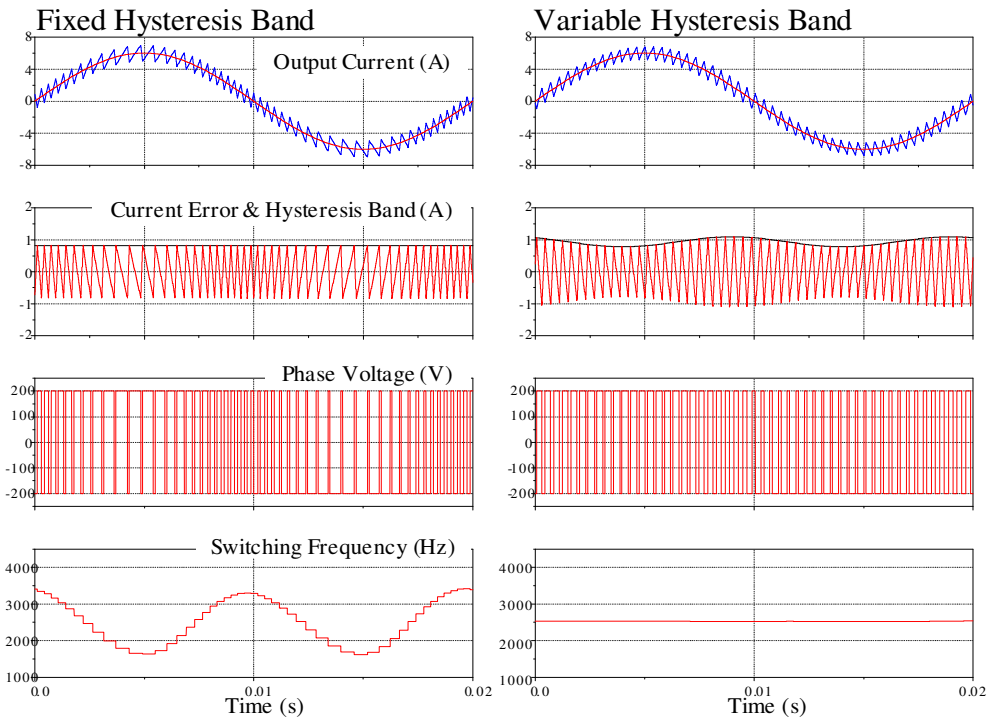


Figure 3.7: Comparison of between the performance of a conventional fixed hysteresis band and proposed variable hysteresis band - target switching frequency $f_{sw} = 2500\,\text{Hz}$, modulation depth = 0.9

3.2.2 Measuring the Normalised Average Inverter Output Voltage

In the previous section, the principles of the new variable hysteresis band current regulation approach have been discussed. Equation (3.14) defines the variable hysteresis band that uses the knowledge of the normalised average inverter output voltage to maintain constant switching frequency. This section now presents a novel strategy to calculate this average inverter output voltage on a switching cycle-by-cycle basis.

Figure 3.8 shows the switching current ripple, together with the switching signal, switched phase voltage and the average output voltage. From this figure, it can be seen that when the output load current reaches the lower hysteresis band, the switching signal switches to the HIGH state at the rise time of T_1. When the output load current hits the upper hysteresis band, the switching signal switches to the LOW state at the fall time of T_2. The switching process continues over one full switching period T until the current reaches the lower boundary again at the rise time of T_3.

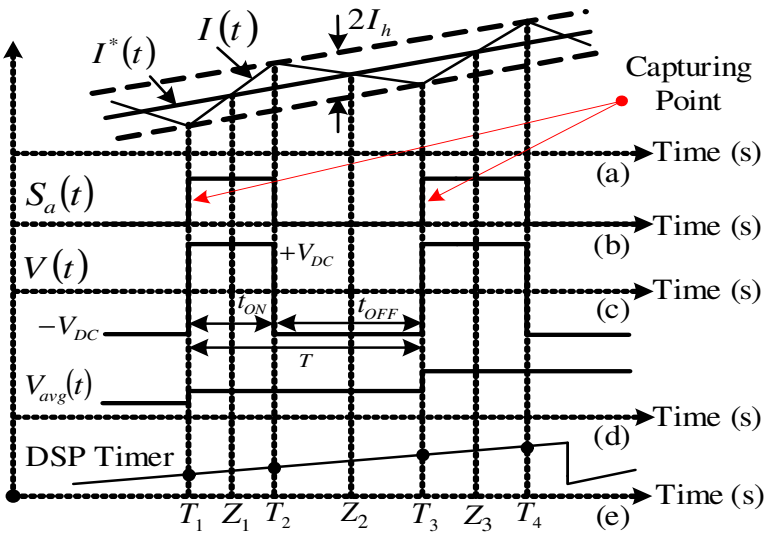


Figure 3.8: Switching behaviour of the hysteresis current regulation process in terms of discrete time events (a) current error (b) gate signals (c) switched phase voltage (d) average inverter output voltage (e) DSP timer

Hence, the switching behaviour of the phase leg can be expressed in terms of the discrete time events T_1 , T_2 and T_3 as follows:

$t_{on} = T_2 - T_1$ is the ON-time of the switching period. (3.16)

$t_{off} = T_3 - T_2$ is the OFF-time of the switching period. (3.17)

$T = T_3 - T_1$ is the switching period. (3.18)

From equations (3.16)-(3.18), the normalised average inverter output voltage (i.e. duty cycle) over one switching period can be expressed as:

$$V_{avg}/V_{DC} = 2\left(\frac{t_{on}}{t_{on}+t_{off}} - 0.5\right)$$
$$\rightarrow V_{avg}/V_{DC} = 2\left(\frac{T_2 - T_1}{T_3 - T_1} - 0.5\right) \quad (3.19)$$

The switching times can be readily measured using the capture/timer input ports on the DSP controller, fed (for simplicity) from the hysteresis comparator outputs. These ports use a continuously cycling internal timer (shown in Figure 3.8 (e)) to record the time instance of a digital input transition, so that the input port can then

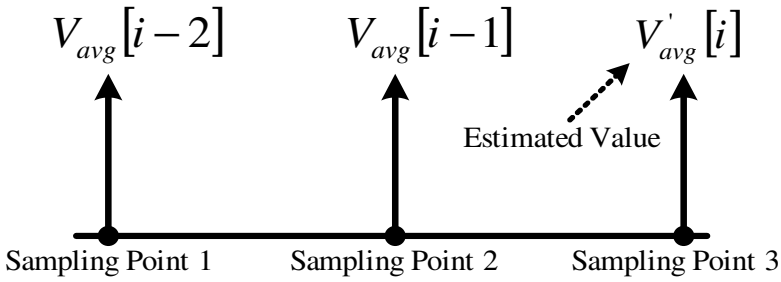


Figure 3.9: Calculation of average inverter output voltage from the previous over sampling points.

record the instance of each (ideal) rising and falling phase leg switching transition. Measuring the average voltage in this way has the following advantages:

1. The approach can accurately estimate the per cycle average inverter output voltage from the switched output of a PWM controlled inverter on a switching cycle-by-cycle basis.
2. The errors caused by bandwidth limits of a conventional analog or digital low pass filter are avoided.
3. Additional voltage sensors are not required to measure the phase leg switched voltage since the normalised average inverter output voltage can be directly determined from the gate switching signals and does not require direct knowledge of the DC bus voltage V_{DC} .
4. Variations in V_{DC} can be accommodated by recalculating $I_{h,\max}$ using (3.19). Note that this calculation can be readily managed at the switching interrupt frequency (or even slower) because V_{DC} does not vary quickly.

However, the calculation of the inverter average voltage using (3.19) does introduce a one switching cycle sampling delay. The next section presents a linear extrapolation method that can be used to compensate for this delay.

3.2.3 Compensating for the Effect of Sampling Delay

In order to calculate the average inverter voltage, the controller must wait for one full switching period to get time information regarding the required switching time events. However measuring the average voltage in this way creates a one switching cycle sampling delay to calculate the variable hysteresis band, using equation (3.14). This delay can be compensated using simple linear extrapolation from the previous

sampled values as shown in Figure 3.9. Ref [59] proposes three linear extrapolation alternatives as follows:

$$V_{avg}[i] = V_{avg}[i-1] \qquad \textbf{\textit{(method 1)}} \tag{3.20}$$

$$V_{avg}[i] = 2 * V_{avg}[i-1] - V_{avg}[i-2] \qquad \textbf{\textit{(method 2)}} \tag{3.21}$$

$$V_{avg}[i] = 3 * V_{avg}[i-1] - 2 * V_{avg}[i-2] \qquad \textbf{\textit{(method 3)}} \tag{3.22}$$

A simulation study was undertaken to select between these alternatives, with the results shown in Figure 3.10. The figure shows the actual load average voltage, together with the extracted average voltage from (3.19), and the voltage estimated by three linear extrapolation methods. From this figure, using method 2 in equation (3.21) gives the best approximation of the true average inverter voltage.

Figure 3.11 then examines the effect of linear extrapolation on the variation of the switching frequency. From this figure, it can be seen that the variable band generated by a feedforward compensated average voltage using equation (3.21) further reduces the variation in the switching frequency in comparison to the variable band generated from the direct extraction of the average load voltage. Based on these comparisons, it can be seen that the effect of sampling delay is not avoidable, however, it can be minimised by choosing the appropriate method of linear extrapolation as a means of feedforward compensation.

Figure 3.12 provides experimental confirmation of the average voltage measurement process, where the average inverter output voltage has been displayed using a spare DAC channel driven by the calculated value from the DSP (details of experimental systems are given in chapter 7).

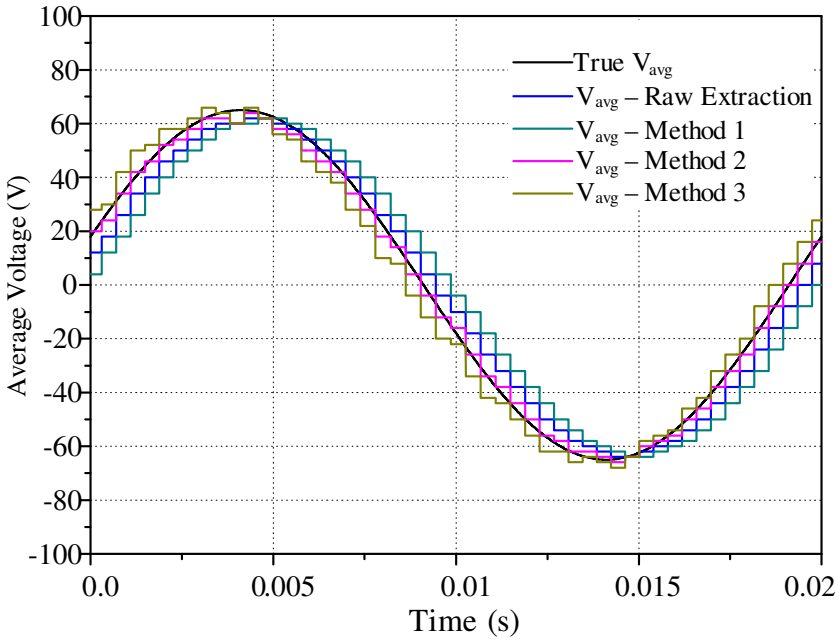


Figure 3.10: Comparison between calculation of the average inverter output voltage showing the true average load voltage, true extraction from the gate switching signals and using the method 2 linear extrapolation. Modulation Depth =0.8

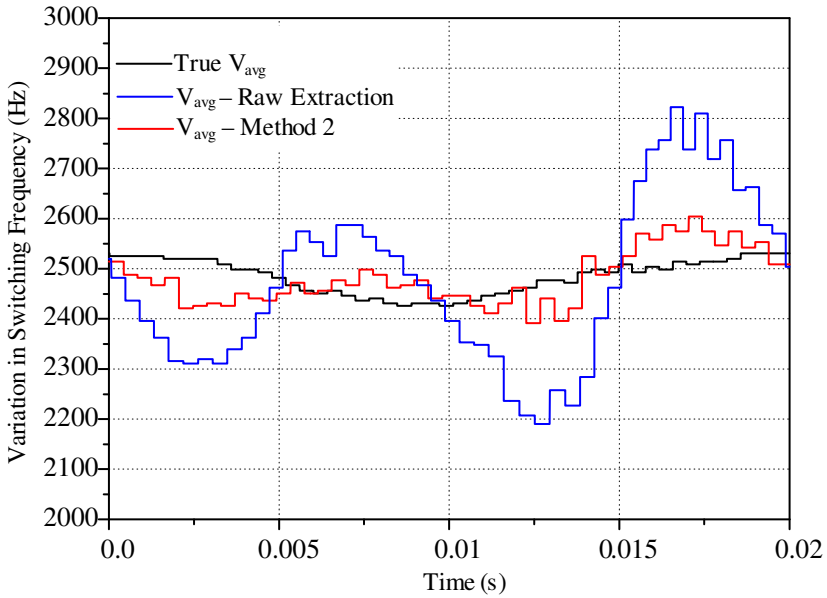


Figure 3.11: Effects of sampling on the variation of the switching frequency using the true average load voltage, true extraction from the gate switching signals and using the method 2 linear extrapolation. Modulation Depth =0.8

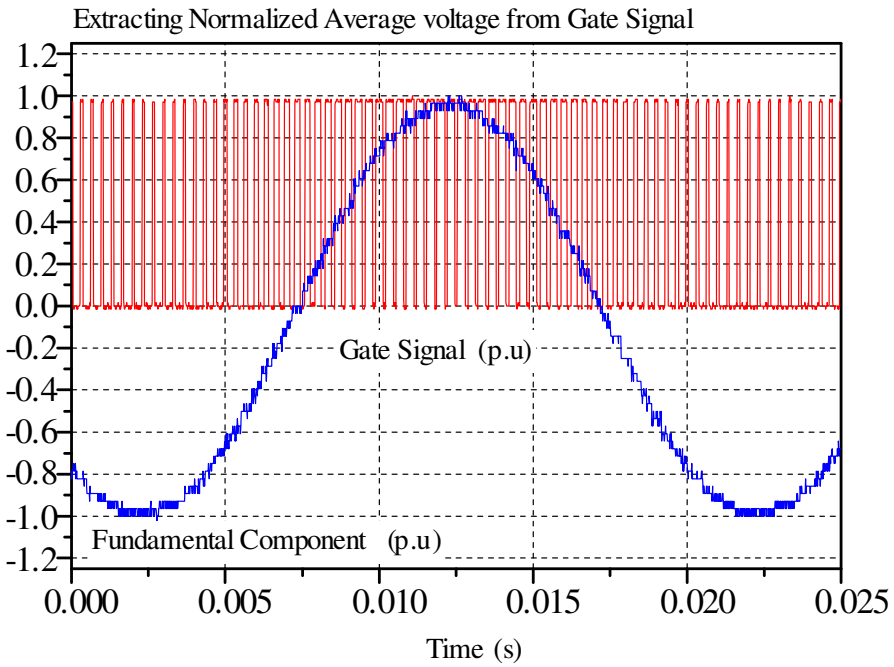


Figure 3.12: Experimentally measured average inverter output voltage (DAC output) compared to the PWM gate signal.

3.3 Synchronisation to a Fixed Reference Clock

The next step in development of the new approach is to fine-tune the calculated variable hysteresis band to synchronise the zero-crossings of the current error to a fixed reference clock. This makes the harmonic performance of the new hysteresis approach very close to that of open-loop asymmetrical sampled PWM with a triangle carrier, which is known to achieve optimum harmonic performance [11][12]. The synchronisation provides the following advantages:

- Further reduction in the variation of the switching frequency
- Achieving the optimum harmonic performance
- Minimising the output current ripple

The synchronisation of the current error zero-crossing is implemented as an additional feedforward step to further optimize the operation of the new hysteresis approach. This ensures a more robust and reliable synchronization process in comparison to the previously reported dead-beat adaptive hysteresis current regulation approaches [81][84] where the variable band calculation is dependent on direct measurement of the current error zero-crossings. Finally, synchronising in this way ensures less deviation in the switching period at higher modulation depths.

3.3.1 Without Dead-Time Compensation

In this section, the synchronisation of current error zero-crossing is first considered without compensating for the effect of dead-time. The next section then extends this analysis to compensate for this effect.

Essentially, since the phase current error is primarily the current switching ripple component, it always crosses zero at the instant where the switched output volt-seconds match the average output commanded volt-seconds. For fixed frequency PWM, this occurs at the peak and trough of the modulating triangular carrier waveform, which therefore makes these instants a convenient synchronizing point for a reference clock. Similarly, the synchronization process presented here compares the zero-crossing transition times of the current error against a fixed reference clock, and uses the time error of this transition to fine-tune the variable hysteresis band.

Figure 3.13 shows this concept, where the time difference between the actual zero-crossing of the current error and the expected zero crossing time from the target clock is given by:

$$\delta t = t_{error,1} - t_{ref,1} \tag{3.23}$$

The time $t_{error,1}$ is the actual current error zero-crossing while $t_{ref,1}$ is the time occurrence of the digital reference clock generated by the DSP. If this timing error is not corrected within the next half switching period, it will continue to become $t_{error,2} - t_{ref,2}$ at the next clock reference point in the next half switching period, unless a correction of δI_h is made to the variable hysteresis band to make the inverter switch early, as shown in Figure 3.13. Using similar triangles, the following relationship can be established from this figure:

$$\frac{I_h}{T/2} = \frac{I_h - \delta I_h}{T/2 + \delta t} \tag{3.24}$$

Rearranging equation (3.24) and solving it for δI_h gives:

$$\begin{aligned} \delta I_h &= -\frac{I_h * \delta t}{T/2} \\ &= -I_h * 2 f_{sw} * \delta t \end{aligned} \tag{3.25}$$

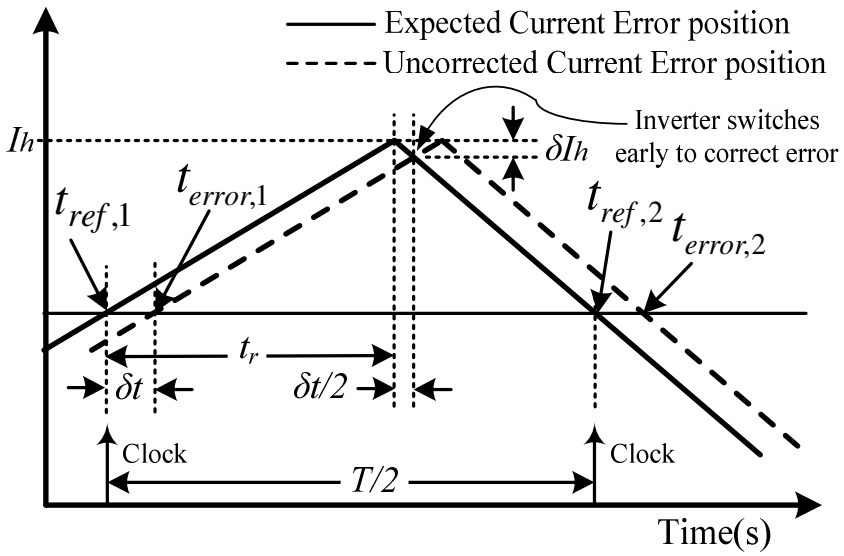


Figure 3.13: Identification of current error zero crossing timing errors.

where f_{sw} is the target switching frequency. Combining this with equation (3.14) gives a final variable hysteresis band of:

$$I_{h,new} = \delta I_h + I_{h,old} \tag{3.26}$$

However it is not necessary to explicitly measure the current error zero crossing times, since they occur essentially midway between the rising and falling edges of the inverter switched output, as shown in Figure 3.8. Hence the zero crossing times are adequately given by:

$$Z_1 \approx (T_1 + T_2)/2 \tag{3.27}$$

$$Z_2 \approx (T_2 + T_3)/2 \tag{3.28}$$

Since these switching times $T_1, T_2, T_3 \ldots$ have already been recorded by the DSP controller using the timer/capture port for the average inverter voltage calculation, no additional zero-crossing detection circuitry is required. Additionally, calculating the zero crossing instants in this way avoids errors caused by second order effects such as:

- Measurement noise
- Double current error zero-crossings within one half switching cycle due to reference current step changes at the sampling instant.
- Loss of the zero-crossings due to the current error free-wheeling behaviour in the overmodulation region.

3.3.2 With Dead-Time Compensation

As discussed in chapter 2, all hard switched voltage source inverters require a dead-time delay to be included between the turn-off of the outgoing switch of a phase leg and the turn-on of the corresponding incoming switch. When the output current is conducting through the anti-parallel diode of a phase leg switch, the effect of dead-time is to delay the actual phase leg voltage transition compared to the ideal command, until the incoming switch actually turns on. This delays either the upper or the lower current slope transitions shown in Figure 3.13 by the dead-time period, which consequently increases the zero crossing time error δt as calculated by

equation (3.23) by half this period. Hence the effect of dead-time can be readily compensated by simply adding a delay of (dead-time)/2 to δt before calculating the required hysteresis band adjustment (3.25) that maintains clock synchronism. The final hysteresis band offset then becomes:

$$\delta I_h = -I_h * 2 f_{sw} * (\delta t + t_{db}) \tag{3.29}$$

Figure 3.14 shows the precision of the current error zero crossing positioning achieved with this new synchronization strategy. From this figure, the current error zero crossings line up almost exactly with the 5 kHz target reference clock. Figure 3.15 shows the substantial improvement in switching frequency regulation that has been experimentally achieved with the fixed clock synchronization and dead-time compensation strategies. In Figure 3.15(a), where both the synchronisation and dead-time compensation are disabled, the switching frequency varies by 10% of the target switching frequency of 2.5 kHz. Using the clock synchronisation process without dead-time compensation reduces this variation to 5% of the target switching frequency. Enabling both the synchronisation and dead-time compensation strategies, further reduces this variation to less than 2% of the target switching frequency. Additionally, this figure confirms the effectiveness of the strategy for the compensation of switching dead-time even at substantial values of 5us and 12us where the switching frequency is still maintained within 2% of the target switching frequency.

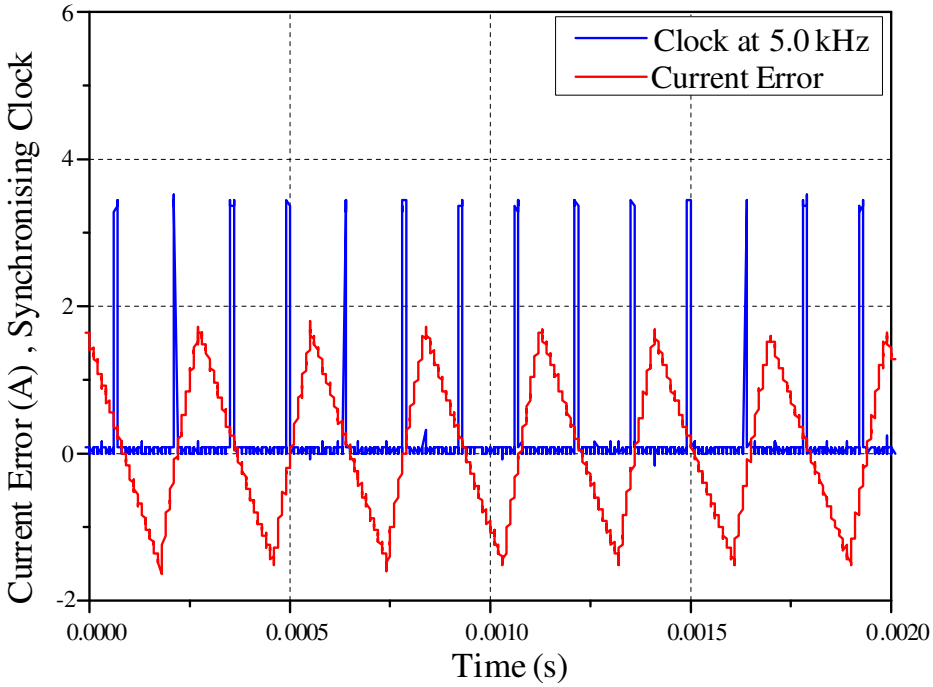


Figure 3.14: Experimental results for synchronization of the current error zero crossings to a fixed reference clock.

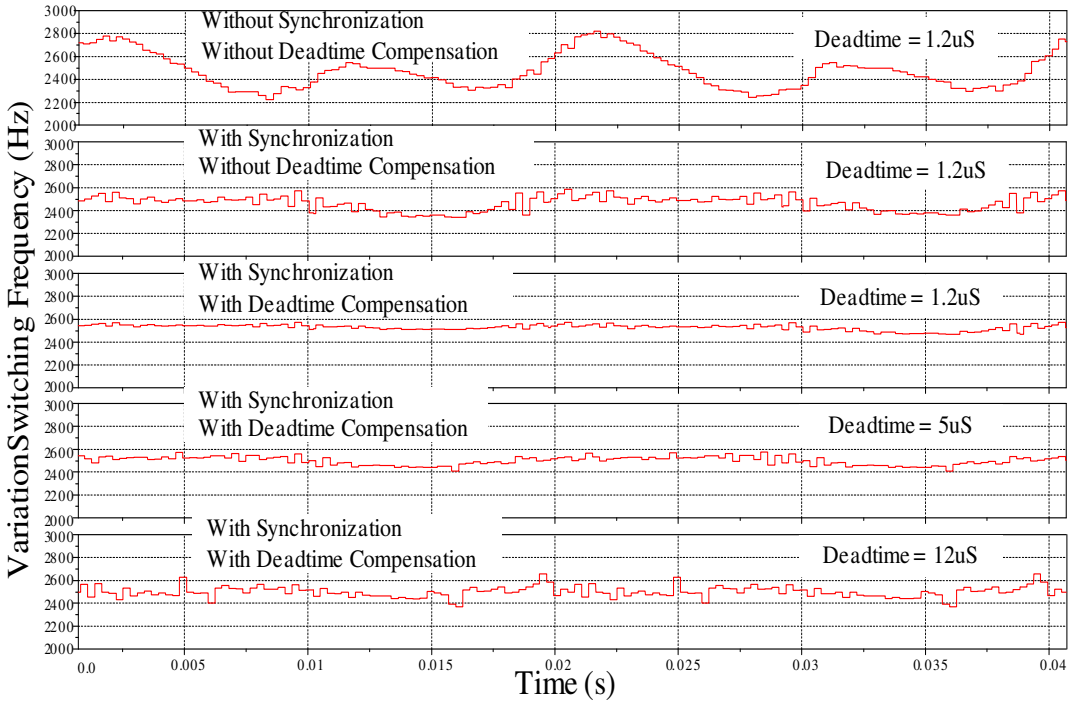


Figure 3.15: Improvement in switching frequency control with current error zero crossing synchronisation and dead-time compensation.

3.3.3 Synchronization under Various Pulse Ratios

Figure 3.16 shows the results of further investigations that have examined the performance of the synchronization process at various modulation depths and at different pulse ratios. From this figure, it can be seen how the maximum switching period variation over each fundamental cycle increases as the modulation depth increases, and how this increase is larger for smaller inverter pulse ratios. This result is entirely expected, since a higher modulation depth means a larger load back-emf voltage, and a smaller pulse ratio means more back-emf magnitude variation during each switching period. However, even at the very low pulse ratio of 9, the maximum switching period deviation is still less than 15% of the target switching frequency, even at maximum modulation depth.

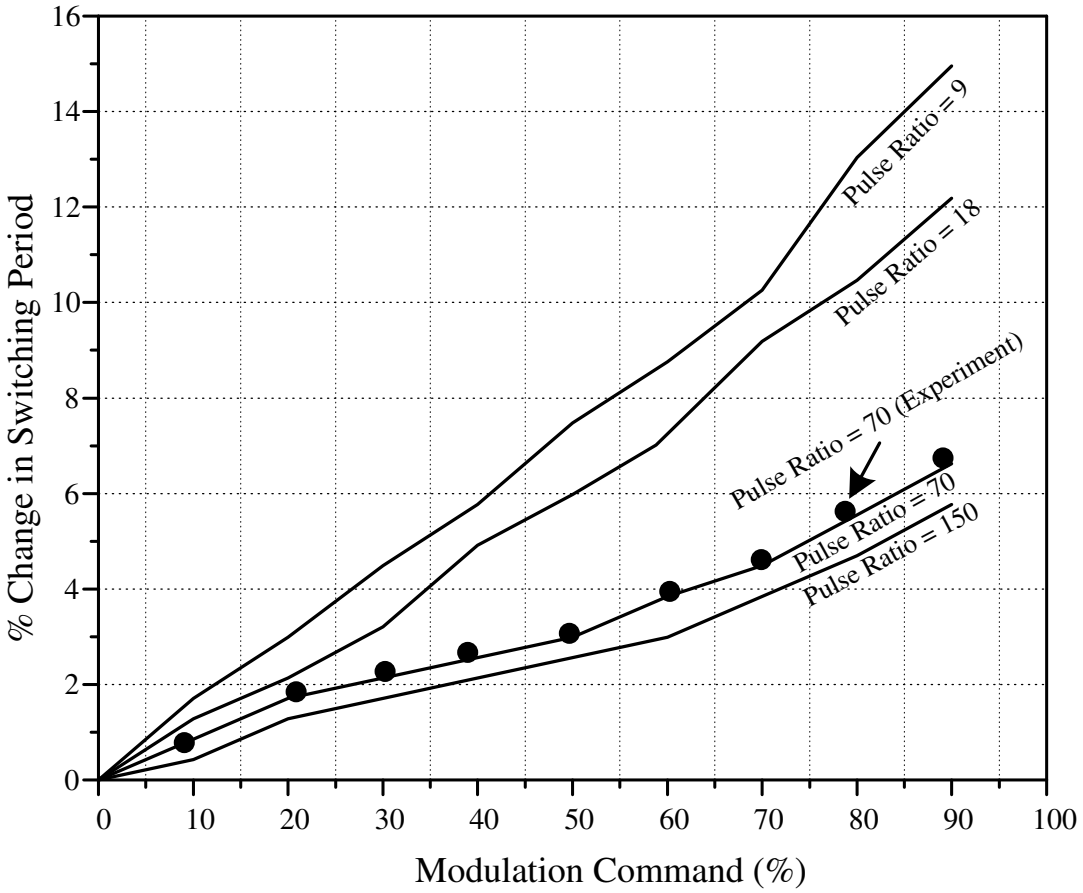


Figure 3.16: Experimental results for synchronization of the current error zero crossings to a fixed reference clock. (Switching Frequency = 2500 Hz)

3.4 Managing the Overmodulation Region

Over modulation occurs when the required average inverter output voltage exceeds the available DC link voltage ($V_{avg} > 1.0 V_{DC}$) for sine-triangle PWM or $V_{avg} > 1.15 V_{DC}$ for three phase CSVM). Overmodulation both distorts the linearity of the modulator operation and generates significantly increased low frequency baseband distortion [11][16].

For variable band hysteresis regulators there is a substantial difficulty with overmodulation, since from equation (3.14) as the peak average voltage approaches V_{DC}, the variable hysteresis band will approach zero, viz:

$$I_h(t) = I_{h,\max}\left(1 - \left(V_{avg}(t)/V_{DC}\right)^2\right) \tag{3.30}$$

If $V_{avg}(t) = V_{DC}$ then $I_h(t) = 0$

Figure 3.17 shows the consequence of this constraint, with high frequency switching occurring at the peak of the average inverter output voltage as the variable hysteresis band limit approaches zero and the controller loses stability.

The new hysteresis current regulation approach solves the constraints of operating in the overmodulation region associated with HCCs by clamping the variable hysteresis band at a minimum value as the controller enters the overmodulation region. Calculation of the variable hysteresis band from the average inverter output voltage allows the DSP to recognize an attempt to overmodulate the inverter because the cyclic average phase leg voltage starts to exceed the available bus voltage. This detection of the region of overmodulation is implemented in two steps.

Firstly, the DSP identifies when the measured average inverter voltage is close to its maximum value. For this investigation, this value is set heuristically to a maximum value of about 95% of the available DC bus voltage. To further improve the robustness of this detection, an auxiliary timer counter is also arranged within the DSP to detect when the next switching edge is expected to occur based on equations (3.16)-(3.18). This detection identifies if the phase leg fails to switch, providing further confirmation of an overmodulation event occurring.

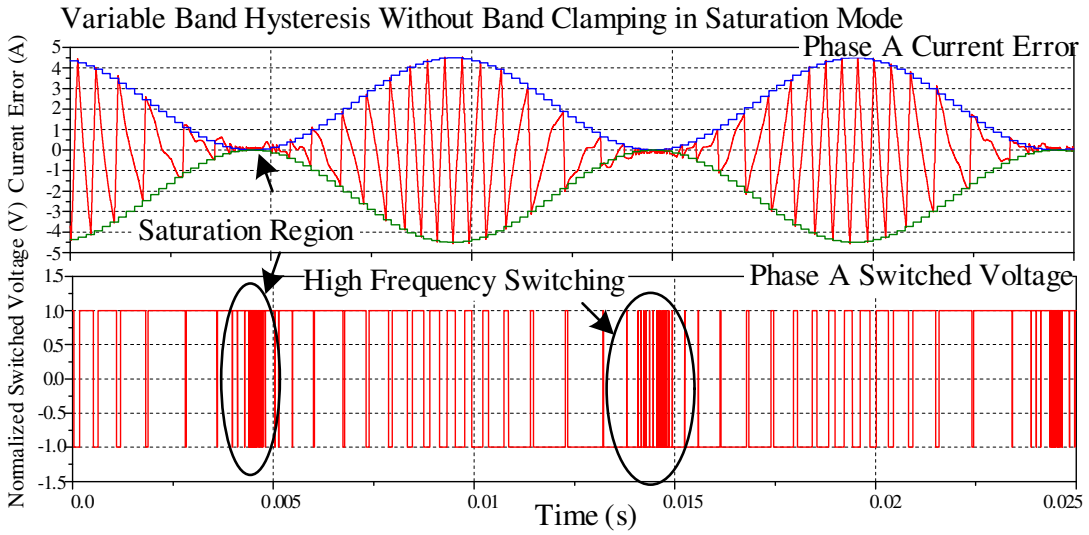


Figure 3.17: Simulated performance of variable band hysteresis current regulation without hysteresis band clamping, peak average inverter output voltage = 1.0*V_{DC}

Secondly, once anticipated overmodulation is detected, the reducing hysteresis band is kept above a defined minimum value. This response is managed by the DSP, which ensures the variable hysteresis limit $I_h(t)$ is kept above a minimum value of about 20% of $I_{h,\max}$. (This limit was determined heuristically by the investigations of this thesis, and has been found to achieve a gradual response into overmodulation without spurious high frequency switching events, despite the unavoidable slightly lagging response of the DSP because of sampling delays.)

VSI switching automatically recommences after overmodulation as phase current error returns within the hysteresis bands and hits either the upper or lower variable hysteresis bands. Once switching recommences as the overmodulation situation reverses, the DSP smoothly ceases variable band limiting.

Figure 3.18 illustrates this band limit clamping technique, showing how the regulator smoothly progress into overmodulation without any sign of high frequency switching. From this figure, it can also be seen how the regulator still tracks the reference current extremely well even in the nonlinear overmodulation region (modulation depth of 120%), with no sign of transient disturbances, as the inverter enters and leaves this region of operation.

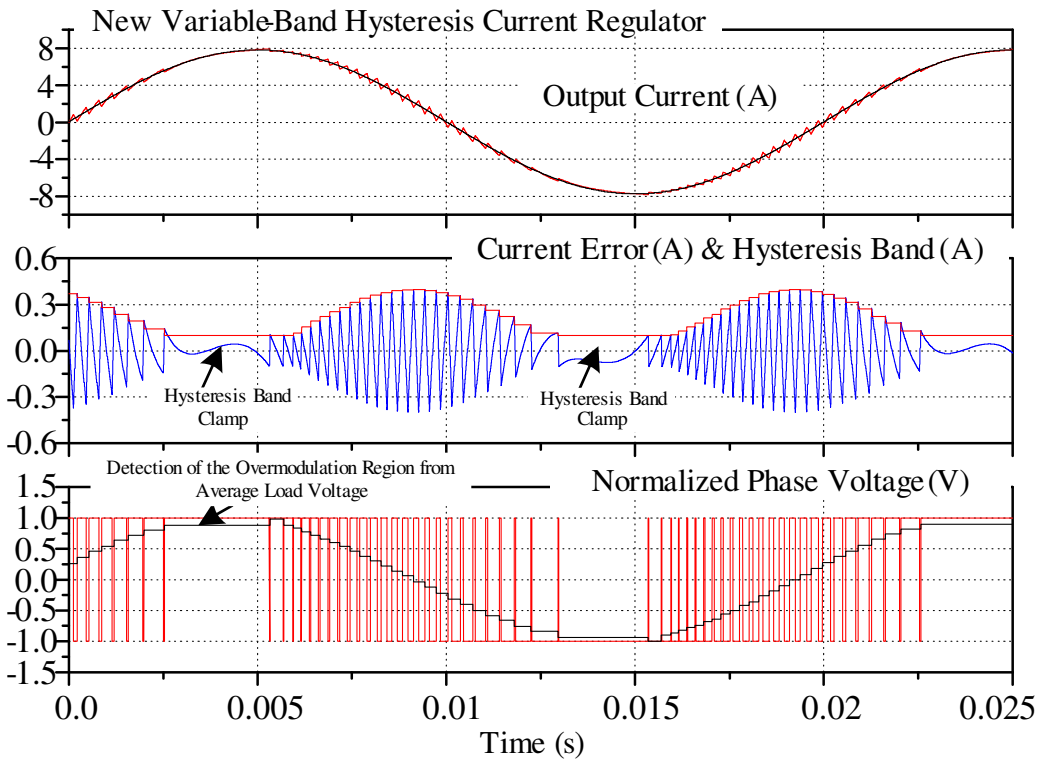


Figure 3.18: Simulated performance of variable band hysteresis current regulation with hysteresis band clamping, peak average inverter output voltage = $1.2*V_{DC}$

Experimental results to confirm the effectiveness of the band clamping strategy for a two-level inverter are provided in chapter 4.

3.5 Consolidated Simulation and Experimental Results[2]

Throughout this chapter, various simulation and experimental results have been presented in order to support the development of the theoretical analysis. This section presents the consolidated simulation and experimental results for the operation of the proposed hysteresis current regulation approach for the two-level single-phase VSI with the circuit parameters listed in Table 3.1. To conduct the experimental investigations in this thesis, a 10 μf capacitor was connected in parallel with the main load resistor to eliminate switching ripple from the load voltage. This load voltage then acts like the motor back-emf or a synchronized grid voltage with the effective load resistance becoming essentially only the (small) inductor series resistance.

Figure 3.19 (simulation) and Figure 3.20 (experiment) show the operation of the new variable band hysteresis controller. This includes the output load current, current error, hysteresis band, phase leg switched voltage for both fixed-band and the new variable hysteresis band current regulator. The variation in hysteresis band to maintain a constant switching frequency can be readily seen. The fixed-band hysteresis controller performance is also shown which acts as a baseline for comparison against the performance of the new hysteresis current regulation approach.

Table 3.1: Circuit parameters for the simulation and experimental implementations of the new hysteresis current regulation approach applied to the two-level single-phase VSI

Circuit Parameter		**Value**
Resistive load	(R) (Ω)	0.2
Inductive load	(L) (mH)	18
Target Switching frequency	(f_{sw}) (kHz)	2.5
Total DC bus voltage	($2V_{DC}$) (V)	100
Reference current	(I_{ref}) (A)	5
Back EMF frequency	(Hz)	50

[2] The operation of the new hysteresis approach for a two-level single-phase leg VSI has been verified using a H-bridge modulated as a two-level VSI to avoid the complexity associated with the connection of the output load to a virtual DC bus mid-point.

Figure 3.21 (simulation) and Figure 3.22 (experiment) show the harmonic spectra of the switched output voltage at an operating modulation depth of 0.9. The figures show how the use of the new variable hysteresis band significantly improves the output voltage harmonic performance compared to a fixed-band HCC. In simulation, the WTHD is reduced from 3.96% for a fixed-band HCC to 1.86% for the variable band HCC. The same results are confirmed in experiment where the WTHD is reduced from 4.63% for a fixed-band HCC to 2.13% for the variable band HCC. The variable band harmonic performance is also essentially equivalent to that of open-loop asymmetrical regular sampled PWM with a large central first carrier harmonic and well defined sidebands harmonics that rapidly decrease in magnitude. This level of harmonic performance has significant advantages in terms of harmonic cancellation between phase legs when applied to other VSI topologies, as discussed later in this thesis.

Figure 3.23 shows the experimental transient performance of the new variable hysteresis band regulator to a 100% change in commanded current. From this figure, the excellent dynamic response expected from a hysteresis regulator has not been at all compromised by the new variable band implementation.

Simulation

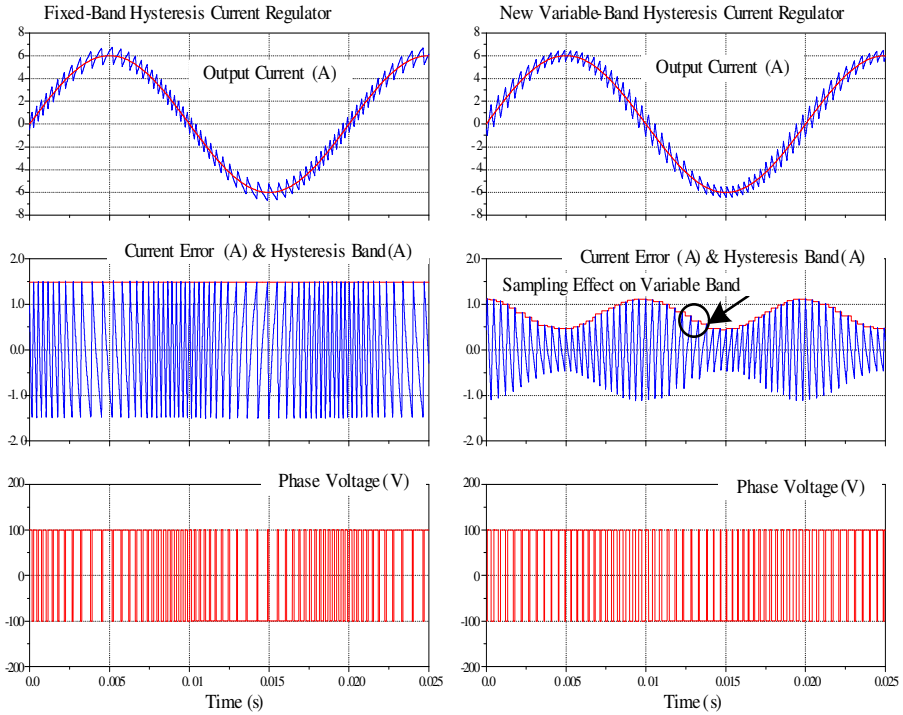


Figure 3.19 Performance of the fixed-band and new variable-band hysteresis current regulation for a two-level single-phase VSI (top) output current (middle) hysteresis band and current error (bottom) switched output voltage Modulation Depth = 0.9

Experiment

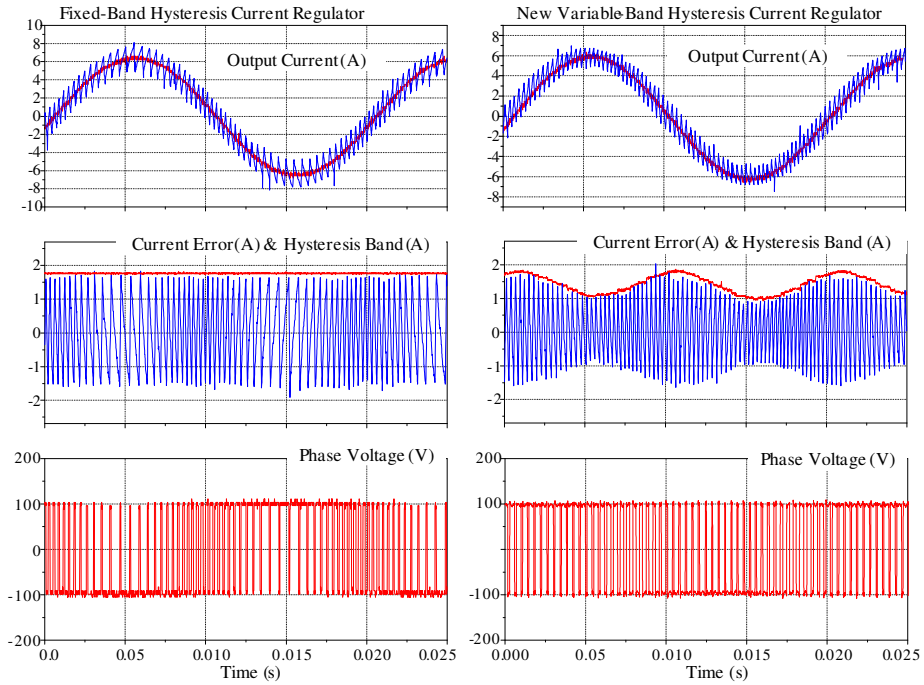


Figure 3.20 : Performance of the fixed-band and new variable-band hysteresis current regulation for a two-level single-phase VSI (top) output current (middle) hysteresis band and current error (bottom) switched output voltage Modulation Depth = 0.9

Simulation

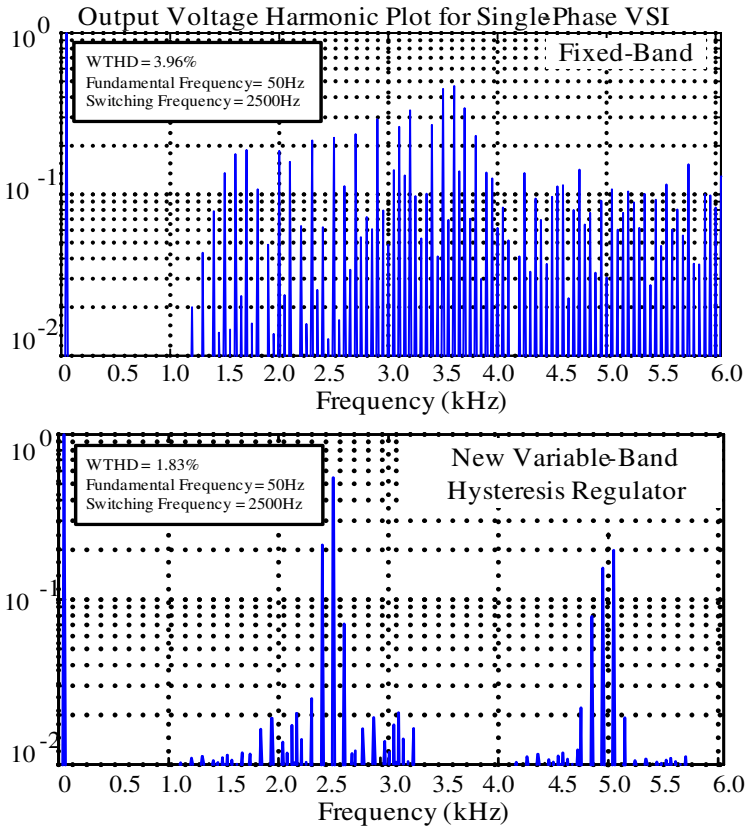


Figure 3.21 Output voltage harmonic spectra for (a) fixed-band (b) new variable band hysteresis current regulator, Modulation Depth = 0.9

Experiment

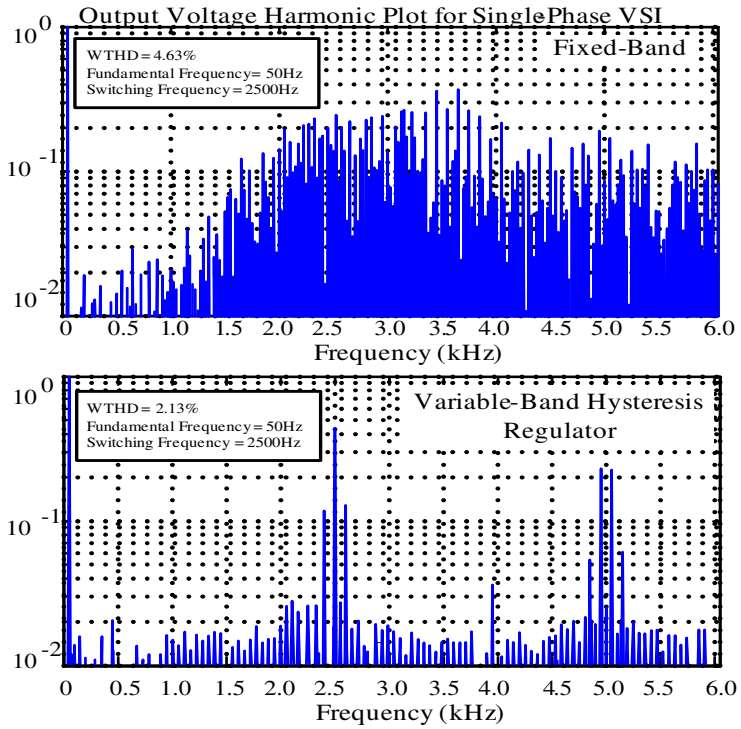


Figure 3.22: Output voltage harmonic spectra for (a) fixed-band (b) new variable band hysteresis current regulator, Modulation Depth = 0.9

Experiment

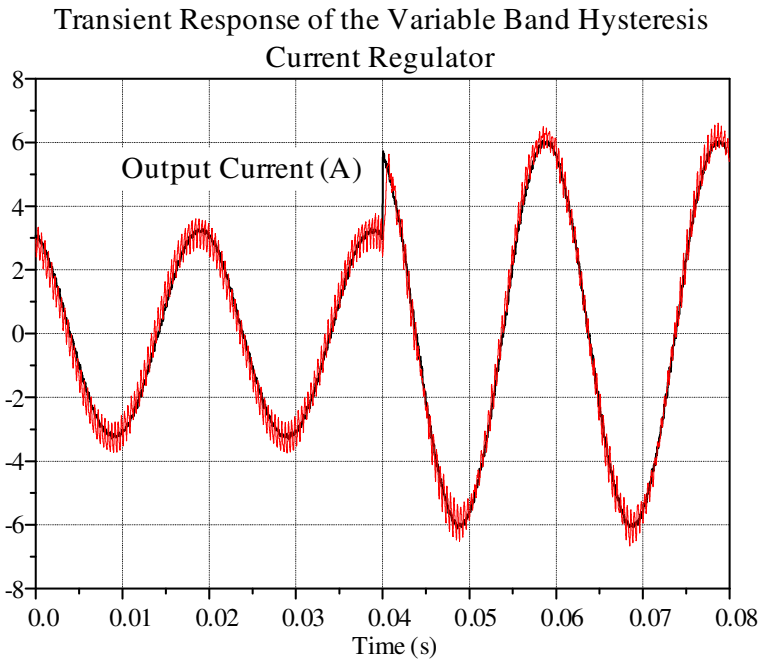


Figure 3.23 Experimental transient step response of the new variable band hysteresis approach– 100% commanded step change in current.

3.6 Summary

This chapter has presented a new approach for hysteresis current regulation of a two-level single-phase VSI to maintain constant switching frequency.

The development of the new hysteresis current regulator began by identification of the average inverter output voltage as the primary cause of variable switching frequency in conventional fixed-band HCC.

A novel technique was then presented to synchronously measure the average inverter output voltage from the switching transition times of the phase-leg switching signal. The measurement strategy does not require the current error derivative or its zero-crossing information to estimate the output load back-emf. Furthermore, it avoids the use of analog or digital filtering with their associated frequency roll-off effects. A linear extrapolation technique has also been applied to the measured average inverter output voltage to compensate for the effect of sampling delay.

The measured average inverter output voltage was then used to develop a new approach for variable band hysteresis current regulation of a two-level single-phase leg VSI to maintain constant switching frequency. In this way, the variable band directly incorporates the effect of the average voltage, and does not require further prediction of the output load back-emf.

A novel strategy is then proposed to synchronise the zero-crossing of the phase-leg current error to a fixed reference clock and to allow for dead-time effects. This is done by adding an extra band offset to the variable hysteresis band, calculated from the time difference of the actual current error zero-crossing and the time of occurrence by the generated DSP clock. Synchronisation in this way does not requires any additional circuitry, while still achieving a harmonic performance similar to open-loop asymmetrical sampled PWM.

Finally, the operation of the controller was extended into the overmodulation region by clamping the variable hysteresis band to a minimum value, to ensure controller switching stability and a smooth transition between the linear and non-linear regions of operation.

Simulation and experimental results for this system are included to confirm the operation of the new hysteresis current regulation approach when applied to a two-level single phase VSI.

Chapter 4

Hysteresis Current Regulation of a Two-Level Three-Phase Inverter

This chapter[1] presents a new hysteresis current regulator for a two-level three-phase VSI by applying the fundamental concepts of the single-phase HCC presented in chapter 3. The controller synchronously measures the three-phase average inverter output voltages and varies the hysteresis bands for each phase leg to maintain a constant switching frequency. The major research stages presented here are:

- Section 4.2.2 presents a technique to compensate the common mode current using the switching information of the three-phase gate signals.
- Section 4.2.3 synchronises the phase A and phase B current error zero-crossings to a fixed reference clock. The resultant strategy achieves a harmonic performance that is very close to open-loop CSVM.
- Section 4.2.4 presents a technique to modulate the phase leg C using an open-loop sine-triangle PWM by replacing the third phase HCC with a fixed frequency directly modulated phase leg. This reduces the implementation complexity of the three-phase HCC while improving the line-to-line harmonic performance.
- Section 4.3 extends the modulation depth by injecting a third harmonic component to fully utilize the available DC bus voltage. The third harmonic voltage is calculated using the information of the three-phase average inverter output voltages.

Simulation and experimental results are presented at the end of this chapter.

---

[1] Materials in this chapter were first published as

1. Holmes, D.G.; Davoodnezhad, R.; McGrath, B.P.; "An improved three phase variable band hysteresis current regulator," *ECCE Asia (ICPE & ECCE), 2011 IEEE 8th International Conference*.
2. Holmes, D.G.; Davoodnezhad, R.; McGrath, B.P.; "An Improved Three-Phase Variable-Band Hysteresis Current Regulator," *Power Electronics, IEEE Transactions on*, vol.28, no.1, pp.441-450, Jan. 2013.
3. Davoodnezhad, R.; Holmes, D.G.; McGrath, B. P." Constant frequency three-phase hysteresis current regulator with extended modulation depth*," Power Electronics and Motion Control Conference (IPEMC), 2012 7th International.*

4.1 Review of Conventional Three-Phase Fixed Band Hysteresis Current Regulation

Figure 4.1(a) shows the topology of a two-level three-phase voltage source inverter. The inverter feeds into an isolated three-phase resistive and inductive load with AC back-emf. Note that the load neutral point (U_0) is not connected to the midpoint of the DC supply voltage (Z), to represent most three-phase applications such as grid-connected or motor drive inverters.

Figure 4.1(b) shows the conventional structure of a three-phase HCC for this voltage source inverter. In this configuration, each phase leg is controlled by a separate HCC, which compares its phase leg current against a commanded reference current to generate a per phase current error. The current errors are then compared against three separate sets of (fixed) hysteresis bands to independently switch each inverter phase leg to the upper or lower DC supply voltage in order to control its output current.

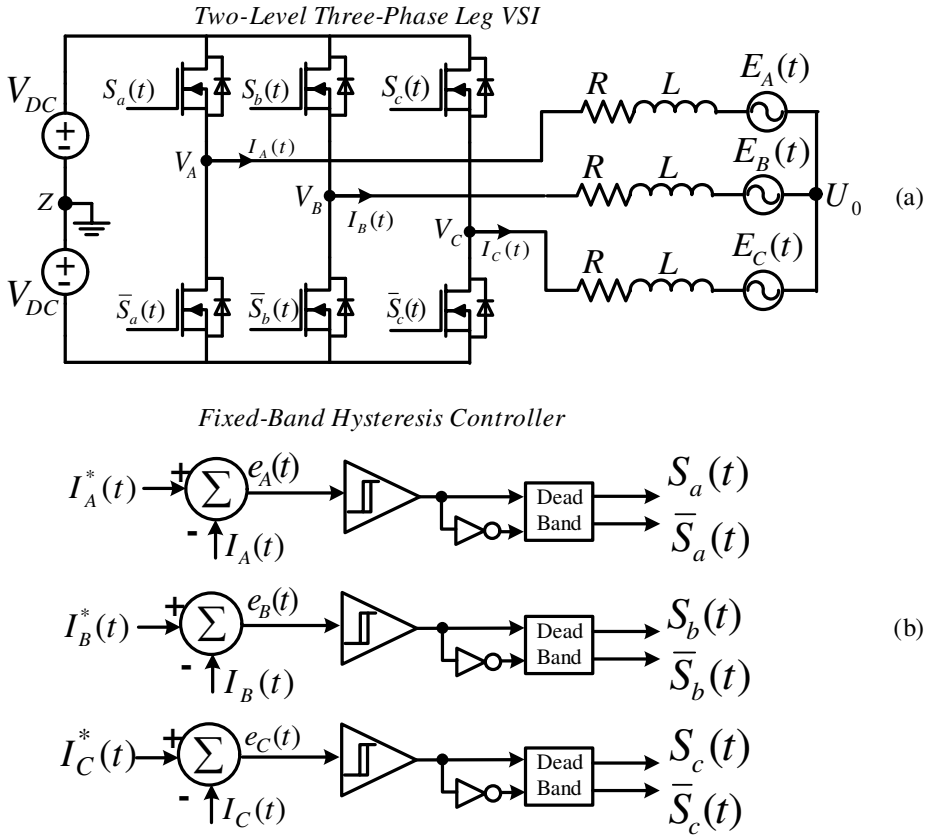


Figure 4.1: (a) Three-phase leg two-level voltage source inverter feeding a back-emf type load with series resistance and inductance (b) Block diagram of a conventional two-level three-phase fixed-band hysteresis controller

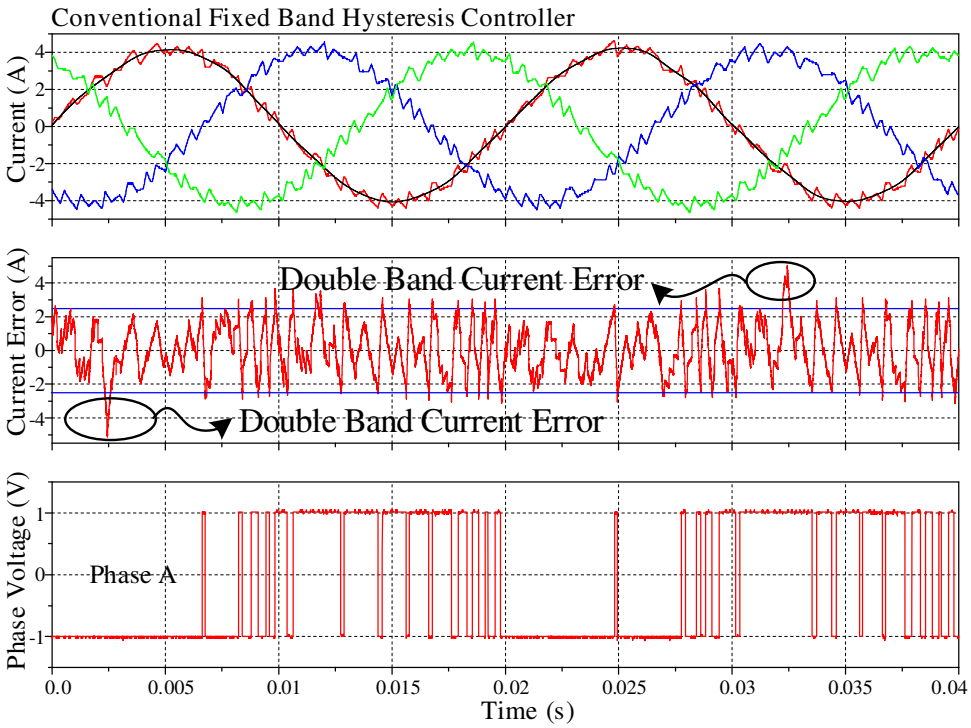


Figure 4.2: Experimental performance of conventional fixed band hysteresis current regulation, (Modulation depth = 0.8)

4.1.1 Double Band Current Error and Irregular Switching Effects in a Conventional Three-Phase HCC

For the hysteresis current regulated VSI shown in Figure 4.1, the instantaneous switching of one inverter phase leg influences the voltage applied to the other two phase legs because the three-phase load back-emfs can mutually interact [51][66][67] with each other. This coupling between the inverter phase legs generates a common mode current that is called the interacting current [51][66][67], which can cause each per phase current error to reach as much as twice the fixed hysteresis band while also causing irregular switching of the inverter [51].

Figure 4.2 illustrates this phenomenon experimentally, showing the three phase regulated currents, the A phase leg current error and the A phase leg switched voltage. The figure shows the well-known limitations of this conventional hysteresis regulator response such as substantial and irregular current ripple for each phase current, occasional excursions of the current error outside the fixed band limits [51], and a wide variation in phase leg switching frequency over the fundamental cycle, with long periods of no switching at all. The interaction effect can be mathematically quantified as follows.

For the isolated balanced three-phase load system of Figure 4.1(a), the summation of the three-phase load currents and the reference currents are always zero, viz:

$$I_A^*(t)+I_B^*(t)+I_C^*(t)=0 \tag{4.1}$$

$$I_A(t)+I_B(t)+I_C(t)=0 \tag{4.2}$$

Subtracting (4.1) from (4.2) and rearranging it for the three-phase current errors $e_A(t)$, $e_B(t)$ and $e_C(t)$, gives:

$$\underbrace{(I_A^*(t)-I_A(t))}_{Phase\ A\ current\ error}+\underbrace{(I_B^*(t)-I_B(t))}_{Phase\ B\ current\ error}+\underbrace{(I_C^*(t)-I_C(t))}_{Phase\ C\ current\ error}=0\rightarrow e_A(t)+e_B(t)+e_C(t)=0 \tag{4.3}$$

where $e_x(t)=I_x^*(t)-I_x(t)$ ($x\in A,B,C$) is the per phase current error.

Equation (4.3) confirms that for a balanced three-phase system, the summation of the three-phase current errors is always zero. Hence, if the instantaneous values of any "two out of the three-phase current errors" are either at the maximum positive hysteresis band $+I_h$ or at the minimum negative hysteresis band $-I_h$, the third phase current error can reach twice the hysteresis band magnitude. This understanding can be summarized mathematically as:

$$\begin{aligned} &if\ e_A(t)=+I_h\ \ and\ \ e_B(t)=+I_h \\ &I_h+I_h+e_C(t)=0\ \rightarrow\ e_C(t)=-2I_h \end{aligned} \tag{4.4}$$

$$\begin{aligned} &if\ e_A(t)=-I_h\ \ and\ \ e_B(t)=-I_h \\ &-I_h-I_h+e_C(t)=0\ \rightarrow\ e_C(t)=+2I_h \end{aligned} \tag{4.5}$$

Figure 4.3 shows the experimental confirmation of this phenomenon, expanding in particular the instant where the phase B and C current errors have a value of $-I_h$, so that the phase A current error reaches twice the maximum hysteresis band of $+2I_h$

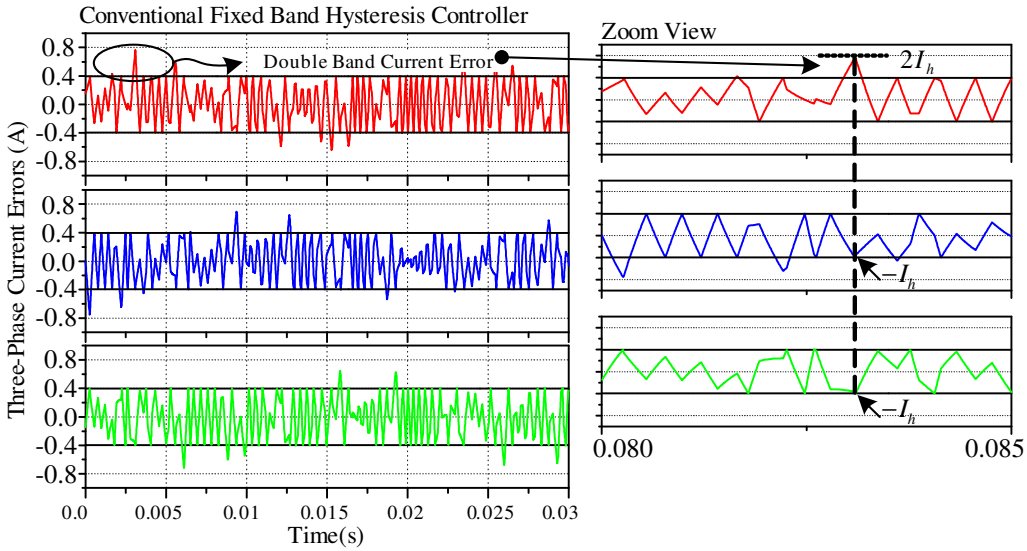


Figure 4.3: Illustration of the double band current error phenomenon in conventional fixed band hysteresis current regulation, (Modulation depth = 0.8)

4.1.2 Common Mode Current in Frame Transformed Hysteresis Current Regulators

Previous work [66][67] has shown that to achieve effective three-phase hysteresis current regulation, the common mode interacting current must be removed from the measured output load current before making the hysteresis switching decision. One strategy that is sometimes proposed to achieve this outcome is SV-based HCC, using either the stationary $\alpha\beta$ or synchronous dq frame of reference [3][93][96]. However, neither transformation is actually effective, as will now be demonstrated.

4.1.2.1 Stationary αβ Frame

In the stationary $\alpha\beta$ frame, the two orthogonal reference currents (I_α^*, I_β^*) and actual currents (I_α, I_β) are determined from the three-phase currents as:

$$\begin{bmatrix} I_\alpha^* \\ I_\beta^* \end{bmatrix} = \sqrt{\frac{2}{3}} \begin{bmatrix} 1 & -1/2 & -1/2 \\ 0 & \sqrt{3}/2 & -\sqrt{3}/2 \end{bmatrix} \begin{bmatrix} I_A^* \\ I_B^* \\ I_C^* \end{bmatrix} \tag{4.6}$$

$$\begin{bmatrix} I_\alpha \\ I_\beta \end{bmatrix} = \sqrt{\frac{2}{3}} \begin{bmatrix} 1 & -1/2 & -1/2 \\ 0 & \sqrt{3}/2 & -\sqrt{3}/2 \end{bmatrix} \begin{bmatrix} I_A \\ I_B \\ I_C \end{bmatrix} \tag{4.7}$$

Subtracting equations (4.7) from (4.6) to get the stationary frame current errors gives:

$$e_\alpha = I_\alpha^* - I_\alpha = \frac{\sqrt{2}}{2\sqrt{3}}(I_A^* - I_A) \longrightarrow e_\alpha = \frac{\sqrt{2}}{2\sqrt{3}} e_A \tag{4.8}$$

$$e_\beta = \frac{\sqrt{2}}{2}\left(\left(I_A^* - I_B^*\right) - \left(I_A - I_B\right)\right) = \frac{\sqrt{2}}{2}\left(I_{AB}^* - I_{AB}\right) \longrightarrow e_\beta = \frac{\sqrt{2}}{2} e_{AB} \tag{4.9}$$

Equation (4.8) confirms that e_α is simply a scaled version of the phase *A* current error e_A and hence the transformation from the ABC to the $\alpha\beta$ frames of reference does not remove the common mode interacting current.

4.1.2.2 Synchronous dq Frame

In the synchronous dq frames, the two rotating reference currents (I_d^*, I_q^*) and the actual currents (I_d, I_q) are given by:

$$\begin{bmatrix} I_d^* \\ I_q^* \end{bmatrix} = \frac{2}{3}\begin{bmatrix} \cos\theta & \cos(\theta - 120°) & \cos(\theta + 120°) \\ \sin\theta & \sin(\theta - 120°) & \sin(\theta + 120°) \end{bmatrix}\begin{bmatrix} I_A^* \\ I_B^* \\ I_C^* \end{bmatrix} \tag{4.10}$$

$$\begin{bmatrix} I_d \\ I_q \end{bmatrix} = \frac{2}{3}\begin{bmatrix} \cos\theta & \cos(\theta - 120°) & \cos(\theta + 120°) \\ \sin\theta & \sin(\theta - 120°) & \sin(\theta + 120°) \end{bmatrix}\begin{bmatrix} I_A \\ I_B \\ I_C \end{bmatrix} \tag{4.11}$$

Subtracting equations (4.11) from (4.10) to get the synchronous frame current errors gives:

$$\begin{aligned} e_q &= I_q^* - I_q = (I_A^* - I_A)\cos(\theta) + \frac{1}{\sqrt{3}}\left(I_{BC}^* - I_{BC}\right)\sin(\theta) \\ \rightarrow e_q &= e_A\cos(\theta) + \frac{1}{\sqrt{3}} e_{BC}\sin(\theta) \end{aligned} \tag{4.12}$$

$$e_d = I_d^* - I_d = (I_A^* - I_A)\sin(\theta) - \frac{1}{\sqrt{3}}\left(I_{BC}^* - I_{BC}\right)\cos(\theta)$$
$$\rightarrow e_d = e_A \sin(\theta) - \frac{1}{\sqrt{3}} e_{BC} \cos(\theta) \tag{4.13}$$

Equations (4.12) and (4.13) confirm that the common mode interacting current still exists in both d and q components of the current error, since they both contain a term with the phase A current error only.

From this analysis presented in this section, it can be concluded that frame transformation of the three-phase currents does not remove the common mode interacting current from the measured phase leg currents. Hence, SV-based hysteresis current control strategies will still suffer from the influence of this effect and so neither of above two approaches are effective to eliminate common mode interference between the phase legs.

4.2 Hysteresis Current Regulation of a Three-Phase VSI

This section develops a new constant frequency hysteresis current regulator for a two-level three-phase VSI that achieves a harmonic performance that is very close to open-loop CSVM. The major development stages of the hysteresis current regulator for the three-phase VSI in Figure 4.1(a), based on the single-phase leg VSI in Figure 3.1(a), are as follows:

- Additional hysteresis current regulator for phase leg B.
- Compensation of the common mode interacting current.
- Synchronisation of the phase A and phase B current error zero-crossings to a fixed reference clock.
- Modulation of phase leg C using open-loop sine-triangle PWM.
- Extension of the modulation depth region by injecting a third harmonic component.

4.2.1 Extension of New Hysteresis Current Regulation Approach to Three-Phase VSI

From Figure 4.1 (a), the inverter load relationship can be written as:

$$V_x(t) = RI_x(t) + L\frac{dI_x(t)}{dt} + E_x(t) + U_0(t) \qquad x \in A, B, C \tag{4.14}$$

where:

$V_x(t)$ is the phase-leg switched voltage.

$I_x(t)$ is the phase-leg output current.

R_x is the phase-leg output load resistance.

L_x is the phase-leg output load inductance.

$E_x(t)$ is the phase-leg output load AC back-emf.

$U_0(t)$ is the common mode (floating) neutral point voltage.

Note that the only difference between equation (4.14) and the equivalent single-phase equation (3.1) is the additional load neutral point voltage $U_0(t)$.

From previous work [66][67], it is known that in order to achieve effective three phase hysteresis current regulation, it is necessary to separate the common mode interacting current component from the measured current for each phase. Only the non-interacting current error component is then used for the process of hysteresis comparison. From this understanding, each per phase output current of a three-phase VSI can be separated into two components, as follows:

$$I_x(t) = I_x^s(t) + \gamma(t) \qquad x \in A, B, C \qquad (4.15)$$

where:

$I_x(t)$ is the phase-leg output current.

$I_x^s(t)$ is the non-interacting component of the phase-leg current.

$\gamma(t)$ is the common mode interacting current.

Now, substituting equation (4.15) into (4.14), the load equation can be restated as:

$$V_x(t) = R\left(I_x^s(t) + \gamma(t)\right) + L\frac{d\left(I_x^s(t) + \gamma(t)\right)}{dt} + E_x(t) + U_0(t) \qquad x \in A, B, C \qquad (4.16)$$

If the load neutral point voltage is equated to:

$$U_0(t) = -L\frac{d\gamma(t)}{dt} - R\gamma(t) \qquad (4.17)$$

equation (4.16) further simplifies to give:

$$V_x(t) = RI_x^s(t) + L\frac{dI_x^s(t)}{dt} + E_x(t) \qquad x \in A, B, C \qquad (4.18)$$

where $I_x^s(t) = I_x(t) - \gamma(t)$ is the non-interacting component of the phase current.

In this form, equation (4.18) is identical to the single-phase leg hysteresis controlled system described in chapter 3, if the interacting current $\gamma(t)$ is subtracted from the measured current before the hysteresis control process.

As before, the non-interacting component of the per phase output current can now be further split into the fundamental component $I_{x,f}^s(t)$ and the switching ripple component $I_{x,r}^s(t)$. Then, using the same mathematical analysis as was presented in

section 3.2.1, the per phase switching ripple component $I_{x,r}^{s}(t)$ can be separated into two switching time events over one switching period, to achieve a time varying hysteresis band expression of:

$$I_{x,h}(t) = I_{h,\max}\left(1 - \left(V_{x,avg}(t)/V_{DC}\right)^2\right) \qquad x \in A,B,C \qquad (4.19)$$

where $I_{h,\max} = \dfrac{V_{DC}}{4Lf_{sw}}$ is the maximum allowable hysteresis band.

In this form, the three-phase hysteresis current regulator has three separate controllers similar to the hysteresis current regulator described in chapter 3 (Figure 3.6). Each controller subtracts its corresponding per phase leg output current from a commanded reference, and switches its converter phase leg to $+V_{DC}$ or $-V_{DC}$ when the corresponding current error crosses the lower or higher (respectively) variable hysteresis band $I_{x,h}(t)$.

Each three-phase average inverter output voltage can also be measured in a similar way as described in section 3.2.2, using the phase leg switching signals to give:

$$V_{x,avg} = 2V_{DC}\left(\frac{T_{x,2} - T_{x,1}}{T_{x,3} - T_{x,1}} - 0.5\right) \qquad x \in A,B,C \qquad (4.20)$$

4.2.2 Compensation of the Common Mode Interacting Current

4.2.2.1 Common Mode Current in the ABC Frame of Reference

In a balanced three-phase system, the sum of the three-phase currents is always zero, and the sum of the three-phase back-emf voltages must also be zero, as follows:

$$I_A(t) + I_B(t) + I_C(t) = 0 \qquad (4.21)$$

$$E_A(t) + E_B(t) + E_C(t) = 0 \qquad (4.22)$$

Now from this knowledge, summing equation (4.14) across all three phases and solving for the load neutral point voltage gives:

$$-3U_0(t)+V_A(t)+V_B(t)+V_C(t)=$$
$$R(I_A(t)+I_B(t)+I_C(t))+L\frac{d(I_A(t)+I_B(t)+I_C(t))}{dt}+E_A(t)+E_B(t)+E_C(t)$$

$$\Rightarrow U_0(t)=\frac{V_A(t)+V_B(t)+V_C(t)}{3} \tag{4.23}$$

Equation (4.23) shows that the load neutral point voltage can be expressed in terms of the instantaneous switching state of three switched phase voltages. Now equating (4.17) and (4.23) and ignoring the output load resistance R (this assumption is valid for most inverters because the L/R time constant of the filter inductance is almost always much longer than the switching period of the inverter), this the common mode interacting current can be directly calculated from the inverter switched outputs as follows:

$$\begin{aligned}\gamma(t)&=-\frac{1}{L}\int U_0(t)dt\\&=-\frac{1}{3L}\int[V_A(t)+V_B(t)+V_C(t)]dt\end{aligned} \tag{4.24}$$

Equation (4.24) provides a mathematical expression for the common mode interacting current in terms of the three-switched phase voltages. Once this common mode current is subtracted from each phase's measured current, the new variable hysteresis band concepts developed in chapter 3 can be immediately applied to the three phase VSI.

4.2.2.2 Practical Implementation of Compensating the Common Mode Current

From equation (4.24), it is necessary to measure the three switched phase voltages to calculate the common mode interacting current [66][67]. Knowing the fact that the three switched phase voltages can be easily replicated from their corresponding switching signals $S_a(t)$, $S_b(t)$ and $S_c(t)$, the phase leg switched voltages can be expressed as:

$$V_x(t)=2V_{DC}(S_x(t)-0.5) \qquad x\in\{A,B,C\}, S_x(t)\in\{0,1\} \tag{4.25}$$

where:

$V_x(t)$ is the phase-leg switched voltage.

V_{DC} is half of the total DC bus voltage.

$S_x(t)$ is the phase-leg switching signal.

Now from (4.25), the common mode interacting current can be restated in terms of the three-phase switching signals as:

$$\gamma(t) = -\frac{2V_{DC}}{3L}\int[(S_A(t)+S_B(t)+S_C(t))-1.5]dt \qquad S_x(t)\in\{0,1\} \tag{4.26}$$

Equation (4.26) shows that the common mode interacting current can be easily measured using the three-phase switching signals with appropriate gain and offset adjustment, as shown in Figure 4.4. Furthermore, the three-phase gate signals have already been used to measure the three-phase average inverter output voltages for calculation of the variable hysteresis bands.

Measuring the common mode current in this way has the following advantages:

- There is no need for additional voltage sensors to measure the three switched phase voltages [66][67].
- The isolation between the control circuitry and the power stage of the system is well maintained since measuring the actual three switched phase voltages is not required.
- Calculating the common mode current from the logic PWM signals minimizes the noise associated with the analog measurement circuitry.

Figure 4.5 provides simulation results to confirm the performance of compensating for the common mode current using the three-phase gate signals. Figure 4.5(a)(b) shows the normalized three-phase gate signals and the reconstituted

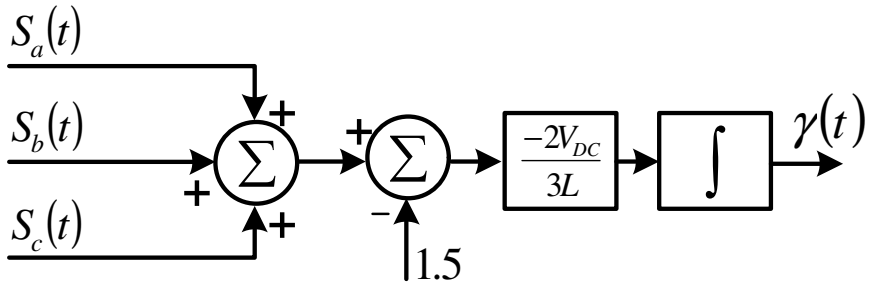


Figure 4.4: Measuring the common mode interacting current from the instantanous switching states of the three-phase gate signals

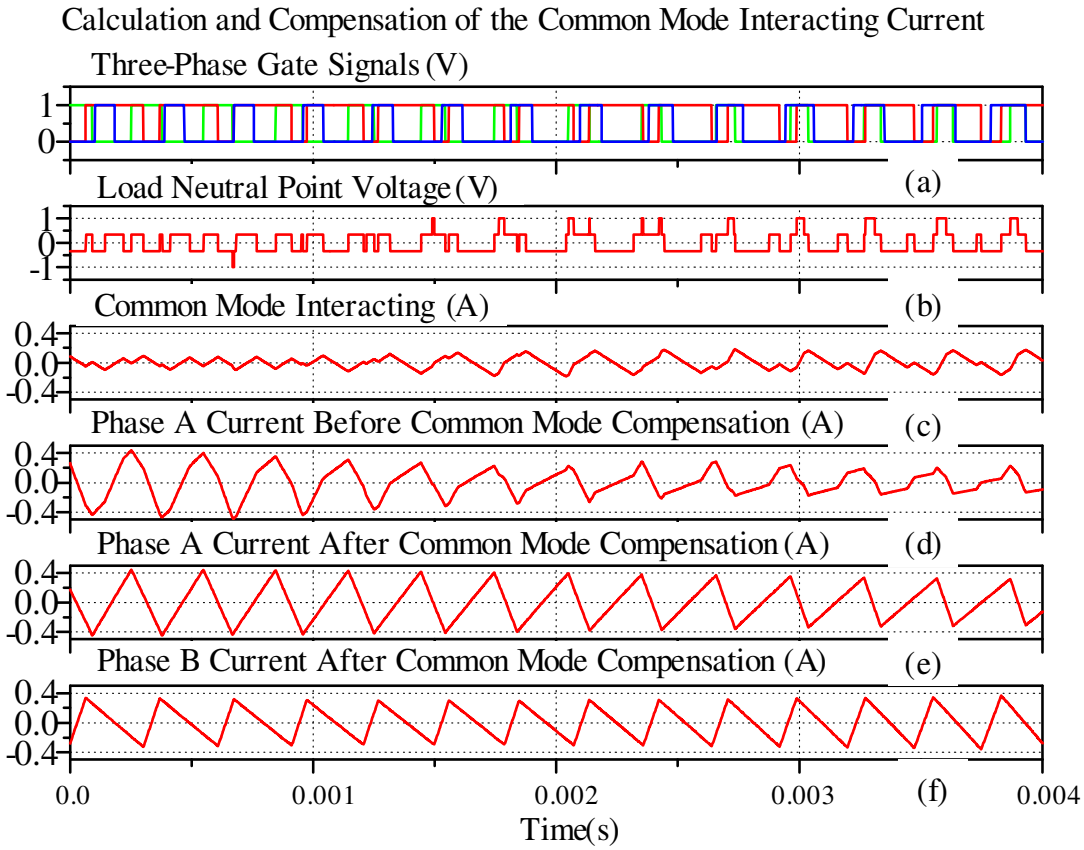


Figure 4.5: Compensation of the common mode interacting current (a) three phase gate signals (b) normalized load neutral point voltage (c) common mode interacting current (d) phase current error before common mode compensation (e) phase A current error after common mode (f) phase B current error after common mode

load neutral point voltage. Figure 4.5(c) shows the common mode interacting current calculated using the three-phase gate signals as per equation (4.26). Figure 4.5(d) shows the phase A current error before compensation of the common mode current. As expected, the current error has a non-uniform switching behaviour caused by the common mode current in Figure 4.5(c). Figure 4.5(e)(f) shows the resultant non-interacting components of the phase A and phase B currents after common mode compensation, with ripple caused only by the per phase leg switching processes. The figure confirms that calculation of the common mode current from equation (4.26) achieves uniform triangular phase A and phase B current errors which are then used for the hysteresis comparison against a set of variable hysteresis bands to generate the per phase switching signals. It should be noted that from Equation (4.26), knowledge of the DC bus voltage and the load inductance is required to correctly scale the common mode interacting current. However the hardware arrangement of the controller in this thesis allows the DSP to tune the scaled bus voltage DAC output to match any particular load inductance, as described in Chapter 8.

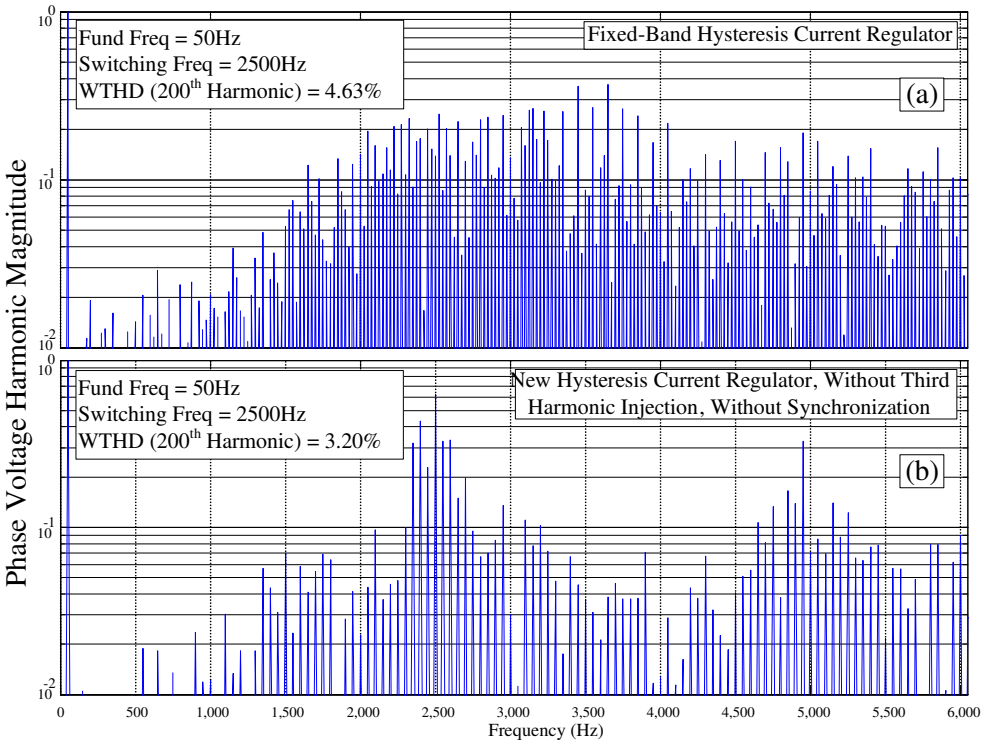


Figure 4.6: Phase leg voltage harmonic spectra (a) Fixed-band HCC (b) New variable band hysteresis current regulator - peak backemf = $0.9V_{DC}$

Figure 4.6 provides experimental results to confirm the phase voltage harmonic performance of the new hysteresis current regulator against the conventional fixed band hysteresis current regulator. The harmonic performance of the new three-phase variable band hysteresis current regulator as shown in Figure 4.6(b) is very similar to that of the open loop PWM controller, with a major carrier frequency harmonic (which is a common mode across all three phase legs), and clear sideband components. In contrast, the classical fixed frequency hysteresis controller shows the well-known wide band noise spectrum and increased THD that is expected with this type of controller. By comparing these two figures, it can be seen that the WTHD is significantly reduced from 4.63% for a conventional HCC to 3.20% for the new variable band hysteresis current regulator.

4.2.3 Synchronisation to a Fixed Reference Clock

For a two-level three-phase system, synchronization of the three-phase current error zero-crossings to a reference clock ensures that the VSI only selects the “three nearest” space vectors within each switching cycle [84]. Furthermore, this ensures that the inverter switched phase voltages are aligned within each switching period. This switching process is well known to produce an output harmonic spectrum that is

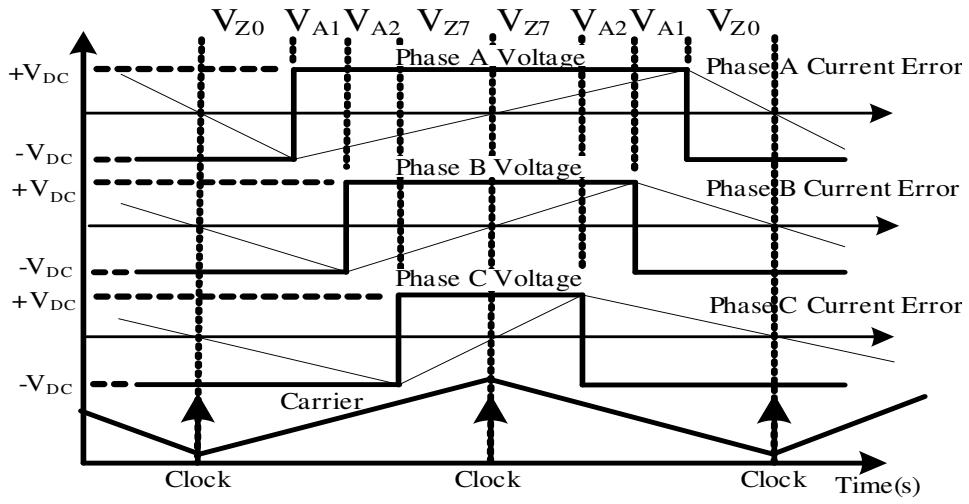


Figure 4.7: Balanced Three Phase Current Error Zero Crossings with centered three phase leg switched outputs.

close to optimum CSVM by evenly sharing the zero voltage space vector periods between the start and the end of each half carrier switching sequence [11][15][16]. Figure 4.7 shows this process for a fixed frequency carrier PWM process, where the phase current errors cross zero at the peak and trough of the modulating triangular carrier waveform and the inverter follows a zero/active space vector switching sequence of $V_{Z0} - V_{A1} - V_{A2} - V_{Z7}$ during the first half carrier period, and the reverse sequence $V_{Z7} - V_{A2} - V_{A1} - V_{Z0}$ during the second half carrier period.

Hence, this is the target switching objective for the new three-phase variable band hysteresis current regulation approach presented in this chapter.

The current error synchronisation process developed in section 3.3 can be readily extended to a three-phase VSI by calculating the appropriate band offset per phase leg as follows:

$$\delta I_{x,h} = -I_{x,h,old} * 2 f_{sw} * \delta t_x \qquad x \in A, B, C \qquad (4.27)$$

$$I_{x,h,new} = \delta I_{x,h} + I_{x,h,old} \qquad x \in A, B, C \qquad (4.28)$$

where δt_x is time difference between the actual zero-crossing of the per phase current error and the expected zero crossing time from the target clock.

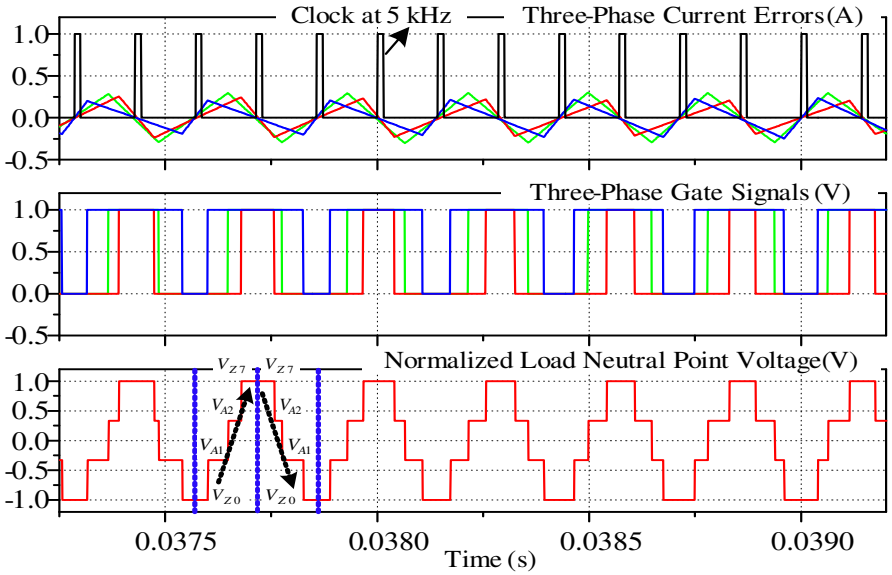


Figure 4.8: Detailed switching performance showing excellent frequency control and synchronized phase leg switching cycles.

Figure 4.8 shows the simulation results to confirm the performance of the synchronisation process for the three-phase VSI in a similar way as for the single-phase VSI shown in Figure 3.14.

Figure 4.8(a) shows the precision of the synchronised three-phase current error zero-crossing positionings which are almost exactly lined up with the 5kHz clock. Figure 4.8(b) confirms that the synchronization process using the additional feedforward band offset, ensures to achieve a significantly improved synchronisation process by eliminating the instantaneous phase delay between the three-phase VSI pulses. Figure 4.8 (c) shows the load neutral point voltage to further confirm the synchronisation process, identifying the even distribution of the inverter zero voltage space vector periods between the start and the end of each half switching cycle. It can also be seen from this figure that the inverter follows a zero/active space vector switching sequence of $V_{Z0}-V_{A1}-V_{A2}-V_{Z7}$ during the first half carrier period, and the reverse sequence $V_{Z7}-V_{A2}-V_{A1}-V_{Z0}$ during the second half carrier period. The result is a significant improvement in space vector selection compared to the PLL strategies used by the previously reported dead-beat HCC [47][66][84] to achieve the ideal CSVPWM patterns shown in Figure 4.7.

A further advantage for this approach is that for a hysteresis current regulated VSI where the reference current is generated using a DAC, there is a fundamental hazard in terms of a potential double zero-crossing of the current error within one switching

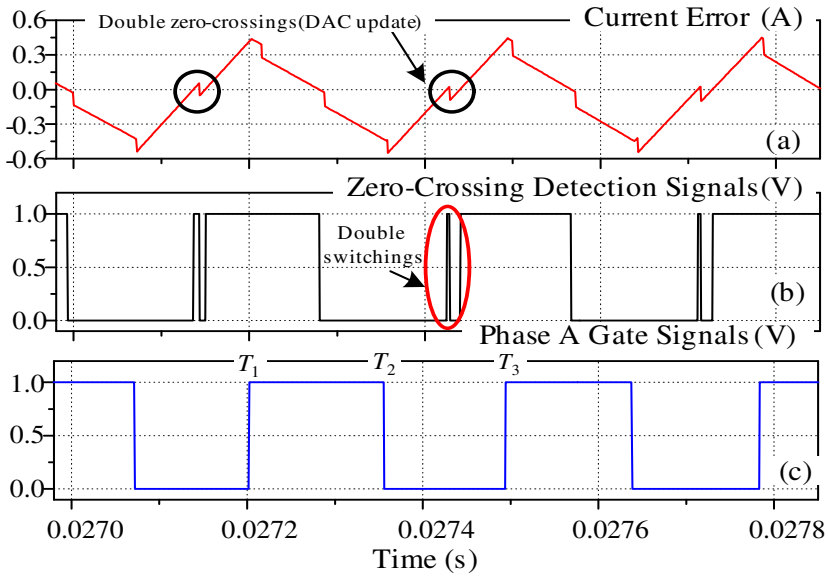


Figure 4.9: (a) Double zero-crossing of the current error (b) phase gate switching signal (c) Double switching caused by the double current error zero-crossings

cycle. The double zero-crossing effect is shown in Figure 4.9(a). The notch is caused by the sampled update of the target reference current from the controlling DSP, and may create additional spurious zero crossing events which can confuse a direct zero crossing detecting circuit, as shown in Figure 4.9(b). This significantly reduces the controller stability of dead-beat [84] and time–based hysteresis current regulators [81] since they directly measure the current error zero-crossings to predict the next switching time of the VSI. The proposed algorithm in this section avoids this hazard because of the way in which the current zero crossings are predicted by the measured phase leg switching events using the gate switching transition time as shown in as Figure 4.9(c), i.e.

$$Z_1 \approx (T_1 + T_2)/2 \tag{4.29}$$

$$Z_2 \approx (T_2 + T_3)/2 \tag{4.30}$$

Experimental results are provided in section 4.6 to confirm the effectiveness of this strategy to avoid current error double zero-crossings.

4.2.4 Implementation Using Two Current Regulators

4.2.4.1 Revisiting Linear Current Regulators

It is well established from linear modulation theory [51][53][54][52] that three phase current regulation is a two degree of freedom problem that only requires two closed loop controllers. Hence, using three independent controllers can result in unbalanced three-phase output currents and fluctuations in the load neutral point voltage [124], particularly when the nonidealities of a practical system are taken into account.

This over-constrained problem can be analysed as follows. Figure 4.10 shows the average load model circuit diagram of a three-phase system feeding a three-phase resistive and inductive load with AC back-emf. In this representation, the average load model of the phase-leg switched voltage is represented by the average inverter output voltage. From this figure, the two KVL loops, assuming negligible load resistance, can be written as:

$$V_A(t) = V_B(t) + 2L\frac{dI_A(t)}{dt} - L\frac{dI_B(t)}{dt} + E_A(t) - E_B(t) \tag{4.31}$$

$$V_B(t) = V_C(t) + 2L\frac{dI_B(t)}{dt} - L\frac{dI_A(t)}{dt} + E_B(t) - E_C(t) \tag{4.32}$$

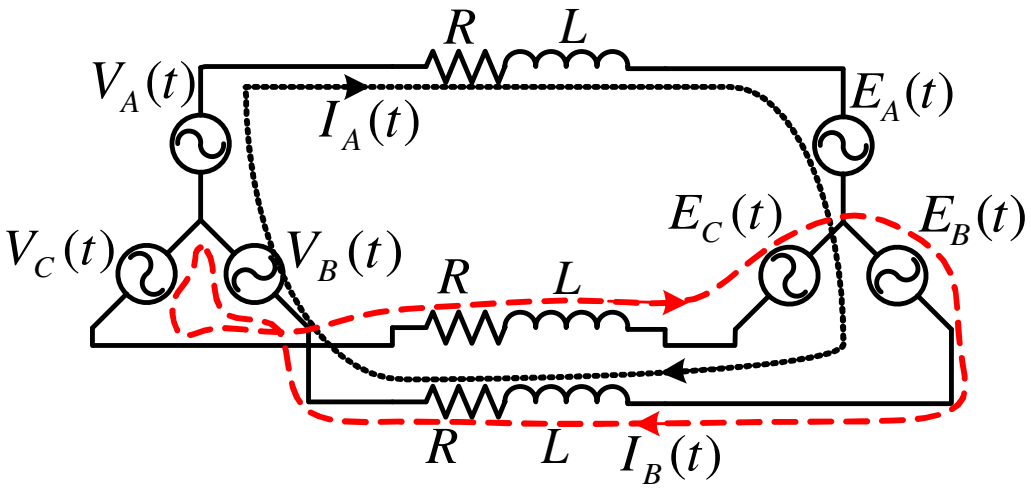


Figure 4.10: Equivalent average model of a three-phase VSI feeding a RL load with series AC back-emf

Now combining equations (4.31) and (4.32) and knowing that the summation of the three-phase currents and back-emfs is zero, it can be concluded that the summation of the three-phase average inverter output voltages must also be zero as follows:

$$V_A(t)+V_B(t)+V_C(t)=0 \tag{4.33}$$

Substituting equation (4.22) and (4.33) into (4.32), the phase B average voltage can be restated as:

$$2V_B(t)=-V_A(t)+2L\frac{dI_B(t)}{dt}-L\frac{dI_A(t)}{dt}V_C(t)+2E_B(t)+E_A(t) \tag{4.34}$$

Substituting equation (4.34) into (4.31) will give:

$$\begin{aligned} 3V_A(t)&=3L\frac{dI_A(t)}{dt}+3E_A(t)\\ I_A(t)&=\int\frac{V_A(t)-E_A(t)}{3}dt \end{aligned} \tag{4.35}$$

Equation (4.35) shows that the phase A output current can be independently controlled using the phase A average voltage and its load back-emf. The other two phase currents can then be similarly expressed as:

$$I_B(t)=\int\frac{V_B(t)-E_B(t)}{3}dt \tag{4.36}$$

$$I_C(t)=\int\frac{V_C(t)-E_C(t)}{3}dt \tag{4.37}$$

Equations (4.35)-(4.37) confirm that the per phase current can be independently identified from its average phase voltage and load back-emf. Additionally equation (4.21) shows that the summation of the three-phase currents is always zero. Hence, it can be concluded that for a three-phase system employing a linear current regulation system, it is common to implement only two closed loop current regulators, with the third phase leg being PWM controlled using a reference voltage calculated from the negative of the sum of the commanded voltages of the other two phase legs. For

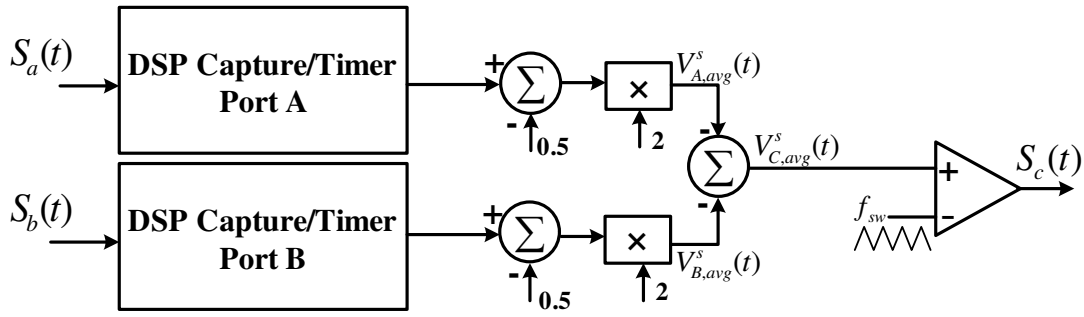


Figure 4.11: Details of third phase leg commanded modulation voltage calculated from the average of the two closed loop switched voltages.

linear regulators, this strategy is viable because the two linear closed loop regulators command an average voltage for their phase legs which feeds into their respective modulation systems.

4.2.4.2 New Three-Phase Hysteresis Current Regulator

Conventional three-phase hysteresis regulators require three separate regulators (one for each phase), since there is no obvious way to identify and sum an average commanded voltage for two phase legs, and use this to control the third phase leg.

The new three-phase hysteresis current regulator presented in this chapter overcomes this problem. The proposed HCC uses the information of the three-phase average inverter output voltages to calculate the three-phase variable hysteresis bands. Once these average load voltages are calculated for the phase legs A and B, the phase leg C average inverter voltage can then be directly calculated from (4.33) as:

$$V_{A,avg}(t)+V_{B,avg}(t)+V_{C,avg}(t)=0 \rightarrow \\ V_{C,avg}(t)=-\left(V_{A,avg}(t)+V_{B,avg}(t)\right) \tag{4.38}$$

The third phase leg average inverter output voltage is implemented as a scaled modulation command that can be fed to a standard sine-triangle pulse width modulator switching at f_{sw} as shown in Figure 4.11, without requiring a third phase leg current regulator. This simplifies the complexity and the cost of the analogue implementation of the new hysteresis current regulator. Additionally the line-to-line harmonic performance of the new hysteresis current regulation approach will be significantly improved since the third phase leg is now modulated using open-loop fixed frequency PWM.

Note that since the third phase modulation command is calculated from the other two average output voltages, the applied voltage lags behind the actual calculated modulation command due to the effect of sampling delay. However, this delay was readily compensated using linear extrapolation from the previous sampled values, as described in section 3.2.3.

The open-loop modulation of the third phase leg can be implemented both with and without synchronisation of the phase leg current error zero-crossings to a fixed reference clock. However using two synchronised hysteresis current regulators further improves the phase alignment of the three-phase VSI pulses since now the phase A and phase B current error zero-crossings are synchronised to the peak and trough of the phase C fixed-frequency triangle carrier.

Figure 4.12(a) provides simulation results, which shows the phases A and B normalized average inverter output voltages calculated using equation (4.20) This figure also shows the phase C modulation command calculated from the other two phase leg average voltages using equation (4.38), along with the third phase triangular carrier. Figure 4.12(b) shows the phase C switching gate signals that are generated using open-loop sine-triangle PWM shown in Figure 4.11.

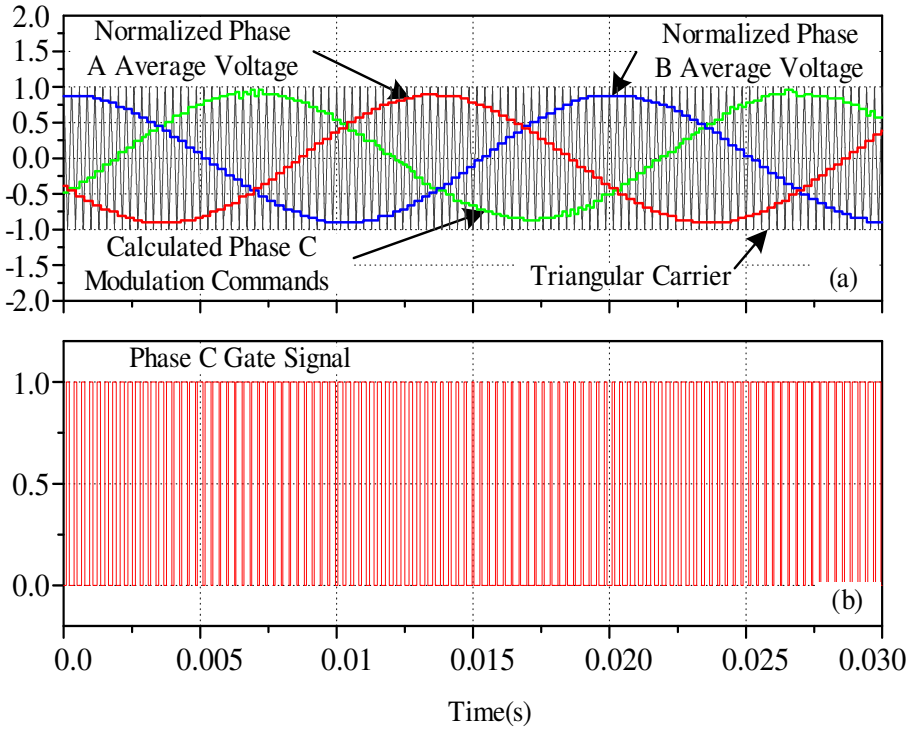


Figure 4.12: Third phase leg commanded modulation voltage extracted from the two hysteresis regulated phase legs and the generated third phase gate signal from the open-loop sine-triangle PWM.

4.3 Extending the Modulation Range Using Common Mode Third Harmonic Injection

4.3.1.1 Revisiting Linear Current Regulators

It is well known from open-loop PWM theory that the operating modulation range of an inverter can be extended to 1.15 by injecting a third harmonic zero-sequence component[2] to fully utilize the available DC bus voltage [11][15]. The third harmonic component is a common mode term across all three phases and hence does not affect the fundamental component of the three-phase line to line voltages. However, it does reduce the maximum peak amplitude of the per phase fundamental voltage which increases the linear modulation range of the inverter.

The third harmonic voltage can be calculated from the information of the three phase modulation commands [11] using:

$$V_{3rd} = \frac{\max(V_{A,avg}, V_{B,avg}, V_{C,avg}) + \min(V_{A,avg}, V_{B,avg}, V_{C,avg})}{2} \tag{4.39}$$

where:

$V_{A,avg}, V_{B,avg}, V_{C,avg}$ are the three-phase modulation commands

Linear regulators add this common mode third harmonic signal into the three-phase sinusoidal modulation commands after the current regulation process as shown in Figure 4.13.

In contrast, for a conventional three-phase hysteresis regulated VSI, knowledge of three-phase average load voltages is not normally available since the current control and modulation process are combined together. Hence a more fundamental approach is required to implement third harmonic injection into a three-phase hysteresis current regulated inverter.

[2] In this thesis, the term "third harmonic zero-sequence component" is used to generically identify all triplen series common mode harmonics that may be required to be injected as a compensating offset.

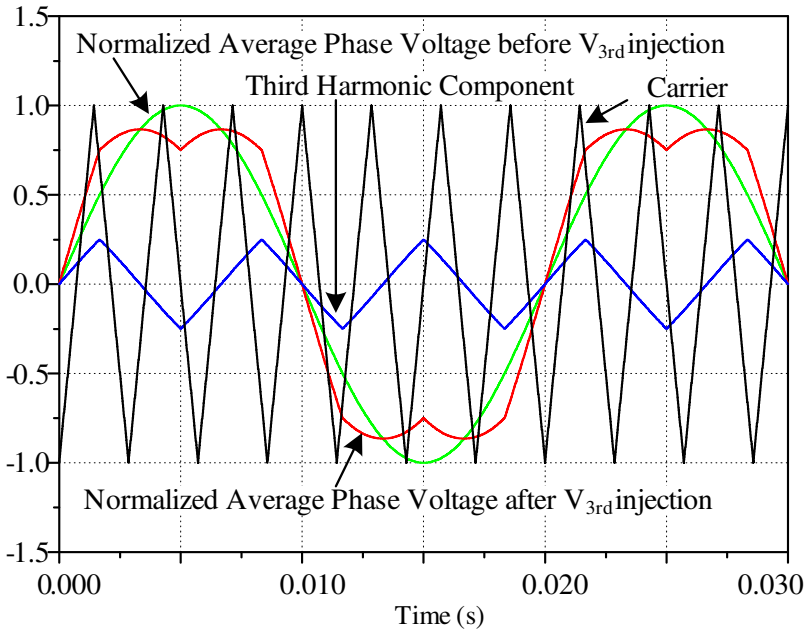


Figure 4.13: Normalized average inverter output voltage (open-loop modulation command) both before and after the injection of common mode third harmonic component.

4.3.1.2 New Three-Phase Hysteresis Current Regulator

The new three-phase variable band hysteresis current regulator uses the three-phase average inverter output voltages from equation (4.20), to calculate the three-phase variable hysteresis bands. Hence, in principle the common mode third harmonic term can be independently calculated using equation (4.39) similar to open-loop PWM.

The challenge is how to inject the third harmonic voltage into the hysteresis regulation process. Three possible approaches can be as follows:

- Method 1: Inject into the third phase leg modulation command, anticipating phase leg A and phase leg B will track this change.
- Method 2: change the variable hysteresis bands to include the third harmonic component.
- Method 3: Adjust the common mode interacting current calculation.

For method 1, the third harmonic voltage term would be injected directly into the third phase leg, which is controlled using an open-loop sine-triangle PWM. Unfortunately, this third harmonic voltage is not tracked by the A and B phases, and

hence causes unbalanced currents since there is no separate HCC to compensate for this effect. Consequently, this approach is not viable.

For method 2, the third harmonic voltage term would be injected into the variable hysteresis band calculation by modifying the per phase average voltage and recalculating the variable hysteresis bands based on equation (4.19). However, the common mode third harmonic voltage term does not explicitly appear as a current term in the per phase current error, and so the regulator does not switch in such a way as to force a third harmonic voltage to appear in the per phase average inverter voltages. Consequently, this approach is not viable.

For method 3, the third harmonic voltage term is directly injected into the common mode interacting current, recognising that both the third harmonic voltage and the interacting current are common mode effects across all three-phase voltages. Thus the common mode third harmonic term is incorporated into the hysteresis modulation process by adding it to the common mode interacting current in a similar way as the common mode neutral point voltage $U_0(t)$. The third harmonic voltage term is similarly integrated similar into the calculation of the common mode interacting current to get a third harmonic current term of:

$$I_{3rd}(t) = \frac{1}{3L}\int V_{3rd}(t)dt \tag{4.40}$$

This third harmonic current I_{3rd} is then subtracted from the common mode interacting current $\gamma(t)$ to generate a "modified interacting current" $\gamma'(t)$ which can be subtracted from the per phase leg current error before making the hysteresis switching decision. Figure 4.14 shows the block diagram of this concept, which can

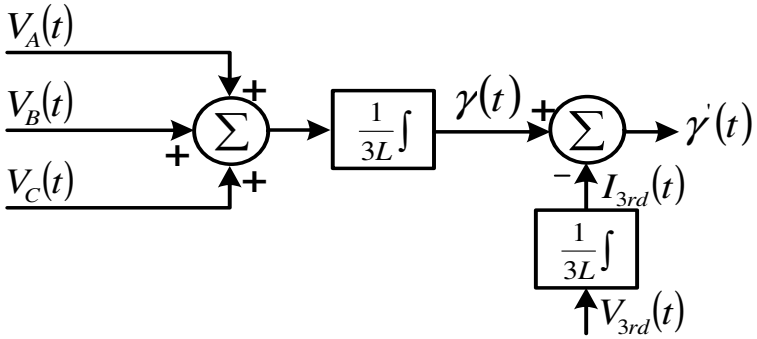


Figure 4.14: Block diagram of incorporation of common mode third harmonic voltage into the hysteresis modulation process

be mathematically expressed as:

$$\gamma'(t) = \gamma(t) - I_{3rd}(t) \tag{4.41}$$

where $\gamma'(t)$ is the modified interacting current with the included third harmonic current term.

In fact, it is not necessary to explicitly calculate the third harmonic current term. Instead, it can be directly incorporated into the calculation of the common mode current by subtracting the third harmonic voltage from the summation of three-switched phase voltages before the integration as:

$$\gamma'(t) = \frac{1}{3L}\int ([V_A(t) + V_B(t) + V_C(t)] - V_{3rd}(t))dt \tag{4.42}$$

Figure 4.15 shows the simulation results for the injection of the third harmonic voltage into the common mode current in this way. The figure shows the three-phase average inverter voltages and the variable hysteresis band calculated by the DSP with third-harmonic compensation included. The phase voltages clearly have the same well-known double-peak voltage waveform as CSVPWM, confirming that third-harmonic compensation has been correctly integrated into the algorithm. Additional

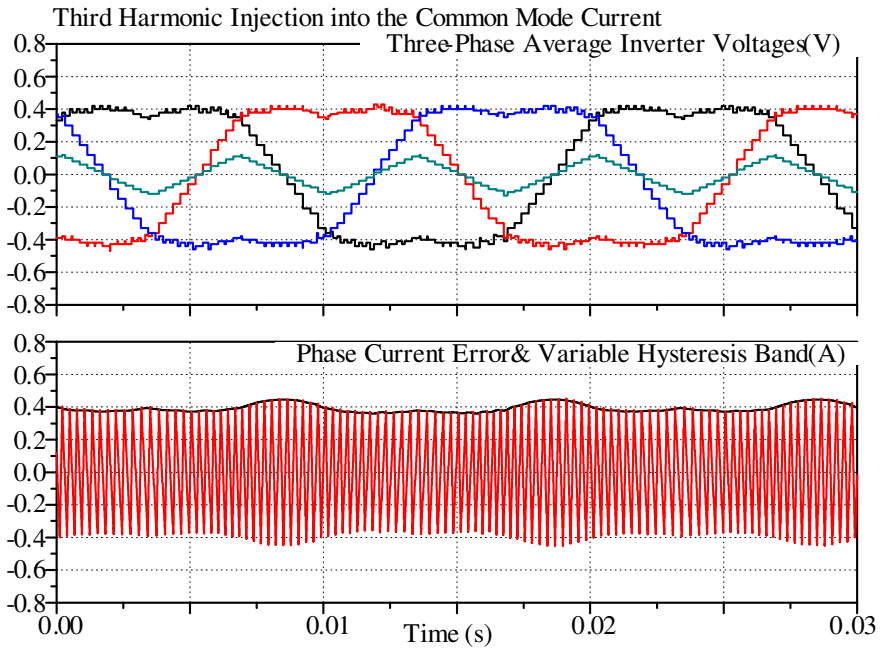


Figure 4.15: Injecting third harmonic voltage term into the common mode interacting current

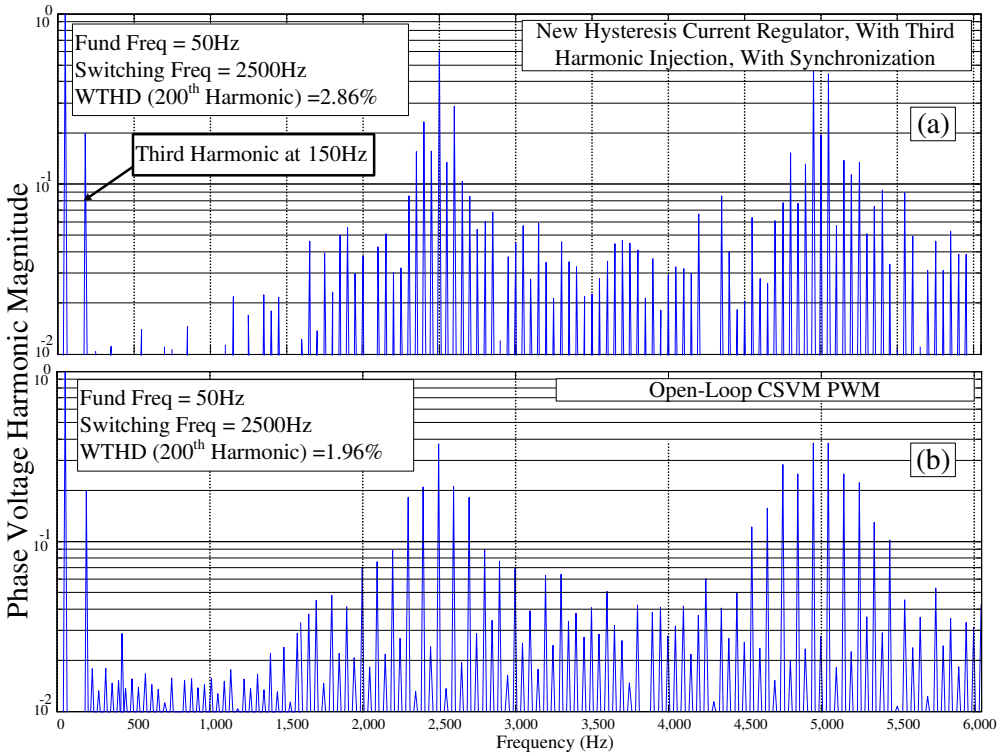


Figure 4.16: Phase leg voltage harmonic spectra (a) new HCC with common mode third harmonic injection, with current error synchronisation(b) open-loop CSVM PWM controller - peak backemf = $0.9V_{DC}$

experimental results are provided at the end of this chapter to confirm the effectiveness of this strategy at higher modulation depths.

Figure 4.16 provides experimental results to compare the harmonic performance of the new hysteresis current regulator with third harmonic injection, against open-loop CSVM. The figure confirms that the proposed HCC has achieved a target harmonic performance that is close CSVM, with a significant carrier at the target switching frequency of 2500Hz and the expected third harmonic voltage baseband harmonic at 150Hz. The result is a significant reduction in weighted total harmonic distortion from 3.20% in Figure 4.6(b), to 2.86% in Figure 4.16(b) achieved by injecting the third harmonic voltage terms, and synchronisation of the current error zero-crossings to the fixed clock.

4.4 Managing the Overmodulation Region

Section 3.4 has shown that, as the peak of the average inverter output voltage reaches the available DC bus voltage, the variable hysteresis band limit approaches zero. This causes high switching frequencies and the controller loses stability. The section then proposed a technique to solve the constraints of operating in the overmodulation region by clamping the variable hysteresis band at a minimum value as the controller enters this region.

Figure 4.17 illustrates the performance of this band limit clamping technique for a three-phase VSI at a modulation depth of 120%. Figure 4.17(a) shows how the regulators smoothly progress into overmodulation with extremely good current control even in the nonlinear overmodulation region. There is also no sign of transient disturbance, as the inverter enters and leaves this region of operation. Figure 4.17(b)(c) shows phases A and B current errors and variable hysteresis bands. From this figure it can be seen how the variable hysteresis band is clamped at the point when the regulator enters overmodulation and the VSI switching recommences after the per phase current error returns to within the hysteresis bands.

Figure 4.17(d) shows the normalized phase A and B average voltages and the phase C modulation command. Note in this figure, how the phase A and B average inverter voltages are clamped at a maximum value of about 90% as the controller enters overmodulation. The third phase modulation command also smoothly goes into overmodulation, as the sum of the phase A and phase B average voltages calculated using equation (4.38) exceeds one per unit.

Figure 4.17(e)(f)(g) shows the normalized three-switched phase voltages to further confirm that the controller maintains stability during overmodulation without any sign of high frequency switching in this region.

Further experimental and simulation results are provided at the end of this chapter for a modulation depth of 130% with third harmonic injection.

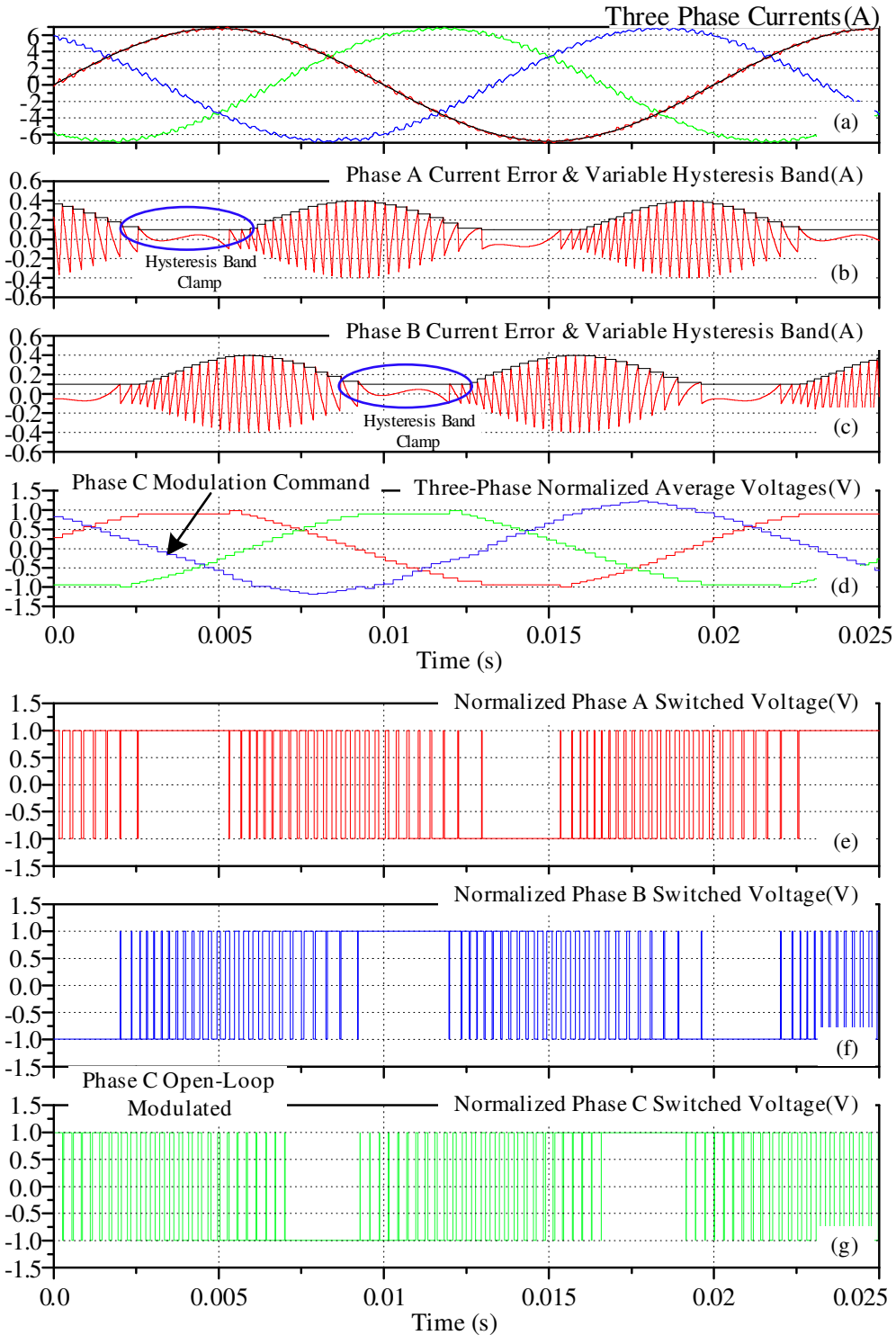


Figure 4.17: Simulated performance of variable band hysteresis current regulation with hysteresis band clamping, peak average inverter output voltage = $1.2*V_{DC}$

4.5 Implementation of the New Two-Level Three-Phase Hysteresis Current Regulator

So far, this chapter has presented the development stages of the new three-phase hysteresis current regulator with some simulation and experimental results to confirm its performance under various operating conditions. Figure 4.18 now shows the overall structure of the three-phase variable band hysteresis current regulation system.

For the two controlled phases, the primary hysteresis comparison process is implemented using a conventional op-amp summing junction to calculate the current errors $e_A(t)$ and $e_B(t)$, and to subtract the common mode interacting current $\gamma(t)$. The resultant errors are then fed to two variable threshold analogue comparators to create the phase leg switched output commands, which are post-processed by EPLD

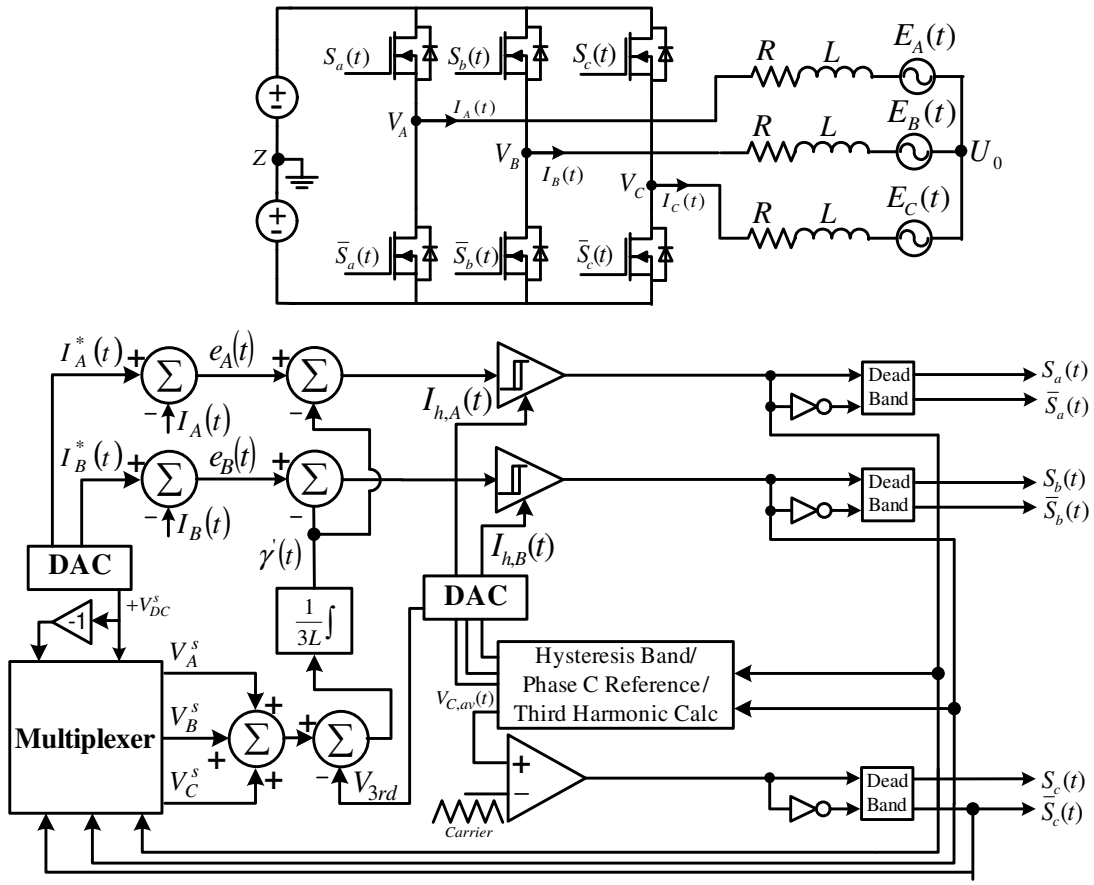


Figure 4.18: Experimental Implementation of two controller variable hysteresis band three-phase current regulator with common mode compensation and third harmonic injection

based dead-time logic counters to generate the final complementary phase leg switch commands $S_A(t), \overline{S_A}(t), S_B(t), \overline{S_B}(t)$.

The two hysteresis commanded phase leg outputs feed into the DSP capture/timer ports, and are used to calculate the average phase leg switched voltages using (4.20). The DSP then calculates the required variable hysteresis bands for these two phases to maintain a synchronized constant switching frequency using equations (4.19) and (4.28) and feeds them back to the analog comparison circuits using DACs. At the same time, the DSP digitally commands the switching of the third phase leg with a modulation command calculated using equation (4.38), as shown in more detail in Figure 4.11.

Finally, the phase leg switching commands are used to multiplex between positive and negative representations of the DC bus voltage, scaled to include the inductance division of equation (4.19) by another DAC output from the DSP. The three multiplexed outputs represent the instantaneous values of the actual phase leg switched voltages, which are then integrated using an analogue circuit to create the interacting current as per equation (4.24). Note that this arrangement allows the DSP to tune the scaled bus voltage DAC output to match any particular load inductance, without requiring hardware circuit changes.

For third harmonic injection, the DSP calculates the third harmonic voltage term from equation (4.39) which is then fed back into the analog circuitry used for the calculation of the interacting current.

4.6 Consolidated Simulation and Experimental Results

Throughout this chapter, selected simulation and experimental results have been presented in order to support the development of the theoretical analysis. These results are now re-presented in this section together with more extensive simulation and experimental results as a consolidated reference to show the operation of the new two-level three-phase hysteresis current regulator. Circuit parameters for the system are listed in Table 4.1.

Figure 4.19 (simulation) and Figure 4.20 (experimental) provide experimental results to confirm the effectiveness of the controller that has been developed for a two-level three-phase system. The figures show the centered three-phase normalized switched voltages and the reconstituted normalized load neutral point voltage. They also show the calculated common mode interacting current together with the phase current error both before and after compensation of the interacting current. Figure 4.20(c) shows the common mode interacting current calculated using the three-phase gate signals as per equation (4.26). These figures also confirm the calculation of the common mode current and compensation for this effect, to achieve the clean triangular phase A and phase B current errors which are then used for hysteresis comparison against a set of variable hysteresis bands to generate the per phase switching signals.

Table 4.1: Circuit parameters for the simulation and experimental implementations of the new hysteresis current regulation approach applied to the two-level three-phase VSI

Circuit Parameter		**Value**
Resistive load	(R) (Ω)	0.2
Inductive load	(L) (mH)	18
Target Switching frequency	(f_{sw}) (kHz)	2.5
Total DC bus voltage	($2V_{DC}$) (V)	100
Reference current	(I_{ref}) (A)	5
Back EMF frequency	(Hz)	50

Simulation

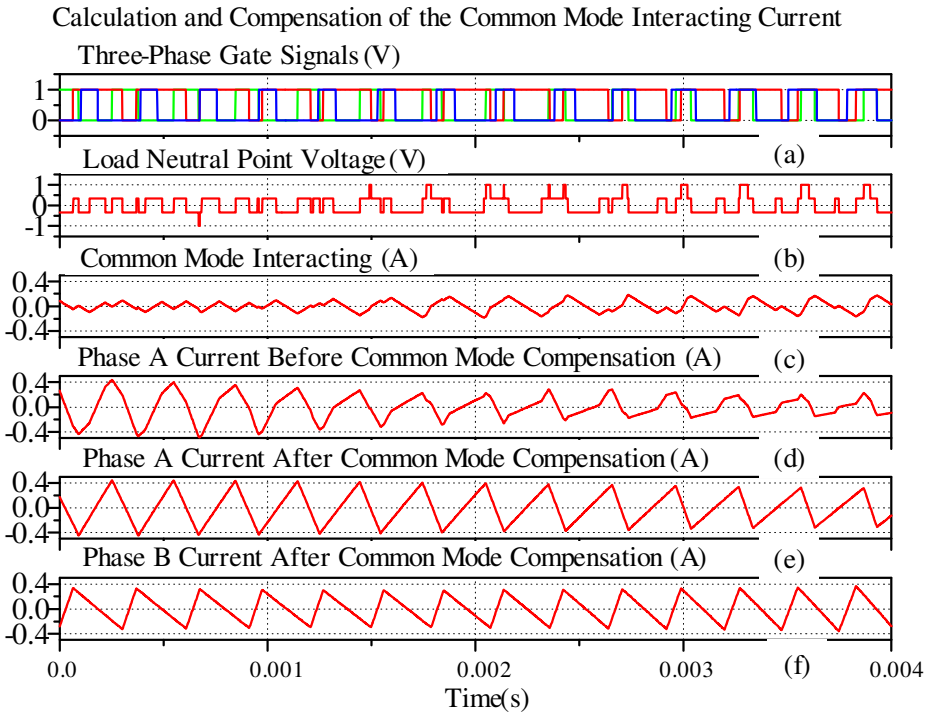


Figure 4.19: Compensation of the common mode interacting current (a) three phase gate signals (b) normalized load neutral point voltage (c) common mode interacting current (d) phase current error before common mode compensation (e) phase current error after common mode

Experiment

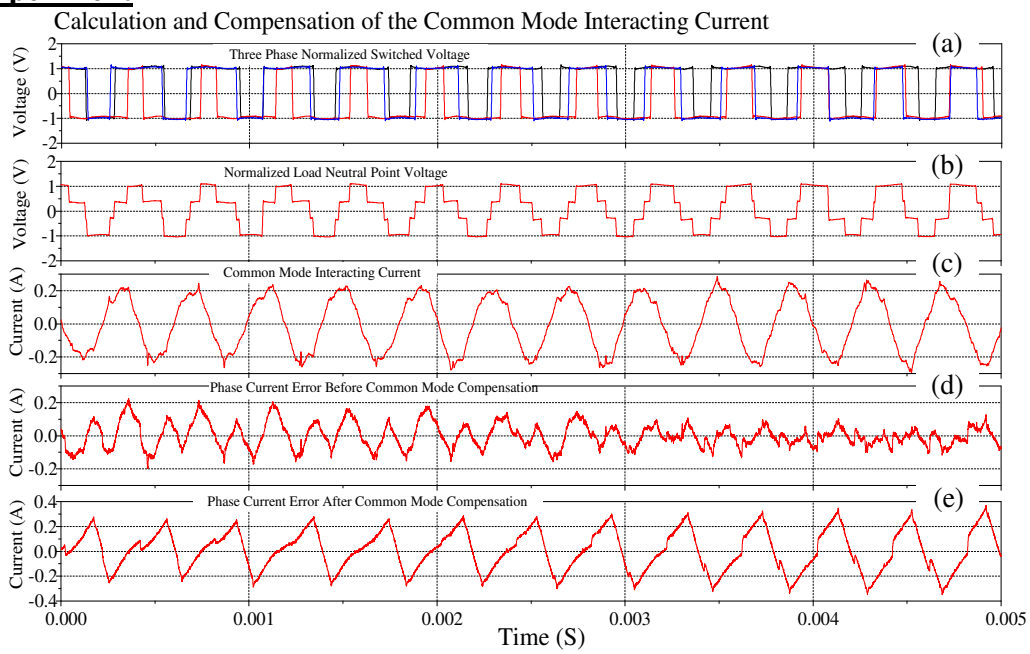


Figure 4.20: Compensation of the common mode interacting current (a) three phase normalized voltages (b) normalized load neutral point voltage (c) common mode interacting current (d) phase current error before common mode compensation (e) phase current error after common mode compensation

Figure 4.21 (simulation) and Figure 4.22 (experimental) show the steady state performance of the new variable hysteresis controller employing only two hysteresis current regulators. The figures show excellent current tracking capability with a totally bounded current error at all times (achieved by the presented technique to compensate the interacting current), an essentially constant phase leg switching frequency and a very regular and coherent inverter switching pattern. The phase C current error is not shown since it employs an open-loop sine-triangle PWM and its modulating command is calculated using equation (4.38).

Figure 4.23 (a)(b) shows the experimental harmonic spectra of the phase leg switched voltages for the classical conventional HCC and the new variable band hysteresis current regulator without third harmonic injection. The harmonic performance is quite close to that of an open loop PWM controller, with a major carrier frequency harmonic (which is common mode across all three phase legs), and clear sideband components. In contrast, the classical fixed frequency hysteresis controller shows the well-known wide band spectrum and increased THD that is expected with this type of controller.

Figure 4.24 (simulation) and Figure 4.25 (experimental) show the transient performance of the new two controller variable hysteresis band regulator to a 100% change in commanded current. From this figure it is clear that the excellent dynamic response expected from a hysteresis regulator has again not been at all compromised by the variable band implementation. The phase legs immediately switch to command the maximum possible current slew, and smoothly recommence variable band switching as the new commanded reference current is increased. Note also the increased variable hysteresis band fluctuation that occurs after the transient because the increased commanded current through the load system causes an effective increase in load back-emf.

These figures also nicely show the effect of common mode interaction on the actual phase currents. For both the simulation and experimental responses, the continuous ramp return of the phase A and B current errors after their errors have exceeded their hysteresis bands, confirm that their phase legs have clamped to a DC rail until the errors return to their band opposite limits and switching recommences. The apparent switching events during this clamping period that can be seen in the actual phase currents are in fact common mode interactions from other phase leg switchings, and have nothing to do with the phase leg hysteresis regulation

processes. The effect is particularly obvious for the phase leg A simulation response in Figure 4.24 because of the particular point in the commanded sine wave where the transient event was triggered, but is also present to some degree in the experimental response in Figure 4.25.

Simulation

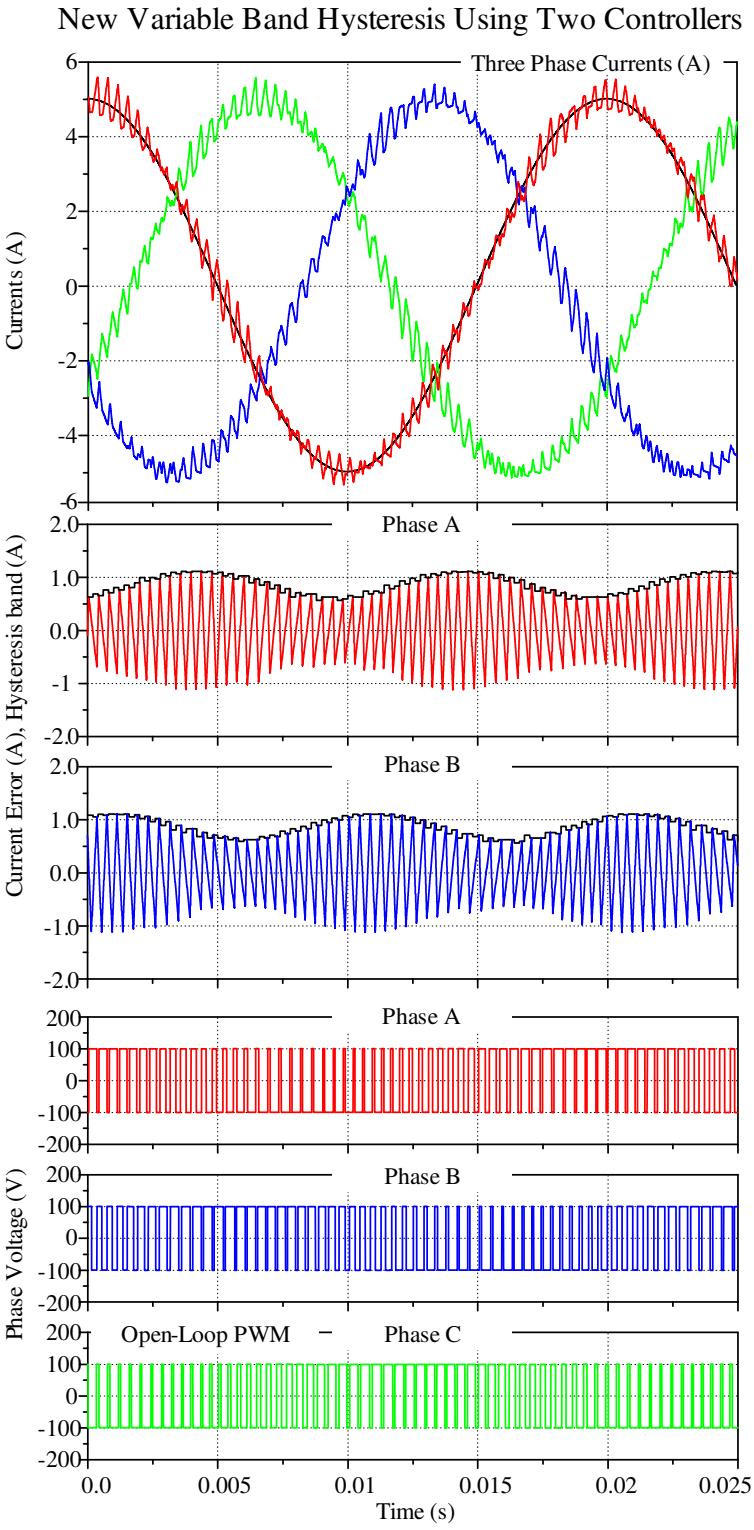


Figure 4.21: Performance of variable band hysteresis current regulation with common mode compensation and using two hysteresis controller
Modulation Depth = 0.9

Experiment

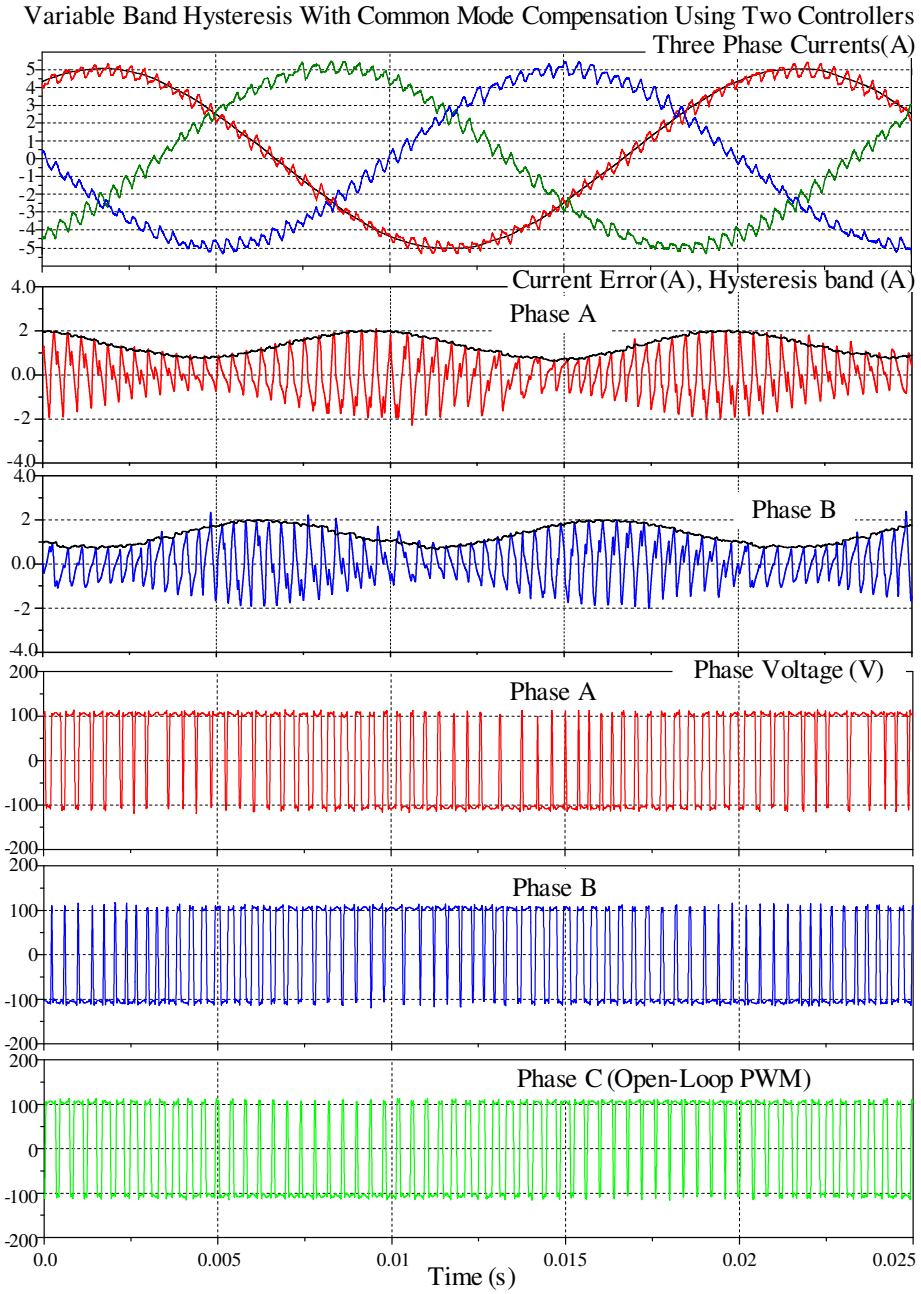


Figure 4.22: Experimental performance of the new variable band two controller three phase hysteresis regulator showing three-phase output currents, phase A and Phase B current errors and hysteresis bands, three-switched phase voltages.
Modulation Depth = 0.9

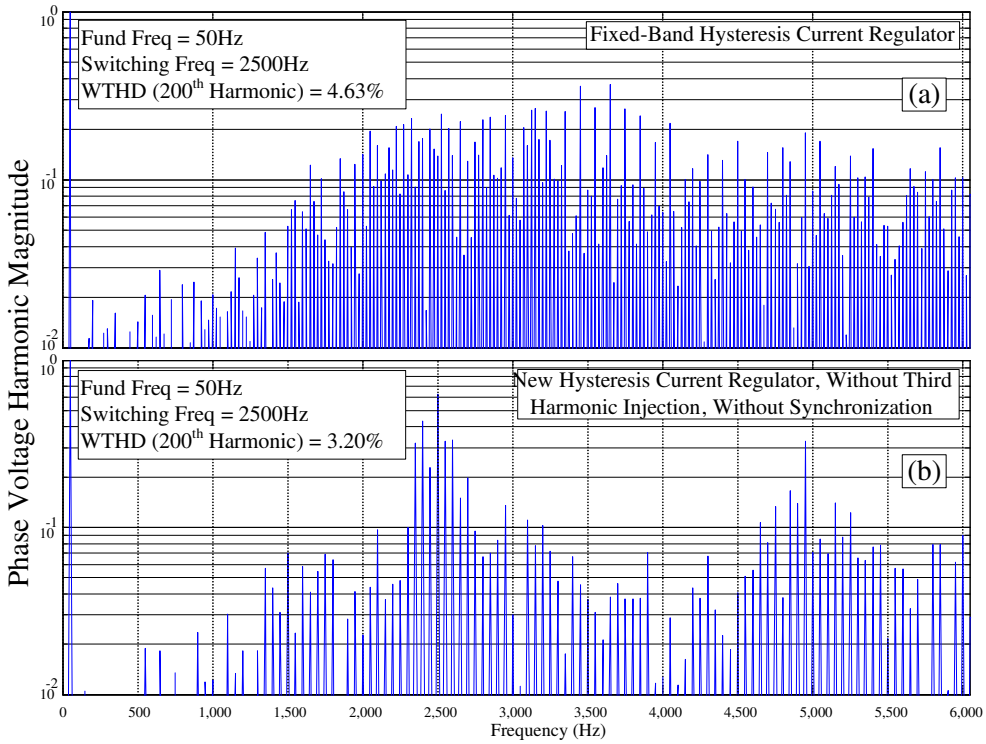


Figure 4.23: Phase leg voltage harmonic spectra (a) Fixed-band HCC (b) New variable band hysteresis current regulator - peak backemf = $0.9V_{DC}$

Simulation

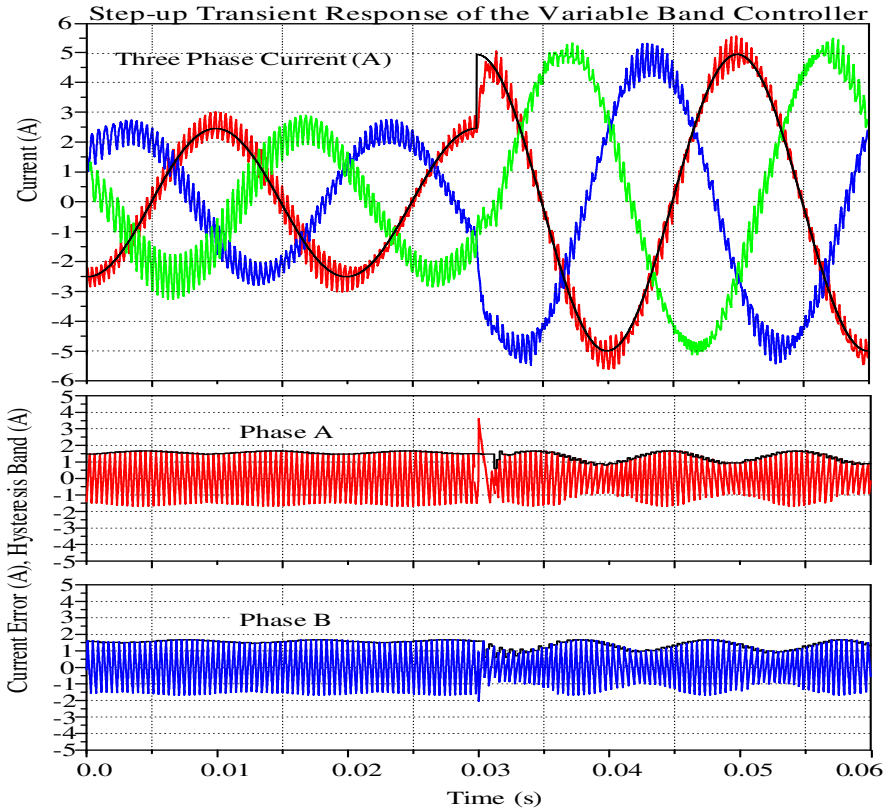


Figure 4.24: Transient step response of the proposed variable band with two hysteresis regulators – 100% commanded step change in current

Experiment

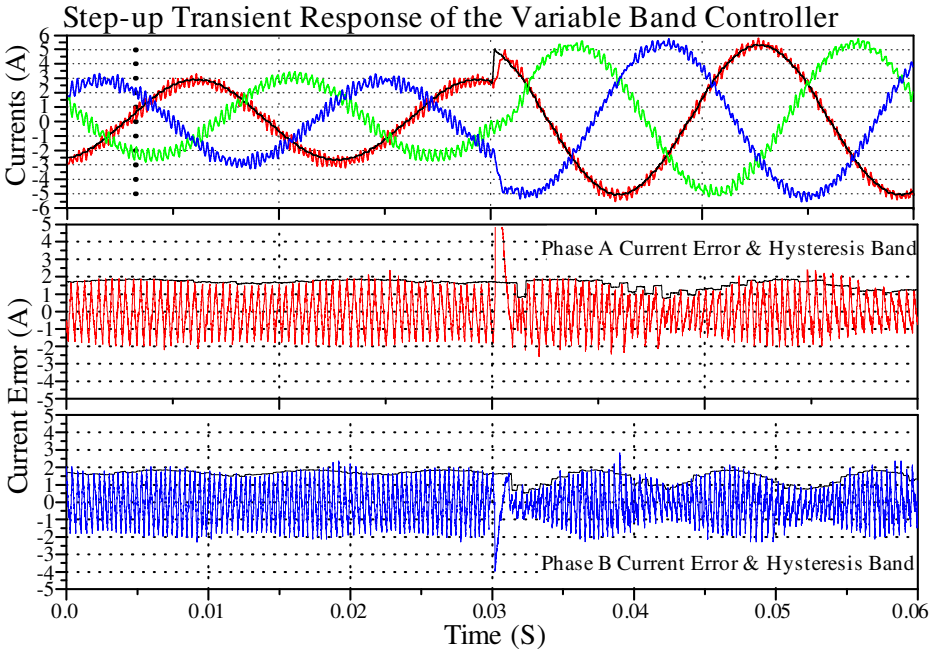


Figure 4.25: Transient step response of improved variable band two controller three phase hysteresis regulator – 100% commanded step change in current.

Figure 4.26 (simulation) and Figure 4.27 (experimental) show the result for the detailed current zero crossing synchronisation and centered zero state phase leg switching patterns, which almost exactly match the ideal CSVPWM patterns shown in Figure 4.7.

Figure 4.27 (experimental) also confirms how the synchronising algorithm ignores the potentially hazardous notches in the current error waveforms that occur just after the synchronized zero crossings as discussed in section 4.2.3. These notches are caused by the sampled update of the target reference current from the controlling DSP, and can cause spurious zero crossing events which can confuse a direct zero crossing detect circuit. The algorithm presented in this thesis avoids this hazard because of the way in which the current zero crossings are predicted by the measured phase leg switching events using equations (4.27) and (4.28).

Figure 4.28 shows the experimental result of the three-phase line-to-line voltages. The synchronisation of the phase leg A and B to the peak and trough of the triangle carrier used for the open-loop modulation of the phase leg C ensures to achieve clearly defined three-level line to line voltages. This further confirms that the performance of the new three-phase hysteresis current regulator is similar to the CSVM. Note in this figure that there are some minor pulse transitions between the adjacent voltage levels around the level transition region. This is caused by the minor 3% variation of the switching frequency as discussed in section 3.2.1. However these short pulses do not affect the harmonic performance of the HCC since they only carry a small amount of energy.

Figure 4.29 shows experimental results for the line to line harmonic spectra for both fixed band and the new hysteresis current regulator employing two hysteresis controllers. As expected, the WTHD of the line voltage spectra for the new hysteresis regulator is lower than for the conventional fixed band hysteresis current regulator. The figure also shows the grouped switching harmonics around the target switching frequency and the excellent harmonic cancellation that has been achieved at the target switching frequency of 2.5 kHz.

Simulation

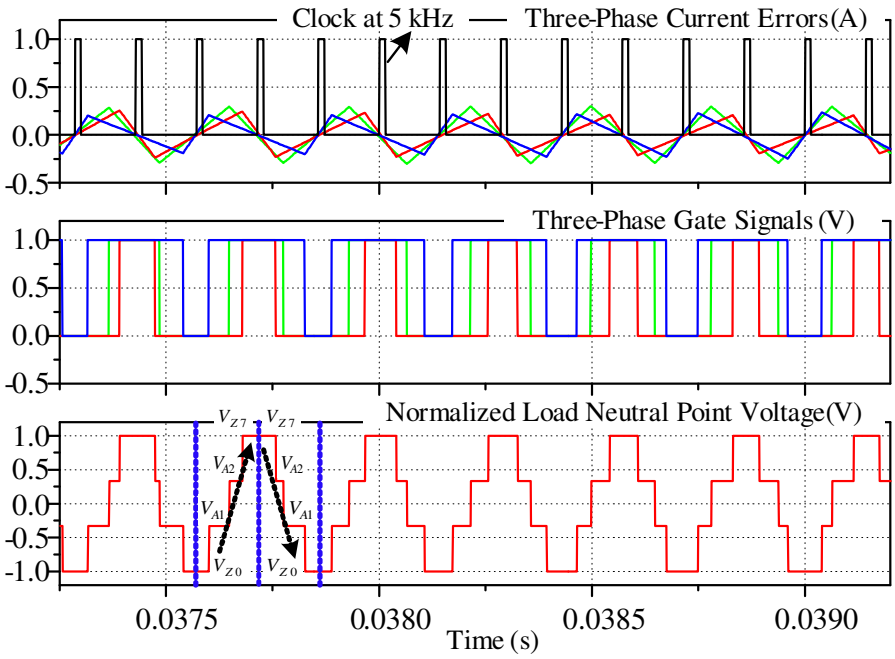


Figure 4.26: Detailed switching performance showing excellent frequency control and synchronized phase leg switching cycles.

Experiment

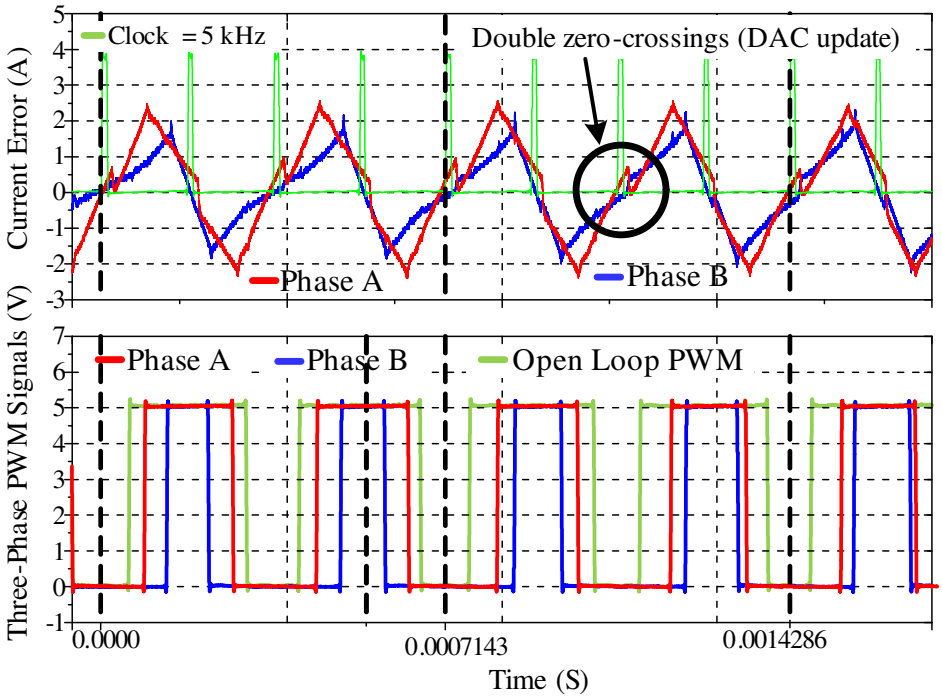


Figure 4.27: Detailed switching performance showing excellent frequency control and synchronized phase leg switching cycles

Experiment

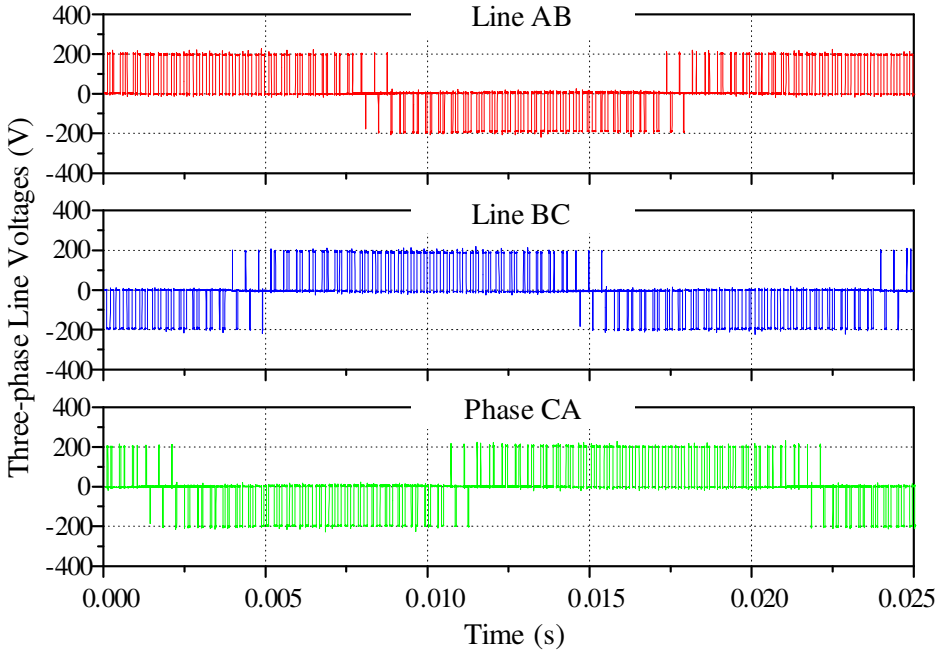


Figure 4.28: Three-level three-phase line to line voltages

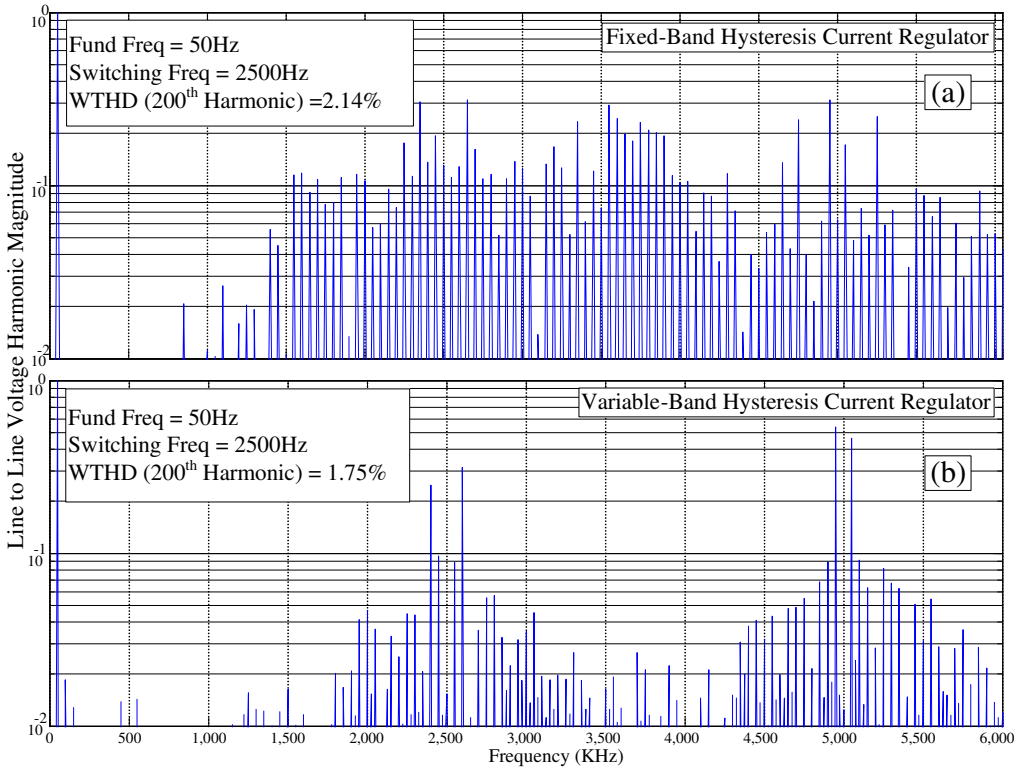


Figure 4.29: Line to line voltage harmonic performance of (a) conventional fixed band hysteresis (b) new variable band hysteresis current regulator with third harmonic injection, Modulation Depth = 0.8

Figure 4.30 (simulation) and Figure 4.31 (experimental) show the performance of the new HCC at a high modulation depth of 90% with third harmonic injection. Figure 4.30(a) and Figure 4.31(a) show the average phase-leg switched voltage and the third-harmonic compensation calculated by the DSP. These results clearly have the same well known double peak characteristic as CSVPWM, confirming that third harmonic compensation has been correctly integrated into the algorithm. Figure 4.30(b) and Figure 4.31(b) show the phase A current error and variable hysteresis band. These figures confirm that the regulator continues to operate well within the linear modulation region (as evidenced by the relatively small hysteresis band variation) with a modulation depth of over $0.9V_{DC}$. Additionally it can be confirmed from the A switched phase voltage shown in Figure 4.30(c) and Figure 4.31(c) that the regulator maintains a constant switching frequency .

Figure 4.32 (simulation) and Figure 4.33 (experimental) show the operation of the new hysteresis current regulator in the overmodulation region without third harmonic injection. These figures confirm the effectiveness of the variable band clamping strategy, where the regulators smoothly progress into overmodulation at the modulation depth of 110% without any sign of high frequency switching, while still maintaining controller stability within this region.

Figure 4.34 (experimental) shows the operation of the new hysteresis regulator for just before the controller enters the overmodulation region, at a modulation depth of 110% with third harmonic injection. The figure shows how the controller continues to switch the inverter at an almost constant switching frequency, which is only possible with correct third harmonic compensation so that the inverter remains in the linear modulation region.

Figure 4.35 shows a further simulation study for a modulation depth of 130% with third harmonic injection. The controller maintains excellent current control capability during overmodulation as the regulators smoothly progress into and out of overmodulation without any sign of high frequency switching or transient disturbance.

Simulation

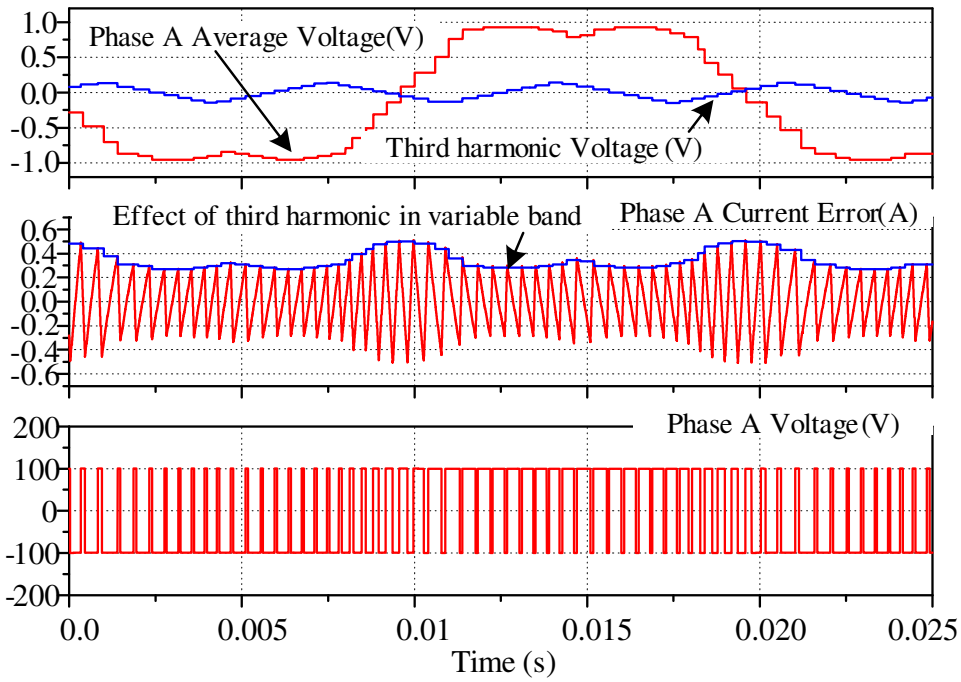


Figure 4.30: Average phase leg voltage with common mode third harmonic injection offset, Modulation Depth = 0.9

Experiment

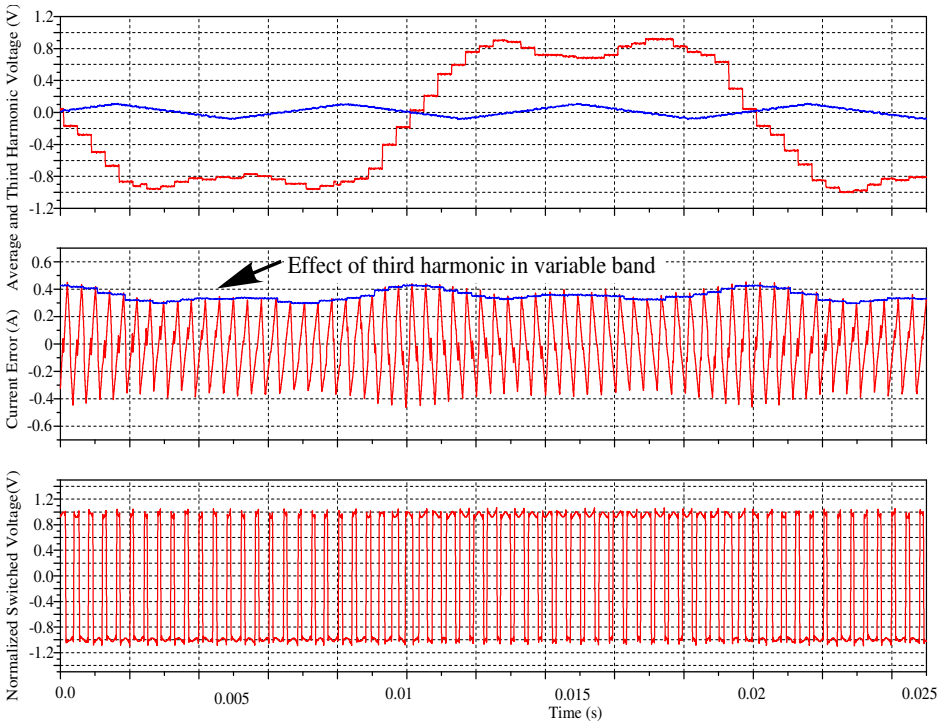


Figure 4.31: Average cycle phase leg voltage with common mode third harmonic injection offset, Modulation Depth = 0.9

Simulation

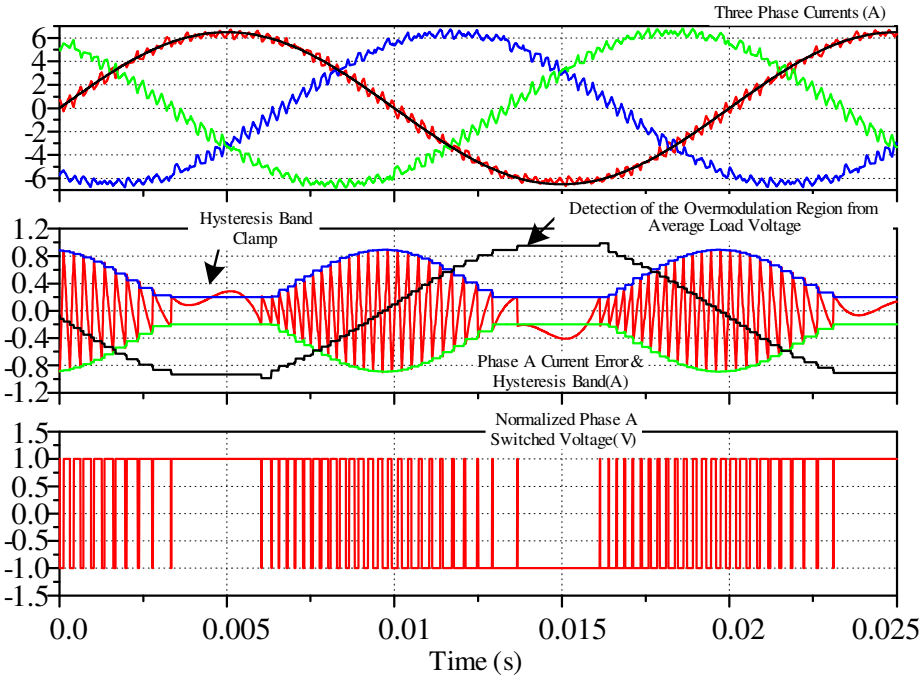


Figure 4.32: Simulated performance of variable band hysteresis current regulation with hysteresis band clamping and without third harmonic injection, peak average inverter output voltage = $1.1*V_{DC}$

Experiment

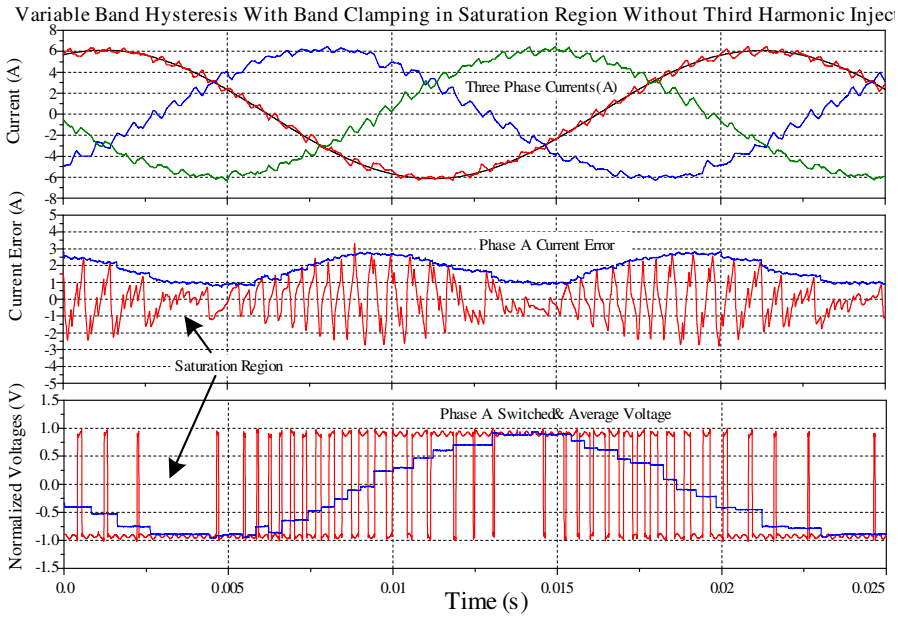


Figure 4.33: Experimental performance of variable band hysteresis current regulation with hysteresis band clamping and without third harmonic injection, peak average inverter output voltage = $1.1*V_{DC}$

Experiment

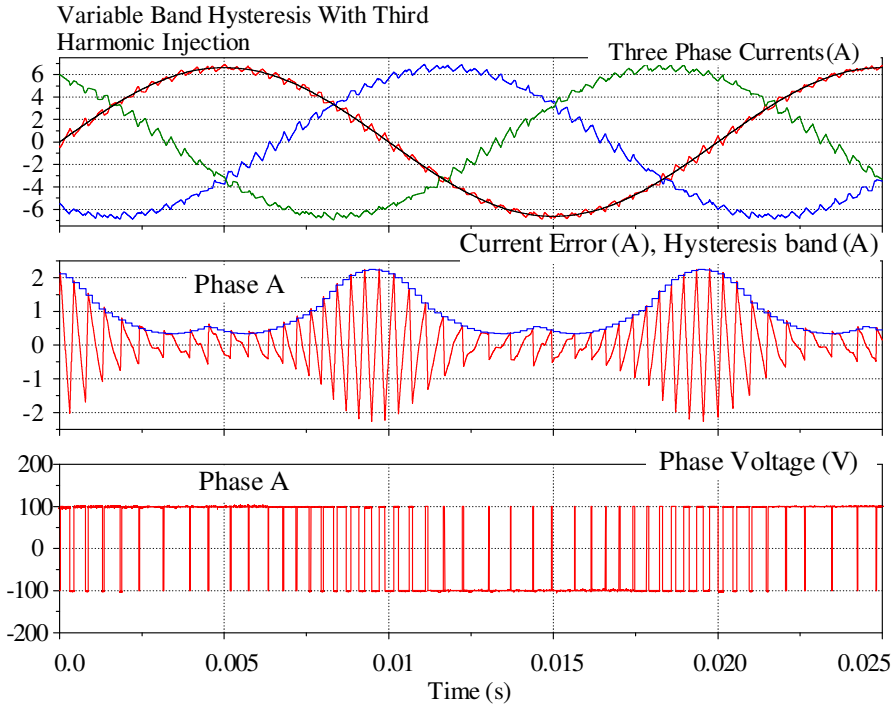


Figure 4.34 Experimental validation of variable band hysteresis current regulation in saturation region with third harmonic injection, Modulation Depth = 1.1.

Simulation

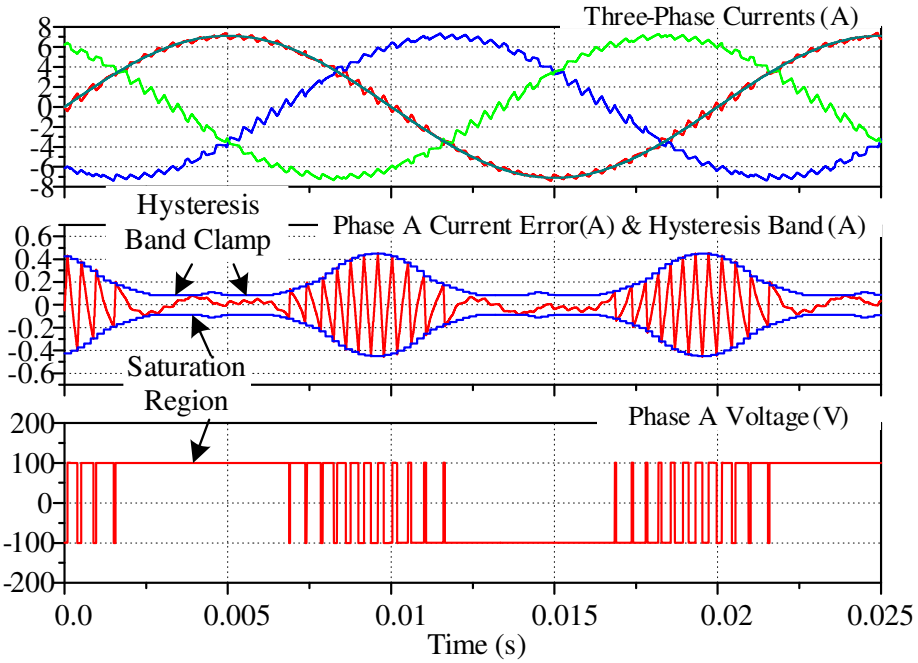


Figure 4.35: Simulation validation of variable band hysteresis current regulation in saturation region with third harmonic injection,
Modulation Depth = 1.3

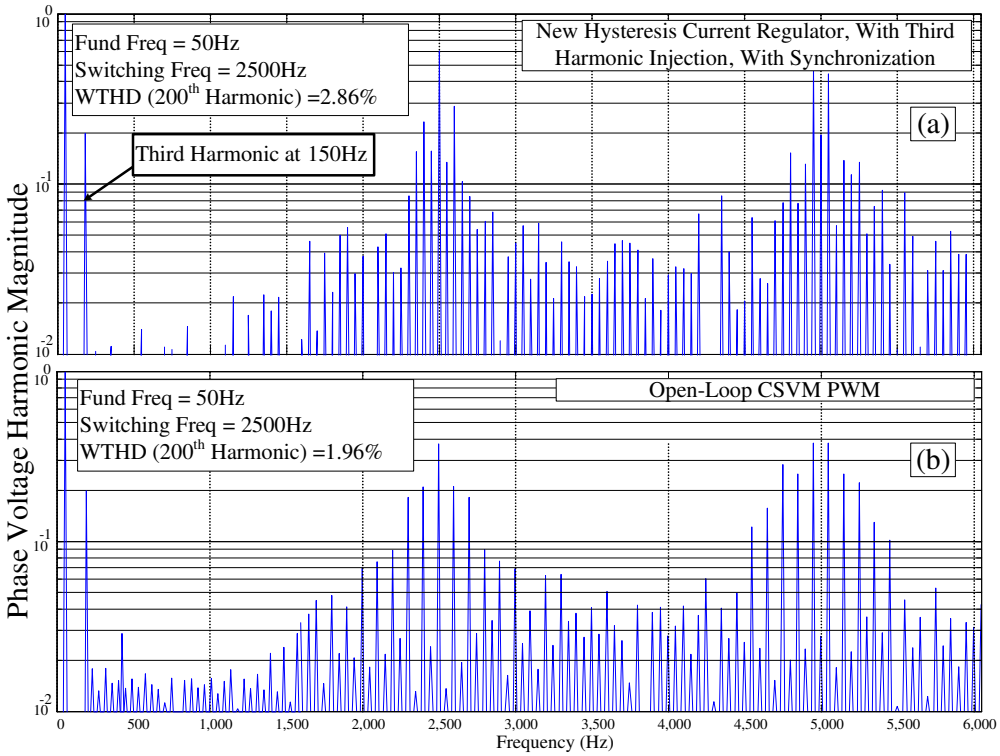


Figure 4.36: Phase leg voltage harmonic spectra (a) new HCC with common mode third harmonic injection, with current error synchronisation(b) open-loop CSVM PWM controller - peak backemf = $0.9V_{DC}$

Figure 4.36(a) provides experimental results for the harmonic performance of the new hysteresis current regulator with third harmonic injection and with synchronization of the phase leg current error zero-crossings. Figure 4.36(b) provides experimental results for the harmonic performance of an open-loop CSVM PWM controller. Comparing these two figures, it can be seen that the proposed HCC has achieved a target harmonic performance that is very close to the CSVM performance, with a significant carrier at the target switching frequency of 2500Hz, the clear sideband components and the expected third harmonic voltage baseband harmonic at 150Hz. Note that in Figure 4.36(a) there are some additional minor side band harmonics around the major first and second carrier harmonics, which are primarily caused by the 2% variation in switching frequency as discussed in section 3.3.

Also Figure 4.36(a) confirms that by injecting the third harmonic voltage together with the synchronization of the phases A and B current error zero-crossings, weighted total harmonic distortion is significantly reduced from 3.20% in Figure 4.23(b) to 2.86% in Figure 4.36(a).

4.7 Summary

This chapter has presented a new approach for hysteresis current regulation of a two-level three-phase VSI to achieve a performance that is similar to open-loop CSVM.

The development of the new two-level three-phase hysteresis current regulator began by extending the new single-phase HCC to a two-level three-phase VSI, separately measuring the three-phase average inverter output voltages and varying the per phase hysteresis bands to maintain a constant switching frequency. The common mode interacting current was then calculated from the three-phase switching commands using a simple summing-integrating circuit to avoid additional voltage measurement circuits.

The three-phase current error zero-crossings were then synchronized to a fixed reference clock to ensure that the VSI selected the three nearest space vectors and thus achieved a harmonic performance similar to open-loop CSVM.

Next, the third phase current regulator was replaced with a fixed frequency directly modulated phase leg, commanded with the inverse sum of the average output voltages of the two current regulated phase legs. This reduced the order of the system to two, to avoid interactions between over-constrained regulators.

The linear modulating range of the new hysteresis regulator was then extended by calculating the third harmonic voltage terms from the information of three-phase average inverter output voltages. This voltage was incorporated into the calculation of the common mode interacting current in a similar way as the common mode neutral point voltage.

Operation of the controller was implemented in the overmodulation region by clamping the phases A and B variable hysteresis bands as described in chapter 3. The overmodulation operation was verified both with and without third harmonic injection to ensure that controller maintained stability in the non-linear region.

Simulation and experimental results for this system have been provided to confirm the operation of the new hysteresis current regulation approach when applied to a two-level three phase VSI.

Chapter 5

Hysteresis Current Regulation of a Three-Level Single-Phase Voltage Source Inverters

This chapter[1] presents a new approach for hysteresis current regulation of three-level single-phase VSIs such as a single-phase H-bridge, and for a single-phase leg NPC and FC inverter. The major research extensions are:

- Section 5.2 applies the fundamental concepts of the new two-level HCC presented in chapter 3, to develop a new variable band HCC for the three-level VSIs. The resultant harmonic performance is similar to asymmetrical double-edge PWM for a H-bridge and LS PWM for the NPC and FC VSIs.
- Section 5.3 presents a novel strategy to detect the impending polarity change of a three-level switched voltage from the zero-crossing information of the average voltage, without requiring multiple hysteresis bands and/or time information of the current error zero-crossing. This means only one hysteresis comparator is required per phase leg while still eliminating the DC steady-state tracking error.
- Section 5.4 develops sequential logic to decode the hysteresis switching signals to generate gate signals that are similar to open-loop double edge asymmetrical PWM for the H-bridge, LS PWM for the NPC phase leg and PS PWM for the FC phase leg. For the FC, a finite state machine is

[1] **Materials in this chapter were first published as**

1. Davoodnezhad, R.; Holmes, D. G.; McGrath, B. P.; "Three-Level Hysteresis Current Regulation for a Three Phase Neutral Point Clamped Inverter" *Power Electronics and Motion Control Conference (ECCE-EPE 2012*).
2. Davoodnezhad, R.; Holmes, D. G.; McGrath, B. P.; "Hysteresis current regulation of three phase flying capacitor inverter with balanced capacitor voltages," Power Electronics and Motion Control Conference (IPEMC 2012).
3. Davoodnezhad, R.; Holmes, D. G.; McGrath, B. P.; "A Three-Level Self-Synchronizing Hysteresis Current Regulator with Constant Switching Frequency" *Energy Conversion Conference (ECCEAsia DownUnder 2013).*

presented to utilize the VSI zero states to maintain a balanced FC voltage without requiring additional voltage sensors.

- Section 5.5 extends the operation of the proposed HCC into the overmodulation region in a similar way as described in section 3.4.

Simulation and experimental results are presented at the end of this chapter.

5.1 Switching Process of a Three-Level Voltage Source Inverter

Figure 5.1 shows the topologies of a three-level single-phase H-bridge, and a single phase leg of a neutral point clamped and flying capacitor inverter. The single-phase H-bridge inverter is constructed by connecting two two-level half bridges across a common DC bus voltage. The three-level neutral point clamped (NPC) inverter is formed by stacking two two-level half bridges in series, with the phase outputs of the lower and the upper half bridges constrained by two series clamping diodes. The topology of the flying capacitor (FC) inverter is similar to the NPC VSI with the clamping diodes replaced by a flying capacitor, charged to half the overall DC bus voltage.

For all of the VSI topologies shown in Figure 5.1, the switched output voltage has three separate states of $V(t) = 0, \pm V_{DC}$. Details of the sequential logic to generate the switching combinations that are required to generate these inverter states are discussed in section 5.4.

Figure 5.2 shows in principle the performance of such a three-level voltage source inverter controlled using three-level fixed frequency open-loop PWM. Examination

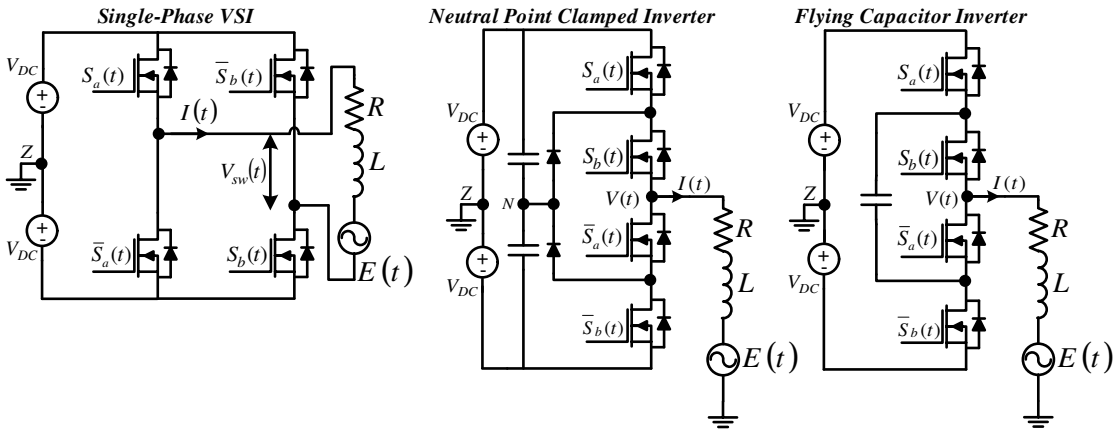


Figure 5.1: Overall block diagram of a three-level HCC for a single-phase H-bridge, and a single-phase leg NPC and FC inverter feeding a back-emf type load with series RL load

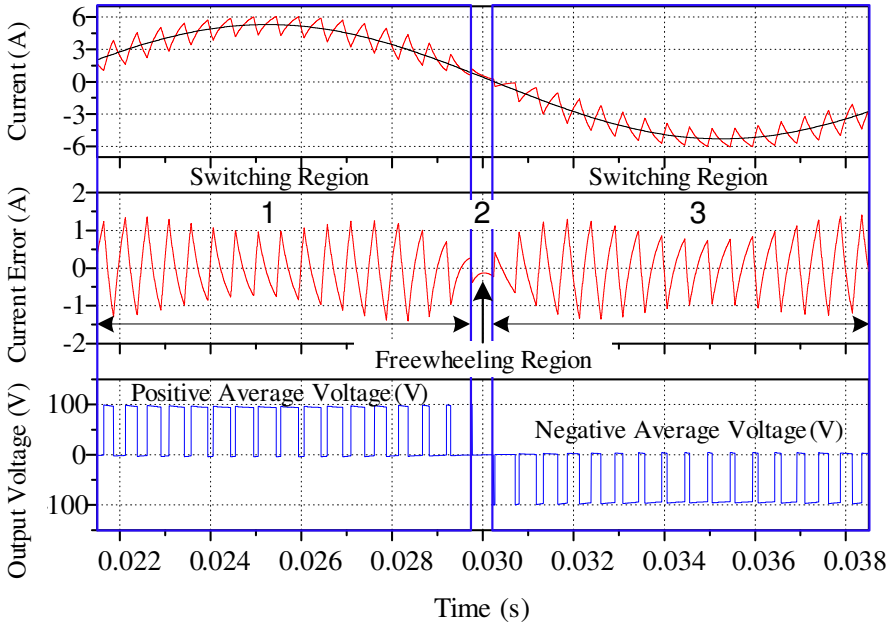


Figure 5.2: Basic principles of a three-level switching process (a) output current (A) (b) current error (A) (c) three-level output switched voltage (V)

of Figure 5.2(b)(c) shows that the switching process of such an inverter is divided into three regions, as follows:

- Region 1: The average inverter voltage is positive and the inverter switches between the positive active and zero states $(0, +V_{DC})$ with a constant switching frequency. For a hysteresis current controlled VSI, these switching states are generated when the current error reaches the upper or the lower (variable) hysteresis bands.
- Region 2: This is a transition region (level change region) where the average output voltage changes its polarity from a positive value to a negative value. As a result the phase voltage switching groups change from $(0, +V_{DC})$ to $(0, -V_{DC})$. This transition causes the current error to freewheel during this region. In a hysteresis current controlled VSI, a polarity command is required in this region to force the inverter to change its output voltage polarity.
- Region 3: The average inverter voltage is negative and the inverter switches between the negative active and zero states $(0, -V_{DC})$ with a constant switching frequency. In a hysteresis current controlled VSI, these

switching states are generated when the current error reaches the upper or the lower (variable) hysteresis bands.

From this analysis, it can be concluded that the new hysteresis current regulator must achieve the following target objectives:

- Variable hysteresis band operation during the switching regions (regions 1 and 3) to maintain constant switching frequency.
- Detect the point of the inverter level change and generate a polarity change command.
- Decode the hysteresis switching signals using the polarity command signal to generate target device gate signals for three-level modulation of the VSI.

Each of these objectives will now be discussed in in detail.

5.2 Three-level Constant Frequency Hysteresis Current Regulation Using Average Voltage

This section applies the fundamental concepts of the new two-level HCC that has been developed in chapter 3 to a three-level voltage source inverter, as follows:

- Vary the hysteresis band using the average voltage of a three-level switched phase voltage to maintain constant switching frequency.
- Synchronously measure the average inverter output voltage from the switching transition times of the phase-leg switching signal.
- Synchronise the phase-leg current error zero-crossings to a fixed reference clock by adding an extra band adjustment to the variable hysteresis band to further improve the frequency regulation.

The developments presented in this section are for switching regions 1 and 3 where the current error is switching between the upper and the lower hysteresis bands.

5.2.1 Three-level Hysteresis Band Variation Using Average Inverter Voltage

5.2.1.1 Mathematical Developments of the Variable Hysteresis Band

As shown in Figure 5.1, the three-level voltage source inverters feed into a series resistive and inductive load/filter with a series AC back-EMF. Irrespective of the particular VSI topology, the general inverter load relationship for this arrangement can be written as:

$$V(t) = RI(t) + L\frac{dI(t)}{dt} + E(t) \tag{5.1}$$

Equation (5.1) is identical to the load relationship of a two level VSI as per equation (3.1), except that the switched phase voltage $V(t)$ can now have the values of 0 and $\pm V_{DC}$.

Assuming that the voltage across the series resistance is small compared to both the back-emf and the voltage across series inductor voltage, equation (5.1) simplifies to:

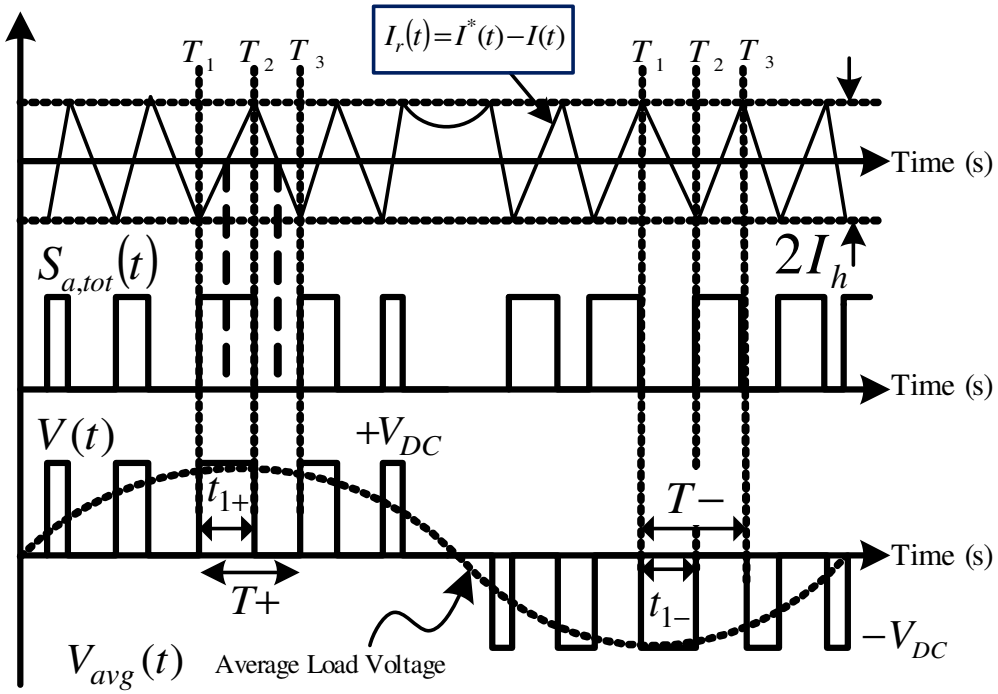


Figure 5.3: Basic principles of a three-level hysteresis current regulation process showing the phase current error, hysteresis switching signals, three-level switched output voltage and its fundamental component

$$V\ (t) = L\frac{dI(t)}{dt} + E(t) \tag{5.2}$$

This equation is the starting point for the three-level hysteresis current regulation strategies presented in this chapter.

Figure 5.3 shows the three-level hysteresis switching process over one fundamental cycle of the average output voltage. In this figure, the hysteresis bands are fixed and the switching frequency is set low, to better show the switching processes of the hysteresis regulator. From this figure, when the average inverter output voltage is positive, if the output current error exceeds the upper hysteresis band, the switching signal $S_{tot}(t)$ becomes low and the phase voltage $V(t)$ switches to a ZERO volt state. When the output current error exceeds the lower hysteresis band, the switching signal $S_{tot}(t)$ becomes high and hence the phase voltage switches to the upper DC bus.

When the average inverter output voltage is negative, if the output current error exceeds the upper hysteresis band, the switching signal $S_{tot}(t)$ becomes low and the switched phase voltage switches to the lower DC bus. When the output current error exceeds the lower hysteresis band, the switching signal $S_{tot}(t)$ becomes high and hence the switched phase voltage switches to ZERO volt state.

As described in chapter 3, the output load current can be further split into the fundamental component $I_f(t)$ and the switching ripple component $I_r(t)$. Now using the same mathematical approach that was presented in section 3.2.1, and from Figure 5.3, the behaviour of the switching ripple component can be analysed into two time intervals over one switching period during the positive and the negative average inverter voltage regions, as follows:

1) Average inverter output voltage is positive $V_{avg}(t) > 0$:

During the switching period ON-time (t_{1+}): The current ramps from the lower hysteresis band $-I_h$ to the upper hysteresis band $+I_h$ and the switched phase leg voltage is $+V_{DC}$. Assuming that average voltage component of the phase leg output is

constant over one switching cycle $V_{avg}(t) = V_{avg}$, the switching time interval t_{1+} can be expressed as:

$$t_{1+} = \frac{2LI_h}{V_{DC} - V_{avg}} \tag{5.3}$$

During the switching period OFF-time ($T_+ - t_{1+}$): The current ramps from the upper hysteresis band $+I_h$ to the lower hysteresis band $-I_h$ and the switched phase leg voltage is 0. The switching time interval $T_+ - t_{1+}$ can be expressed as:

$$T_+ - t_{1+} = \frac{-2LI_h}{0 - V_{avg}} \tag{5.4}$$

Under the assumption that the load average voltage V_{avg} is constant over the positive switching period, combining equation (5.3) and equation (5.4) and solving them for the positive switching period T_+ gives:

$$T_+ = \frac{2V_{DC}LI_h}{V_{avg}\left(V_{DC} - V_{avg}\right)} \tag{5.5}$$

2) Average inverter output voltage is negative $V_{avg}(t) < 0$:

During the switching period ON-time (t_{1-}): The current ramps from the upper hysteresis band $+I_h$ to the lower hysteresis band $-I_h$ and the switched phase leg voltage is $-V_{DC}$. Assuming that average voltage component of the phase leg output is constant over one switching cycle $V_{avg}(t) = V_{avg}$, the switching time interval t_{1-} can be expressed as:

$$t_{1-} = \frac{2LI_h}{V_{DC} + V_{avg}} \tag{5.6}$$

During the switching period OFF-time ($T_- - t_{1-}$): The current ramps from the lower hysteresis band $-I_h$ to the upper hysteresis band $+I_h$ and the switched phase leg voltage is 0. The switching time interval $T_- - t_{1-}$ can be expressed as:

$$T_- - t_{1-} = \frac{2LI_h}{0 - V_{avg}} \tag{5.7}$$

Continuing the assumption that the load average voltage V_{avg} is constant over the negative switching period, combining equation (5.6) and equation (5.7) and solving them for the negative switching period T_- gives:

$$T_- = \frac{2V_{DC}LI_h}{-V_{avg}\left(V_{DC} + V_{avg}\right)} \tag{5.8}$$

Equations (5.5) and (5.8) can be rearranged into essentially identical expressions for the switching frequency during the situations when the average inverter output voltage is either positive or negative, as:

$$f_{sw,+} = \frac{V_{avg}\left(V_{DC} - V_{avg}\right)}{2V_{DC}LI_h} \qquad V_{avg} > 0 \tag{5.9}$$

$$f_{sw,-} = \frac{-V_{avg}\left(V_{DC} + V_{avg}\right)}{2V_{DC}LI_h} \qquad V_{avg} < 0 \tag{5.10}$$

The parameters of equations (5.9) and (5.10) including the filter/load inductance, the hysteresis band and the DC bus voltage are constant, except for the average inverter output voltage (which varies over each fundamental cycle). Figure 5.4 shows the variation of switching frequency over a fundamental period (50Hz) as per equations (5.9) and (5.10), together with the average load voltage, AC back-emf and the load inductance voltage. This figure confirms the theoretical expectations, showing how the switching frequency varies in accordance with the average inverter output voltage. Comparing Figure 5.4 with the two-level switching frequency variation as shown in Figure 3.4, the following differences can be seen:

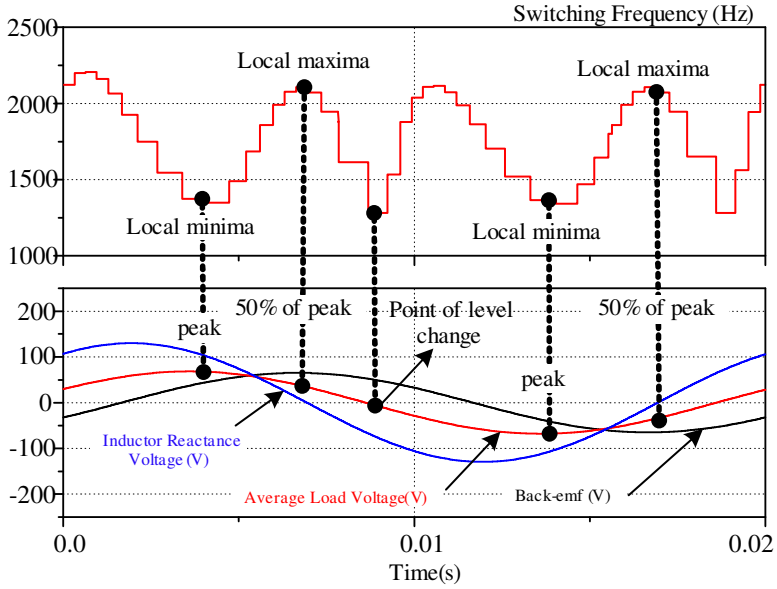


Figure 5.4: (Top) Variation in the switching frequency for a three-level VSI (Bottom) average load voltage, ac load back-emf and the inductor reactance voltage

- For both two-level and three-level VSIs, the local minima occurs when the load average voltage is at its maximum value.
- For the three-level VSI, the local maxima occurs when the load average voltage is at 50% of its peak value. For the two-level VSI the local maxima occurs exactly at the time where the load average voltage is at zero.
- For the three-level VSI, when the average voltage is zero, the output switched voltage is at the point of polarity change and no switching events happen. This can be further confirmed by setting $V_{avg} = 0$ in either equation (5.9) or (5.10), which results in a switching frequency of zero.

Now rearranging equations (5.9) and (5.10) and solving them for the time-varying hysteresis band I_h, gives:

$$I_{h,+}(t) = \frac{V_{avg}\left(V_{DC} - V_{avg}\right)}{2V_{DC}Lf_{sw,+}} \qquad V_{avg} > 0 \tag{5.11}$$

$$I_{h,-}(t) = \frac{-V_{avg}\left(V_{DC} + V_{avg}\right)}{2V_{DC}Lf_{sw,-}} \qquad V_{avg} < 0 \tag{5.12}$$

where $I_{h,+}$ and $I_{h,-}$ show how the hysteresis bands vary during the positive and negative average inverter output voltages respectively. Since the switching frequency during both the positive and the negative half fundamental cycles must be kept constant $f_{sw,+} = f_{sw,-} = f_{sw}$, the three-level variable hysteresis band expressions in (5.11) and (5.12) can now be combined into one form as $V_{avg}(t)$ and $I_f(t)$ change, i.e.:

$$I_h = I_{h\_\max}\left|V_{avg}(t)/V_{DC}\right|\left(1-\left|V_{avg}(t)/V_{DC}\right|\right) \tag{5.13}$$

where $I_{h,\max} = \dfrac{V_{DC}}{2Lf_{sw}}$ is the maximum allowable hysteresis band. (5.14)

Comparing equation (5.13) with the two-level variable hysteresis band expression:

$$I_h(t) = I_{h,\max}\left(1-\left(V_{avg}(t)/V_{DC}\right)^2\right) \qquad I_{h,\max} = \frac{V_{DC}}{4Lf_{sw}} \tag{5.15}$$

the following differences and similarities can be identified:

- Both variable hysteresis band expressions are a function of the average inverter output voltage and the constant system parameters.
- The minimum value of the two-level and three-level variable hysteresis band expressions occurs when the average inverter voltage is at its maximum, viz: $V_{avg}(t) = V_{DC}$
- Two-level variable hysteresis bands vary over one full fundamental cycle of the average inverter output voltage while the three-level variable hysteresis band varies over a half fundamental cycle.
- When the average voltage is zero, the two-level hysteresis band is at the maximum allowable hysteresis band while this occurs for the three-level hysteresis band in the freewheeling region (region 2 in Figure 5.2(b)).

Figure 5.5 shows the relationships between the three-level variable hysteresis band, and the average inverter output voltage over one target fundamental cycle. From this figure, the variable hysteresis band can be seen to be a combination of two

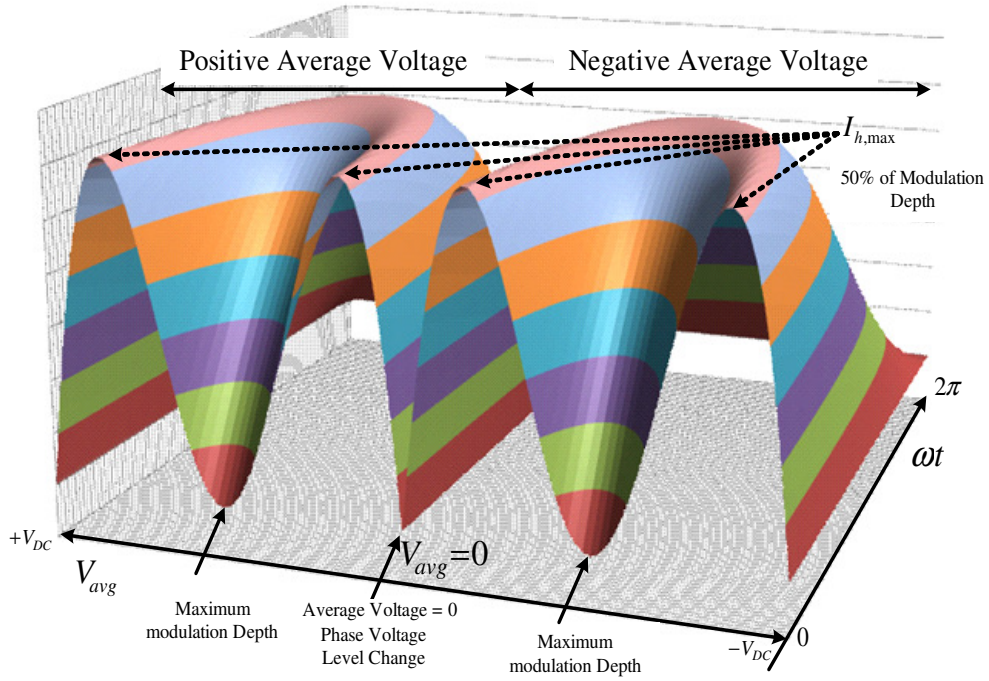


Figure 5.5: Three-dimensional diagram of a three-level variable hysteresis band with respect to the variation in average inverter output voltage and time

mirrored components, one belonging to the period when the average voltage is positive and the other one belonging to the period when the average voltage is negative. Also, from this result, as the average inverter output voltage magnitude increases, the hysteresis band changes significantly with its maximum at 50% of the full modulation depth and its minimum at the full modulation depth. At the point of the switched phase voltage level change, the average inverter voltage is zero and hence from equation (5.13) the three-level variable hysteresis band is zero.

Figure 5.6 compares the performance of the new approach using the average inverter output voltage, with that of a fixed-band hysteresis current regulator. To conduct this comparison, a sinusoidal waveform is used as a current reference, which is synchronised with the phase average output voltage. As expected, using a fixed hysteresis band imposes a variation of 30% on the desired average switching frequency of 2500 Hz while using a variable hysteresis band that varies according to the average output voltage significantly reduces this variation to less than 1% of the target switching frequency. Also note from this figure, how the per phase current error stops switching at the point when the variable hysteresis band reaches zero.

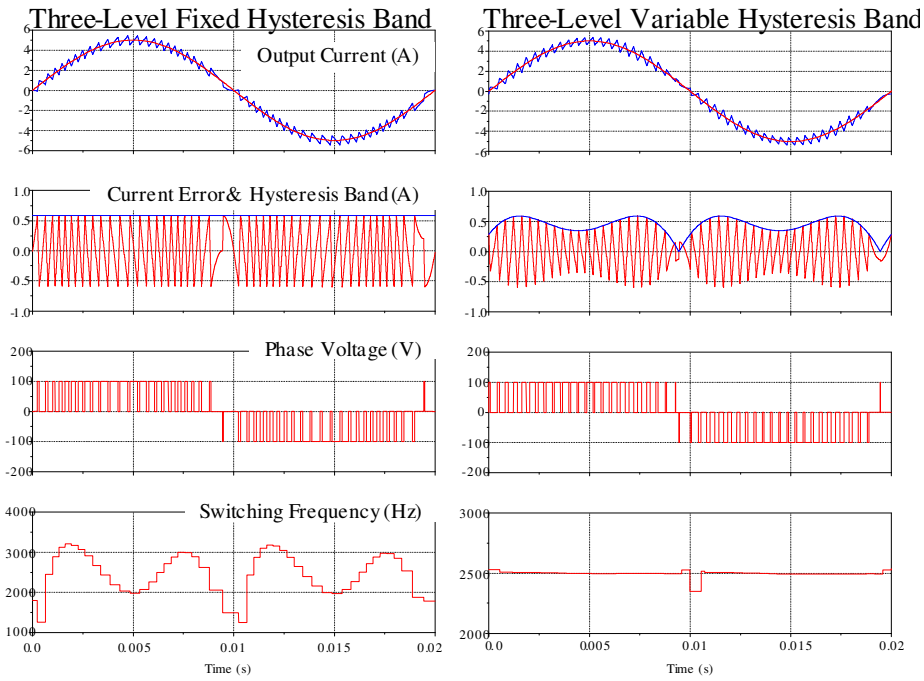


Figure 5.6: Comparison of between the performance of a conventional fixed hysteresis band and proposed variable hysteresis band - target switching frequency $f_{sw} = 2500\ \ Hz$, modulation depth = 0.9

5.2.2 Measuring the Average Load Voltage for a Three-Level Inverter

From the previous section, equation (5.13) defines how the variable hysteresis band uses the knowledge of the average inverter output voltage to maintain a constant switching frequency. Section 3.2.2 has developed a novel technique to synchronously measure the average inverter output voltage by capturing the transition times of the phase leg switching events using a DSP capture/timer port. For a two-level inverter, the output of the hysteresis comparator represents the normalized average inverter voltage, which is used to extract its fundamental component for the variable hysteresis band calculation using equation (3.19).

For a three-level VSI, the absolute value term $\left|V_{avg}\left(t\right)\right|$ must be determined for the variable hysteresis band calculation using equation (5.13). Hence, the output of the hysteresis comparator must be modified before extracting its fundamental component.

Figure 5.7(a) shows the output of the hysteresis comparator (total hysteresis switching signal) $S_{tot}\left(t\right)$ generated by comparing the phase leg current error and the hysteresis band as presented in section 5.4. Figure 5.7(b) shows the fundamental

component of this hysteresis switching signal $S_{tot,avg}(t)$, the actual $V_{avg}(t)$ and the absolute average inverter output voltages $|V_{avg}(t)|$. Comparing these three average voltages the following relationship can be concluded:

- During the time when the average voltage is positive, the fundamental component of $S_{tot}(t)$ is equal to the normalized actual average inverter output voltage.
- During the time when the average voltage is negative, the fundamental component of $S_{tot}(t)$ is equal to the normalized average inverter output voltage level shifted by 100%.

This occurs because during the positive average voltage, the inverter active state occurs when the output of the hysteresis comparator is logic ONE, and the inverter zero state occurs when the output of the hysteresis comparator is logic ZERO. During the negative average voltage, the inverter active state occurs when the output of the hysteresis comparator is logic ZERO, and the inverter zero state occurs when the output of the hysteresis comparator is logic ONE.

Hence, the hysteresis switching signal $S_{tot}(t)$ must be modified to become ONE for an active switched output of either polarity ($+V_{DC}$ or $-V_{DC}$), and ZERO for a

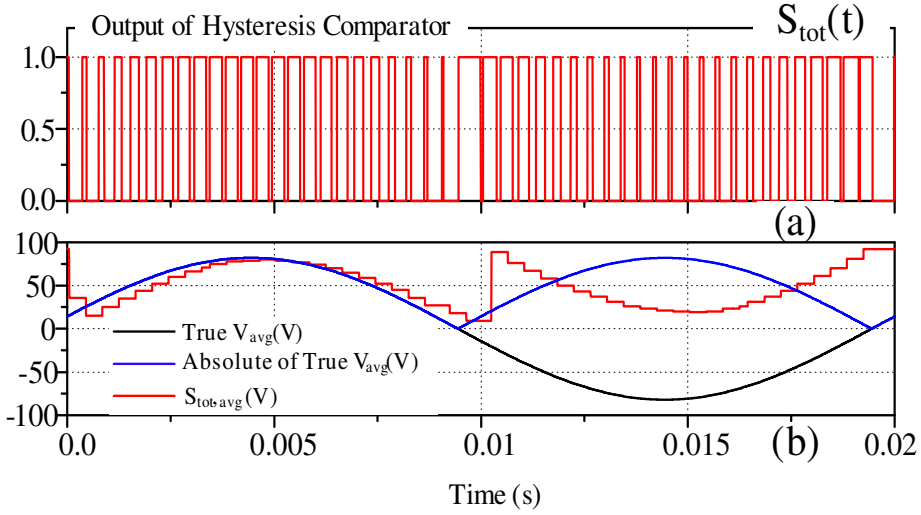


Figure 5.7: (a) Output of hysteresis comparator $S_{tot}(t)$ (b) true (virtual) average voltage, absolute value of true average voltage and extracted fundamental component of the hysteresis switching

0V phase leg output. Figure 5.8(a) shows this modified hysteresis switching signal $S_{OR}(t)$, which equals the hysteresis switching signal $S_{tot}(t)$ when the average voltage is positive and equals the complement of the hysteresis switching signal $\overline{S}_{tot}(t)$ when the average voltage is negative.

Figure 5.8(b) shows the fundamental component of this modified hysteresis switching signal $S_{OR,avg}(t)$ which now equals the absolute normalized average inverter output voltage as follows:

$$S_{OR,avg}(t) = \left(\frac{T_2 - T_1}{T_3 - T_1} \right) \tag{5.16}$$

$$\left| V_{avg}(t) \right| = 2V_{DC} * S_{OR,avg}(t) \tag{5.17}$$

where T_1, T_2, T_3 are the switching instances in Figure 5.3.

As discussed in section 3.2.3, this average voltage can again be linearly extrapolated to compensate for the effect of sampling delay, to achieve a close approximation to the actual normalized average inverter voltage.

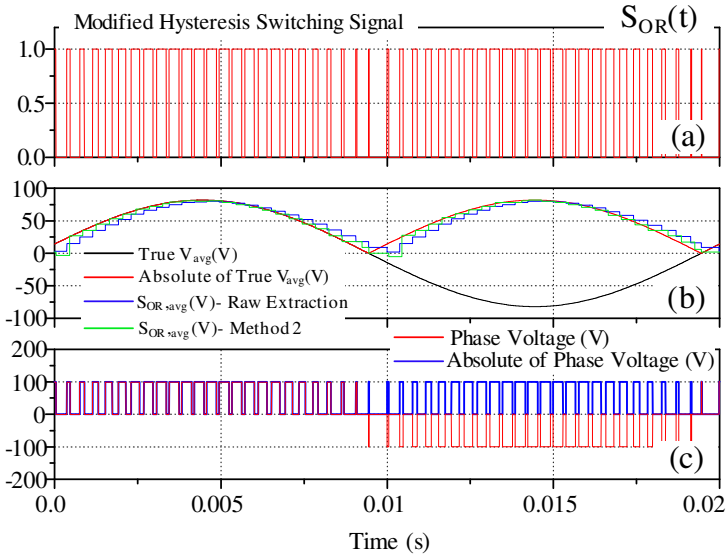


Figure 5.8: (a) Modified hysteresis switching signal $S_{OR}(t)$ (b) true (virtual) average voltage, absolute value of true average voltage, raw extraction of fundamental component of the modified hysteresis switching signal, linear extrapolated of extracted fundamental component (c) three-level phase voltage and absolute value phase voltage

5.2.3 Synchronisation to a Fixed Reference Clock

Section 3.3 has presented an additional technique to better synchronize the current error zero-crossings to a fixed reference clock. This synchronisation is implemented as an additional feedforward step without directly measuring the current error zero-crossings. The same concept is now applied to the three-level voltage source inverter to achieve the following advantages:

- For a three-level single-phase H-bridge, this synchronization ensures a harmonic performance similar to that of open-loop double edge asymmetrical sampled PWM, which is known to achieve optimum harmonic performance [11].
- For the NPC and FC inverters, this synchronization ensures a harmonic performance similar to the harmonically superior open-loop phase disposition (PD) PWM [11][24][34].

Figure 5.9 illustrates the synchronisation process for a three-level switching process, showing the actual and the expected phase current error during the switching regions (regions 1 and 3) and freewheel region (region 2).

5.2.3.1 During the switching regions 1 and 3

Section 3.3 presented a technique to calculate the current error zero-crossing times from the switching time information of the gate signals using equations (3.27) and (3.28). To make this calculation, the current error needs to be switching and reaching the upper or the lower hysteresis bands, which only occurs in the switching regions 1 and 3 as shown in Figure 5.9.

From Figure 5.9, and using similar triangles before the current error starts freewheeling, the band offset required to correct the time error within the first half switching cycle is: (from section 3.3.2)

$$\delta I_h = -I_h * 2f_{sw} * (\delta t + t_{db}) \tag{5.18}$$

Hence, the new calculated three-level variable hysteresis band is:

$$I_{h,new} = \delta I_h + I_{h,old} \tag{5.19}$$

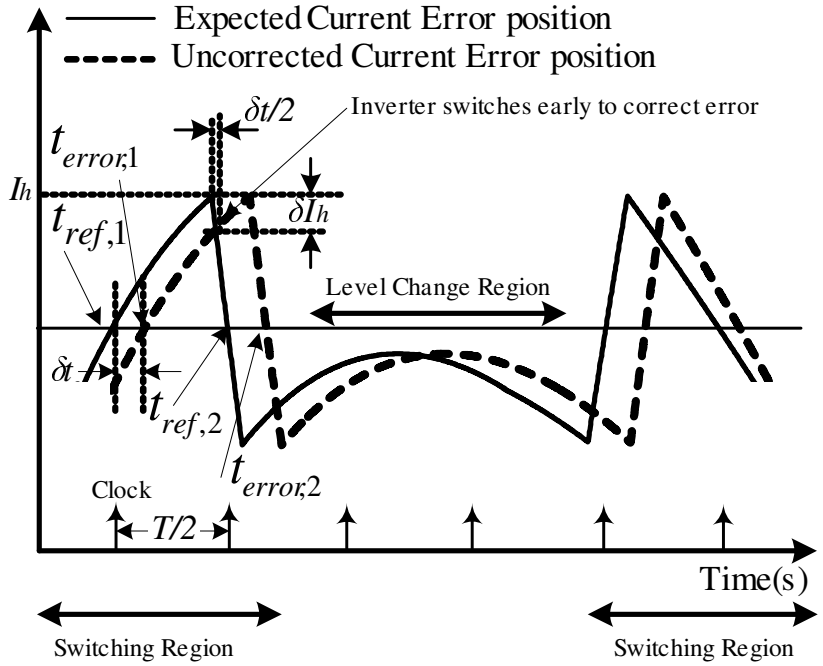


Figure 5.9: Identification of current error zero crossing timing errors for a three-level switching process.

where $I_{h,old}$ is the three-level variable hysteresis band magnitude calculated from equation (5.13).

5.2.3.2 During the level change region 2

Figure 5.9 also shows the behaviour of the current error during the level change region. During this region, the current error starts freewheeling without reaching either the lower or the upper hysteresis bands. Hence, no switching occurs during this region. Consequently the controller performs the following steps:

- Calculation of the variable hysteresis band is suspended and the new variable hysteresis band is kept equal to the hysteresis band calculated before the inverter enters the level change region.
- The synchronization process is suspended and as a result no further band offset is applied to the previously calculated variable hysteresis band.

Once the phase current error starts switching again by reaching the lower or the upper hysteresis bands, the controller resumes the variable hysteresis band calculation and the synchronisation process continues as presented in section 5.2.3.1.

5.2.3.3 Experimental Confirmation of the Synchronization Process

Figure 5.10 experimentally confirms the excellent synchronization achieved by this process. From this figure, synchronization is applied during the hysteresis switching process either side of the level change region. In this way, the controller ensures that the synchronization process and the level change detection algorithms stay independent of each other. The controller maintains stable operation during the current error freewheeling region, since the synchronization process is independent of the direct zero-crossing measurement.

It can also be seen from this figure how the variable hysteresis band is clamped to a minimum value equal to the previously calculated hysteresis band, as the controller reaches the current error freewheeling region. This is because from equation (5.13), the hysteresis band theoretically reduces to zero at each half cycle zero crossing. This would cause undesirable high frequency switching in this region, which would significantly degrade the harmonic performance of the hysteresis regulator and create significant interactions with the synchronization process. The hysteresis band calculation and the synchronization process smoothly resume a half cycle after exiting the level change region.

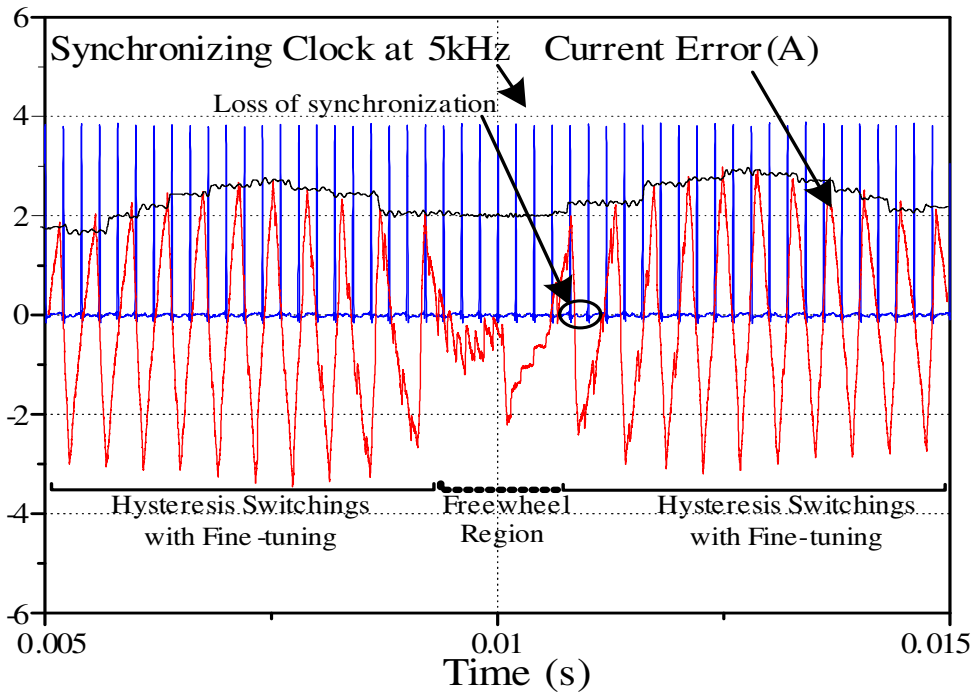


Figure 5.10: Experimental results for synchronization of the current error zero crossings to a fixed reference clock

5.3 Detecting the Output Voltage Level Polarity Change

The next step in the development of the three-level variable hysteresis band HCC is to detect the point when the average phase output voltage changes polarity.

Figure 5.11 shows the performance of a three-level voltage source inverter over one full fundamental cycle using open-loop PD PWM. From this figure and within one full fundamental cycle, the two switching regions (regions 1 and 3) are joined through an intermediate region previously introduced as the level transition region (region 2) as shown in Figure 5.2(b). During this region, the average inverter output voltage changes its polarity from a positive value to a negative value, which causes the current error to freewheel without any switching transitions.

For open-loop PWM this inverter output voltage level change is explicitly implemented as part of the modulation process, where no switching occurs at the point of the average voltage polarity change (modulation command).

In a three-level hysteresis current controlled VSI, a polarity command is required at the point of the average voltage zero-crossing to force the inverter to change its output voltage polarity to reverse the direction of the output current error.

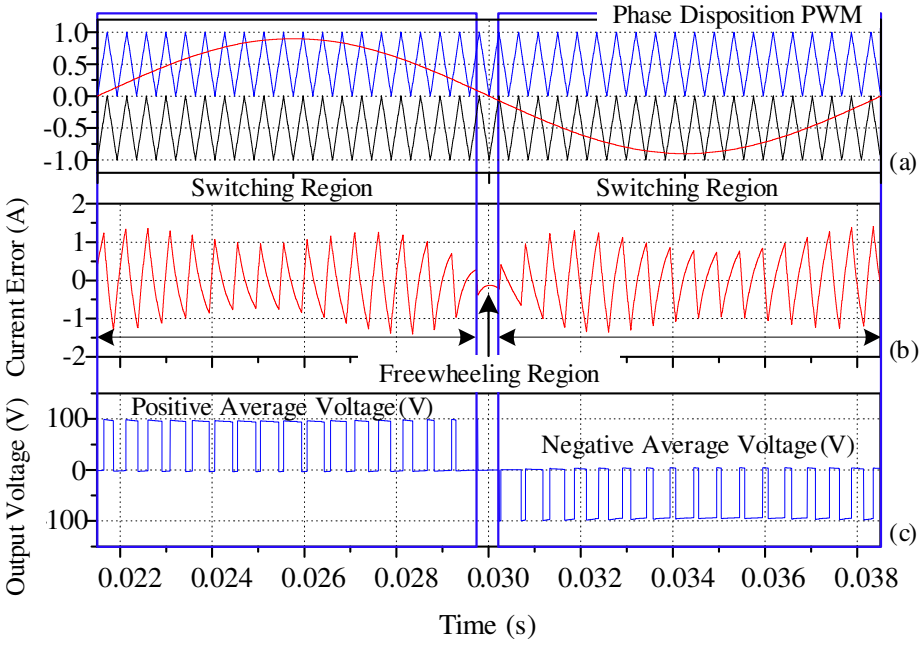


Figure 5.11: Three-level open-loop switching process (a) phase disposition (PD) PWM (b) current error (A) (c) three-level output switched voltage (V)

The proposed HCC uses the absolute normalized average inverter output voltage to calculate the variable hysteresis band using equation (5.13). The zero-crossing information of this average voltage is then used to generate the polarity command as will be now discussed. Figure 5.12 shows the modified hysteresis switching signal $S_{OR}(t)$, the phase leg current error under fixed hysteresis band operation and the extracted absolute average inverter output voltage $S_{OR,avg}(t)$. For clarity, the operating switching frequency is set to be approximately 1 kHz, with a modulation depth of 70%.

Note from this figure how the absolute average inverter voltage reaches a positive minimum value (not zero) as the current error begins to freewheel. At this point, the current error does not ramp back to the correct hysteresis band as it did during the last switching cycle, and hence no switching command is made by the hysteresis comparator. To recognize this event, the controller simply needs to predict when the next switching is expected to occur, which will be a close approximation of when the average voltage crosses zero.

Detecting this output voltage level change is implemented in two steps.

Firstly, the DSP identifies when the fundamental component of the switching signal $S_{OR}(t)$ is reaching to its minimum value near the zero-crossing point. This

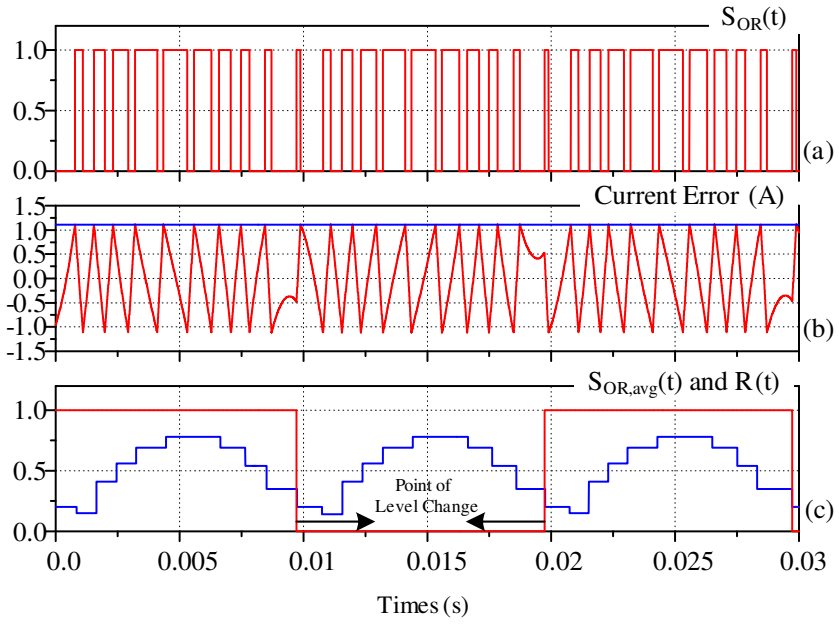


Figure 5.12: (Top) Modified hysteresis switching signal (Middle) phase current error and fixed hysteresis band (c) extracted fundamental component of the modified hysteresis switching signal and the level selection signal

region is shown in Figure 5.12 (c). For this investigation, this value is set heuristically to a minimum value of about 20% of the available DC bus voltage.

Additionally, an auxiliary timer counter is set within the DSP to detect when the absolute average inverter voltage is reaching to its minimum value, for example at every 10mS for a 50Hz average inverter output voltage. This counter is restarted once the point of the inverter polarity change is detected. Using this additional counter has the following advantages:

- The DSP ensures that the prediction algorithm does not confuse the level change region with any possible intermediate current error freewheeling that may be caused by a possible change in the operating conditions such as a sudden load transient.
- At a low back-emf, this becomes more important since the inverter modulation depth may already be at the minimum value of 20%. Hence, the additional counter prevents the prediction algorithm becoming unstable.

Secondly, once the region of the anticipated level change is detected by the DSP, it predicts when the next switching edge should occur based on the previous switching time information of the ON time, the OFF time and the projected switching period. Figure 5.13 shows a zoomed view of the current error freewheeling behaviour near the region of the phase voltage level change to illustrate this process. From this figure, for a nearly fixed switching frequency operation, the next OFF time of the switching signal $T_{OFF\,.pred}$ must be always greater than the previously calculated OFF time $T_{OFF\,,prev}$, i.e.:

$$T_{OFF\,,pred} > T_{OFF\,,prev} \tag{5.20}$$

The prediction algorithm uses this knowledge and sums the previously calculated switching period and the present ON time to give a predicted OFF time of:

$$T_{OFF\,,pred} = T_{prev} + T_{ON} \tag{5.21}$$

If a switching event is not detected by this time, a level change signal $R(t)$ is generated by the DSP to force a switched output polarity change as shown in Figure 5.13(c). This level selection signal is then used by the logic decoder to

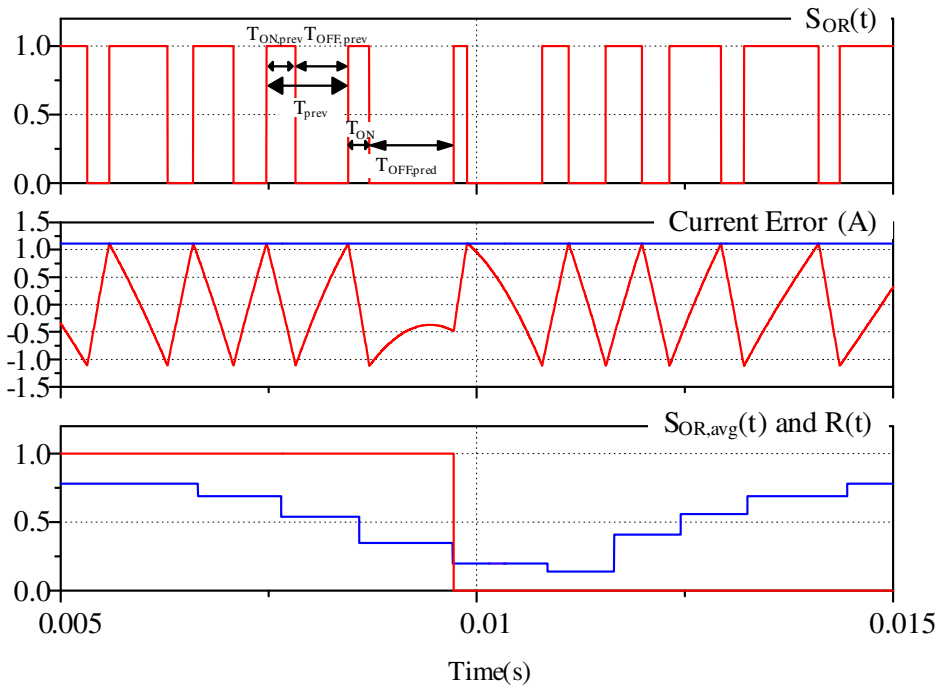


Figure 5.13: Zoomed view of Figure 5.12 near the average voltage zero-crossing

generate the appropriate switching signal for the modulation of the three-level voltage source inverter. The polarity change forces the inverter to immediately switch to an opposite polarity active state. This makes the current error reverse and ramp back to the opposite polarity hysteresis band. When it reaches this band, normal switching recommences.

Figure 5.14 provides the experimental confirmation of this process showing the output current error, which is precisely bounded within the hysteresis limits of $\pm 1.5A$ (set to achieve the 1 kHz switching frequency), hysteresis switching signal $S_{tot}(t)$ generated by the hysteresis comparison of the current error and the hysteresis band, the absolute value of the active switched output pulse (modified hysteresis switching signal) $S_{OR}(t)$, the polarity selection signal $R(t)$ and the reconstituted per-switching-cycle averaged phase leg output voltage $S_{OR,avg}(t)$ (generated by a DAC output from the DSP).

Detecting the point of the output voltage level change from the prediction of the average voltage zero-crossing in this way has the following advantages:

- Only one hysteresis comparator is required per phase leg. This removes the steady state DC tracking error caused by multiple band hysteresis strategies

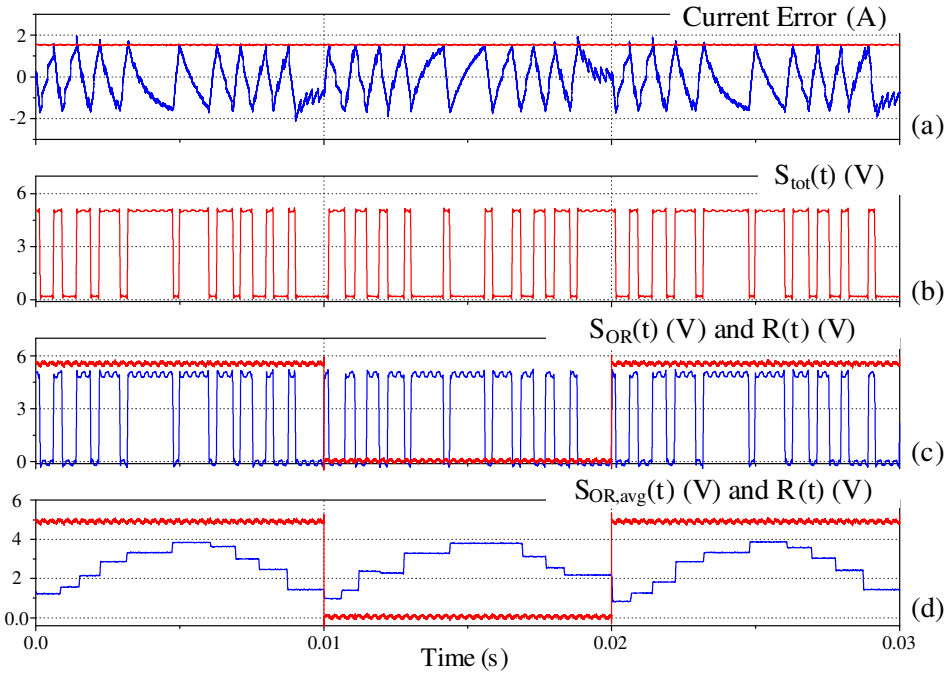


Figure 5.14: (a) Current error and fixed hysteresis band (A) (b) hysteresis switching signal (c) modified hysteresis switching signal (d) extracted fundamental component and the level change signal (V)

[117]. It also reduces the complexity of implementation since only one hysteresis comparator is required per phase leg.

- Detecting the point of voltage polarity change from the average voltage zero-crossing ensures that the phase current error is always within the defined variable hysteresis band. This avoids distortion in the phase output current near the point of the phase voltage level change, caused by an out of band current error. This is particularly important at a lower switching frequency where the maximum allowable hysteresis band in (5.14) can be quite large.

- The controller ensures robust and reliable operation during the transient and overmodulation region by avoiding the interference that can be caused by multiple hysteresis bands.

5.4 Implementation of Hysteresis Modulation for Three-Level Voltage Source Inverters

So far, this chapter has developed a new variable band hysteresis current regulator for any three-level voltage source inverter topology. The only requirement is that the switched phase voltage has three separate switched output states of $V(t) = 0, \pm V_{DC}$.

The next step is to decode the hysteresis output switching signals and the level change signal into specific device switching commands, using additional digital logic to generate the appropriate signals for each VSI topology.

5.4.1 Modulation of a Three-Level Single-Phase Leg Neutral Point Clamped Inverter

5.4.1.1 Open-Loop Modulation of Three-Level NPC Inverter

Figure 5.15 shows the topology of a three-level neutral point clamped inverter with all its possible output switching states and output voltage levels. Table 5.1 summarizes the relationship between the output voltage level and the NPC switching signals. In this topology, the positive active (state 1) and the negative active (state 2) switched phase voltages are generated by turning on the upper ($S_a(t), S_b(t)$) and the lower switches ($\overline{S}_a(t), \overline{S}_b(t)$) respectively. The ZERO state (state 3) output is generated by turning on the middle two switches ($\overline{S}_a(t), S_b(t)$). Note that out of the four possible switch combinations, only three are allowed since state 4 generates a high impedance output and does not provide a current flow path for the output load current. Three-level open-loop level shifted PWM of a NPC inverter achieves this switching arrangement using two level shifted triangle carriers (shown previously in Figure 2.8) where the upper carrier and the positive modulation command generates

Table 5.1: Switching States of the Single Phase Leg NPC Inverter and its corresponding gate signals

Output State	S_a	S_b	Output Voltage
1	1	1	$+V_{DC}$
2	0	0	$-V_{DC}$
3	0	1	0
4	1	0	High Impedance

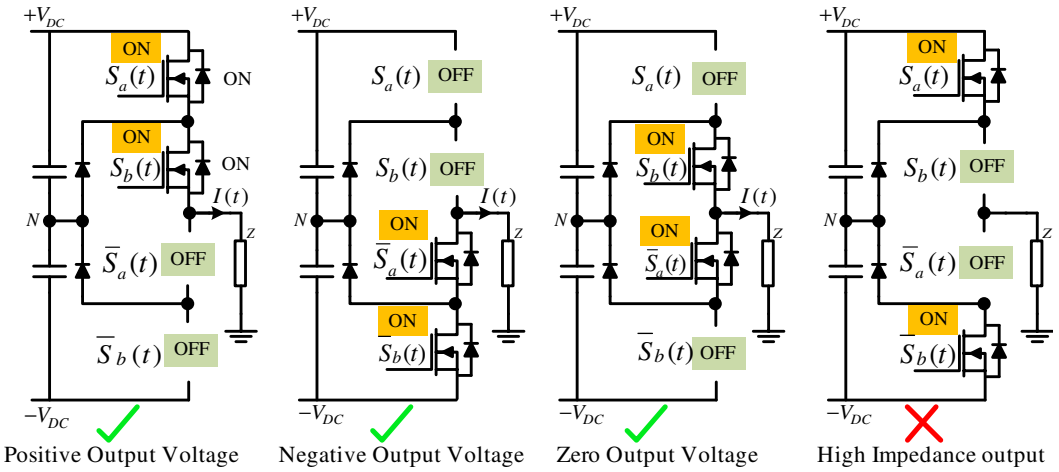


Figure 5.15: Switching states of the three-level NPC inverter and its associated output voltage level

the switched phase voltages ($0, +V_{DC}$) and the lower carrier and the negative modulation command generates the switched phase voltages ($0, -V_{DC}$).

The possible level shifted PWM strategies that have been reported to achieve this switching objective are PD, POD and APOD PWM where their switching patterns and their harmonic performance are essentially identical when applied to a three-level single-phase leg VSI [11][34].

5.4.1.2 Hysteresis Modulation of Three-Level NPC Inverter

Figure 5.16 shows the combinational logic circuit developed to use these switching combinations for the new three-level hysteresis current regulator. The decoding logic is a combination of two AND gates and two OR gates as shown in Figure 5.16. The upper AND gate inputs are $S_{tot}(t)$ and the level selection signal $R(t)$, and it generates switching signals ($S_a(t), \overline{S}_a(t)$) to achieve the switched phase voltages of ($0, +V_{DC}$). The lower AND gate and OR gate has inputs of $S_{tot}(t)$ and the complementary level selection signal $\overline{R}(t)$, and generates switching signals ($S_b(t), \overline{S}_b(t)$) to achieve the switched phase voltage of ($0, -V_{DC}$).

The combinational logic also generates the modified hysteresis switching signal $S_{OR}(t)$ using the feedback OR gate. This is equivalent to the scaled absolute value of the active phase leg switched voltage. The inputs to the feedback OR gate are the device switching signals of:

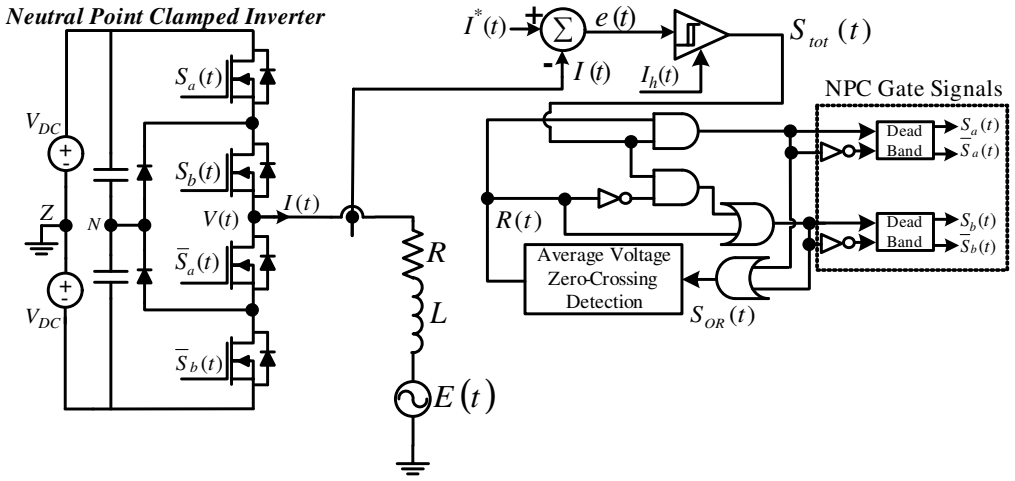


Figure 5.16: Combinational logic circuit to decode the total switching signal $S_{tot}(t)$ into gate switching signals for NPC and FC inverter.

$$S_{OR}(t) = S_a(t) \mid \overline{S}_b(t) \tag{5.22}$$

The OR gate output feeds into the DSP capture/timer port, as a ONE for an active switched output voltage of either polarity, and a ZERO for a 0V phase leg output. The DSP then extracts the scaled per-switching-cycle averaged inverter voltage based on equation (5.16) to calculate the variable hysteresis band and generates the output level change signal as discussed in section 5.3.

Figure 5.17 compares the generated device switching signals from the proposed logic decoder in Figure 5.16, with the switching signals generated by a three-level open-loop PD PWM.

This figure confirms that the hysteresis switching signals ($S_a(t)$, $S_b(t)$) and the open-loop PD PWM signals ($S_{PD,a}(t)$, $S_{PD,b}(t)$) are essentially the same, since:

- Both the $S_a(t)$ and $S_b(t)$ hysteresis switching signals have a discontinuous switching pattern similar to the PD switching signals $S_{PD,a}(t)$ and $S_{PD,b}(t)$
- $S_a(t)$ and $S_{PD,a}(t)$ are both switching during the positive average inverter output voltage.
- $S_b(t)$ and $S_{PD,b}(t)$ are both switching during the negative average inverter output voltage.

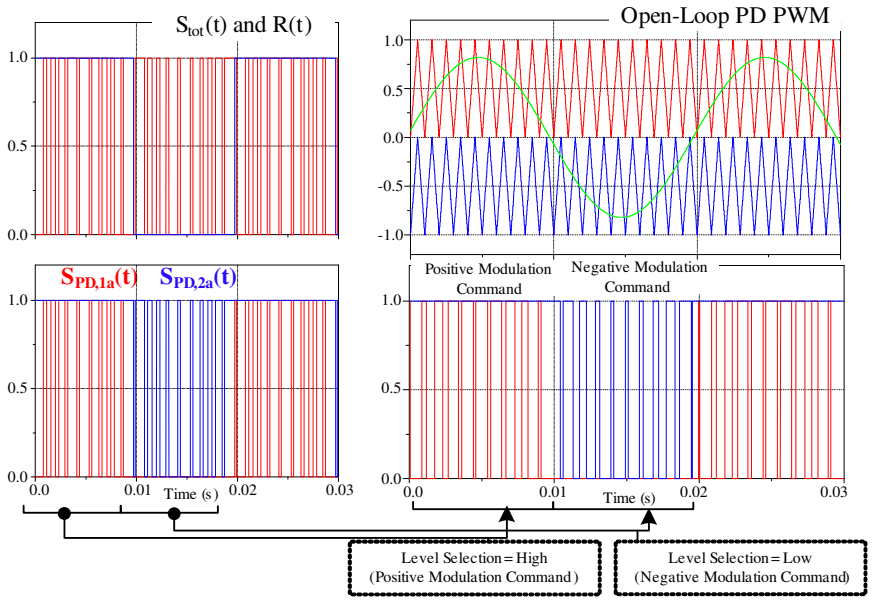


Figure 5.17: Comparison of the switching signals generated from the developed logic decoder and a three-level open-loop PD PWM

Figure 5.18 provides experimental confirmation of this process, showing the hysteresis switching signal $S_{tot}(t)$ generated by the hysteresis comparison of the phase current error and the hysteresis band, the modified hysteresis switching signal

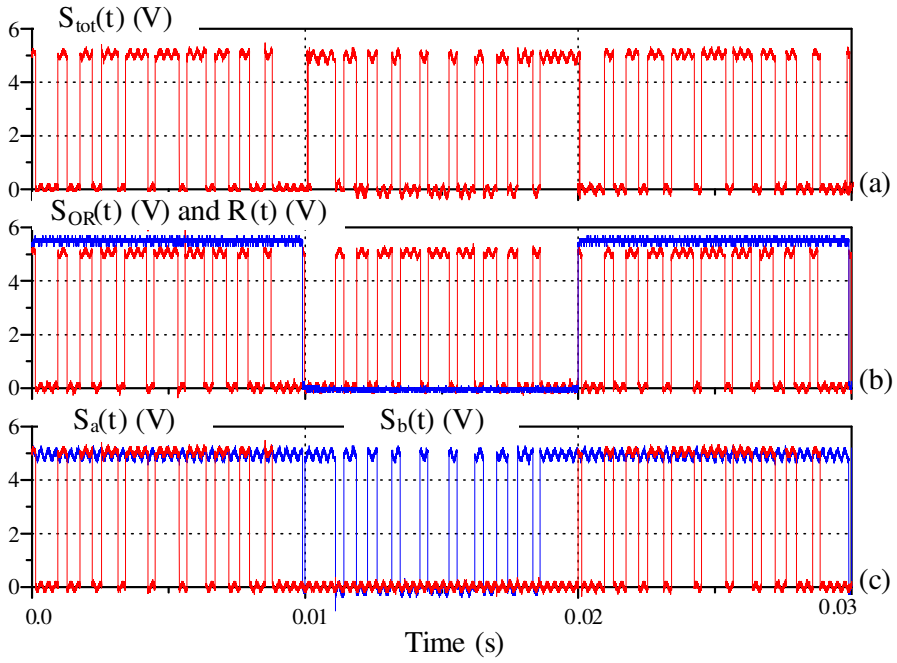


Figure 5.18: Hysteresis switching signal $S_{tot}(t)$ generated by hysteresis comparison of current error and hysteresis band (b) modified hysteresis switching signal (output of feedback OR gate) $S_{OR}(t)$ (c) gate switching signals ($S_a(t)$ & $S_b(t)$), 1 kHz switching frequency.

$S_{OR}(t)$ (absolute value of the active switched output pulse), the level selection signal $R(t)$ and the inverter phase leg switching signals $S_a(t)$ and $S_b(t)$.

5.4.2 Modulation of a Three-Level Single-Phase Leg Flying Capacitor Inverter

5.4.2.1 Open-Loop Modulation of Three-Level FC Inverter

Figure 5.19 shows the topology of a three-level flying capacitor inverter with all of its possible output switching states and output voltage levels. Table 5.2 summarizes the relationship between the output voltage levels and the switching state of the gate signals. Similar to the three-level NPC inverter, for the FC VSI, the positive (state 1) and the negative (state 2) switched phase voltages are generated by turning on the upper ($S_a(t)$, $S_b(t)$) and the lower switches ($\overline{S}_a(t)$, $\overline{S}_b(t)$) respectively.

In contrast to the NPC inverter, the zero switching state of the FC VSI is generated by the two switching combinations of ($\overline{S}_a(t), S_b(t)$) and ($S_a(t), \overline{S}_b(t)$) as shown in Figure 5.19 and Table 5.2. This switching state redundancy can be used to optimise the modulation of this topology [8].

Also from open-loop PWM [35][36], in order to achieve natural balancing of the capacitors in a FC inverter, all the switches should be modulated with the same switching frequency and approximately the same duty cycle while utilizing the switching signal redundant states. In this way, the controller does not require additional voltage sensors to implement an active balancing strategy to maintain a balanced FC voltage. While PD PWM is known to have superior harmonic performance in comparison to PS PWM [24], it requires modification for a FC inverter to naturally balance the capacitor voltages. In [35], a PWM decoder using a finite state machine was presented to resolve this issue by using a "round robin" assignment of the zero switching states to maintain a balanced capacitor voltage.

Table 5.2: Switching States of the Single Phase Leg FC and Single Phase H-Bridge Inverters and their corresponding gate signals

Output State	S_a	S_b	Output Voltage
1	1	1	$+V_{DC}$
2	0	0	$-V_{DC}$
3	0	1	0
4	1	0	0

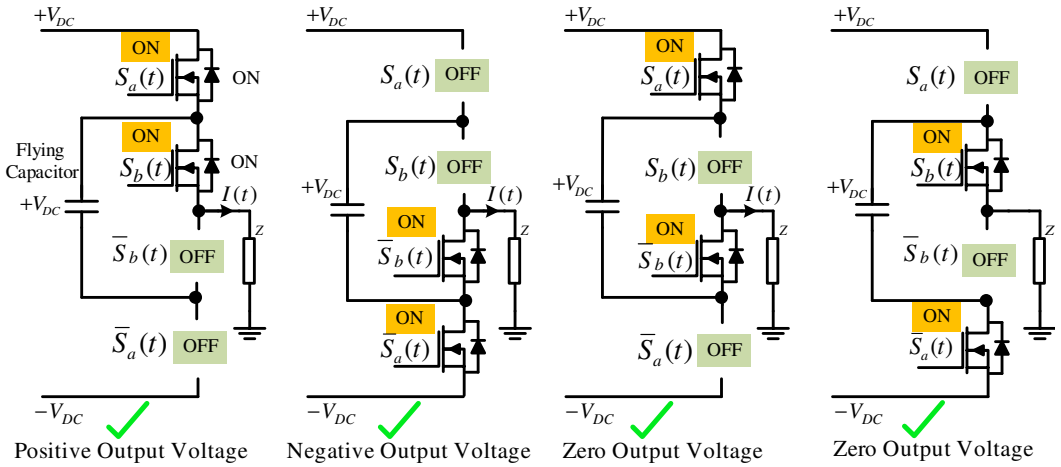


Figure 5.19: Switching states of the three-level FC inverter and its associated output voltage level

This section develops a finite state machine using similar concepts as presented in [35] for the hysteresis modulation of single-phase FC inverter.

5.4.2.2 Hysteresis Modulation of Three-Level FC Inverter

Figure 5.20 shows the modular structure of the three-level single-phase leg FC inverter, consisting of two power cells. Each cell is the combination of two switches and a flying capacitor, modulated in such a way to generate the switching groups (0, $+V_{DC}$) and (0, $-V_{DC}$). Additionally, these two cells must be switched at the same duty cycle at the same switching frequency.

Using this concept, a finite state machine is developed for the proposed three-level hysteresis controller as shown in Figure 5.21. The inputs to the finite state machine are the PD equivalent switching signals ($S_{PD1}(t)$, $S_{PD2}(t)$) that have been generated using the logic decoder developed in section 5.4.1. The upper row of the FSM belongs to the positive average inverter output voltage and the phase voltage switching combination (0, $+V_{DC}$) and the lower row belongs the average inverter output voltage and the phase voltage switching combination (0, $-V_{DC}$). In each row of the finite state machine, the two zero states ($S_a(t)=1$, $S_b(t)=0$) and ($S_a(t)=0$, $S_b(t)=1$) are distributed evenly as follows:

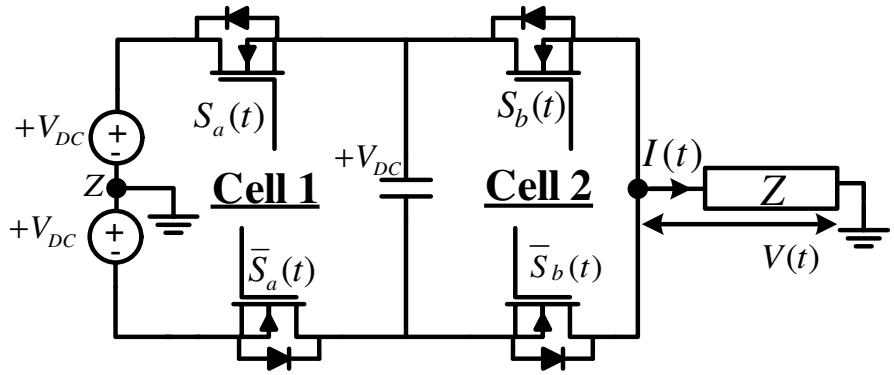


Figure 5.20: Modular structure of a single leg FC inverter consists of two power cells

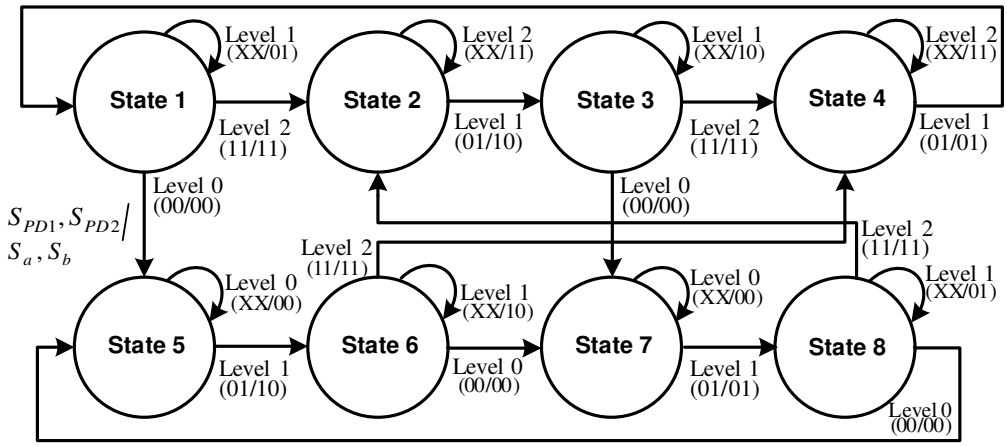


Figure 5.21: State machine diagram used to utilize the redundant zero state

1) During the positive switching region ($V_{avg}(t) > 0$), the FSM uses the sequence of:

$$+V_{DC} \rightarrow 0_1 \rightarrow +V_{DC} \rightarrow 0_2$$

2) During the negative switching region ($V_{avg}(t) < 0$), the FSM uses the sequence of:

$$-V_{DC} \rightarrow 0_1 \rightarrow -V_{DC} \rightarrow 0_2$$

where 0_1 and 0_2 are the two zero switching states as listed in Table 5.2.

The outputs of the FSM ($S_a(t)$, $S_b(t)$) are the switching signals used to modulate the single-phase leg FC. The pattern of these switching signals is similar to the PS PWM technique which is known to naturally balance the capacitor voltages while keeping the harmonic benefits of PD PWM [35].

Figure 5.22 shows the block diagram of the combinational logic and the FSM that is used to decode the hysteresis switching signal and generate the appropriate switching signals for the modulation of the FC inverter. Figure 5.23 provides

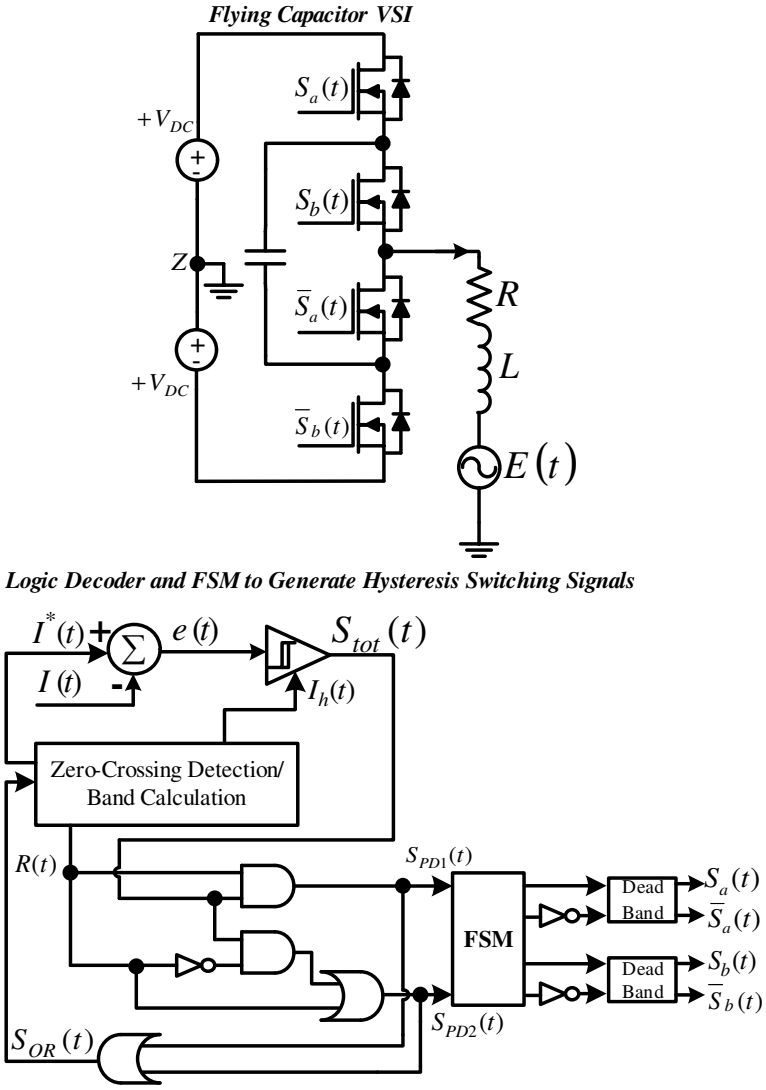


Figure 5.22: Combinational logic and FSM circuit to decode the total switching signal S_{tot} *(t)* into gate switching signals for FC and H-bridge inverters.

experimental confirmation of this process showing the PD equivalent switching signals ($S_{PD,1,a}(t)$, $S_{PD,2,a}(t)$) which are the input to the finite state machine (used previously to directly modulate the three-level NPC inverter). It then shows the switching signals ($S_a(t), S_b(t)$) which are the output from the FSM and are used to directly modulate the three-level FC inverter. Note how the switching pattern of the gate signals in Figure 5.23(a)(b) is similar to open-loop PS PWM.

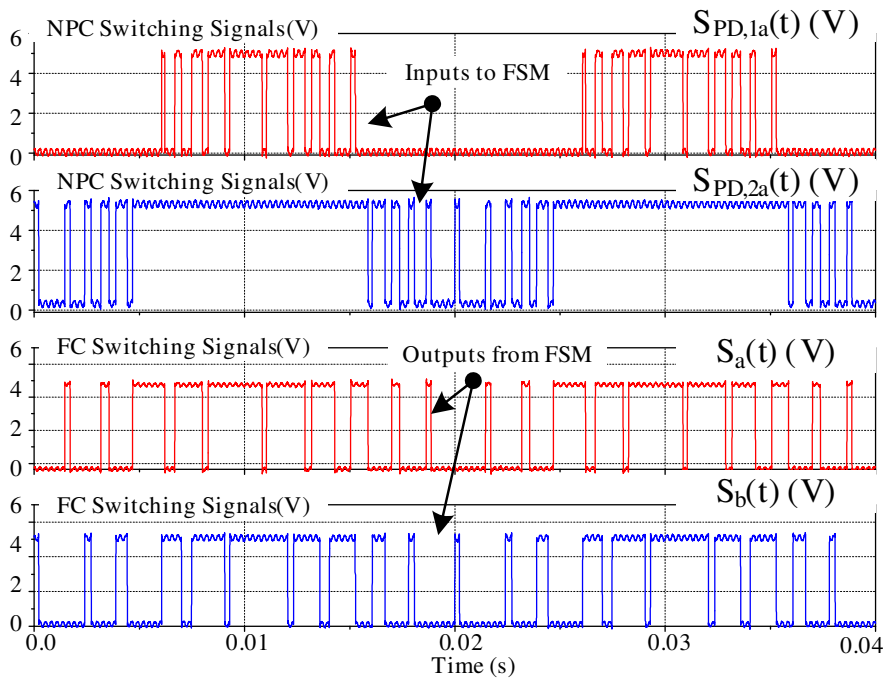


Figure 5.23: Switching signals to modulate the three-level FC inverter (a) (b) Equivalent PD switching signal (input to FSM) (c)(d) Generated output switching signal (Output from FSM)

5.4.3 Modulation of a Three-Level Single Phase H-Bridge Inverter

5.4.3.1 Hysteresis Modulation Single Phase H-Bridge Inverters

Figure 5.24 show the topology of a three-level single-phase H-bridge inverter with all of its possible output switching states and output voltage levels. By comparing this figure with Figure 5.19 it can be concluded that both the three-level single-phase H-bridge and the FC inverter have the same voltage levels and gate switching states as listed in Table 5.2. Hence, the same hysteresis controller and the FSM developed in the previous section can be immediately used for a three-level modulation of a single-phase H-bridge.

However, it is not necessary to use the developed combinational logic presented in section 5.4.1 for three-level hysteresis modulation of a H-bridge inverter. Alternatively, the switching signal of the single hysteresis comparator $S_{tot}(t)$ and the level selection signal $R(t)$ can be directly used as the inputs to the FSM to generate the appropriate switching signals.

Figure 5.25 shows the structure of a controller that achieves a three-level variable band hysteresis result in this way, with a single variable band hysteresis comparator

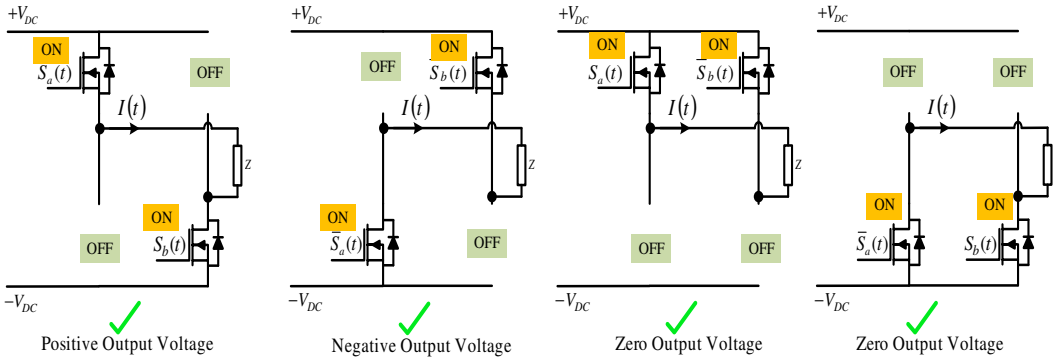


Figure 5.24: Switching states of the three-level single-phase H-bridge inverter and its associated output voltage level

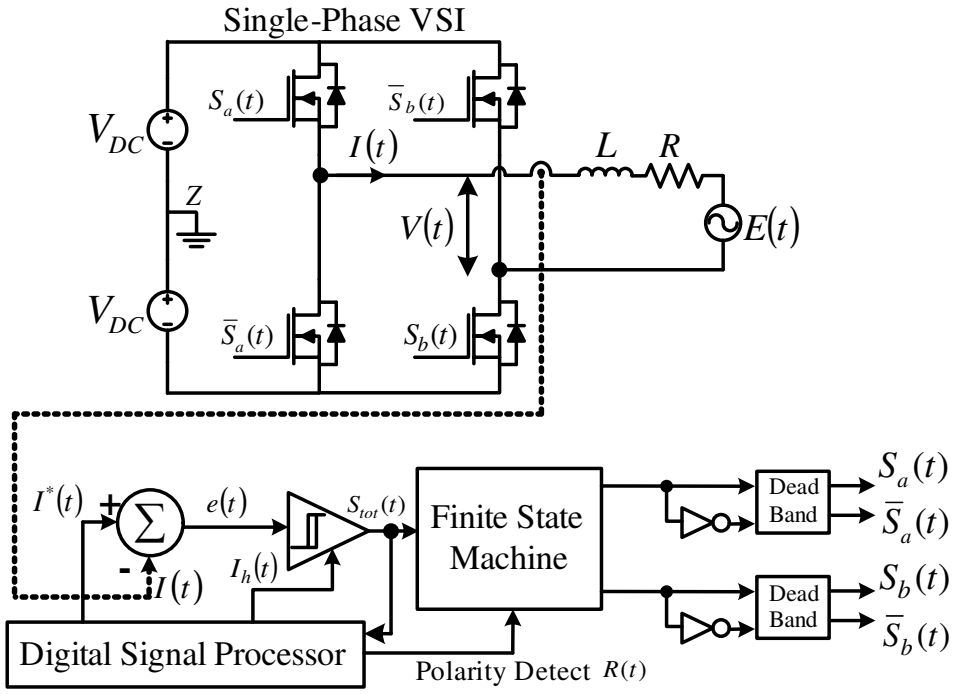


Figure 5.25: FSM circuit to decode the total switching signal Stot (t) and the level change signal $R(t)$ into gate switching signals for H-bridge inverters.

output feeding into a finite state machine shown in Figure 5.26 that translates the comparator output $S_{tot}(t)$ into the required four gate signals for the bridge.

From Figure 5.26 the inputs to the FSM are the comparator hysteresis switching signal $S_{TOT}(t)$ and the polarity detect signal $R(t)$. The FSM utilizes the inverter ZERO states using round robin assignment for both the positive and negative average inverter voltages, to generate the inverter switching signals $S_a(t)$ and $S_b(t)$.

Figure 5.27 provides experimental confirmation of this process showing switching signals for the modulation of the three-level single phase H-bridge using the FSM. The inputs to the FSM are the comparator hysteresis switching signal $S_{tot}(t)$ and the

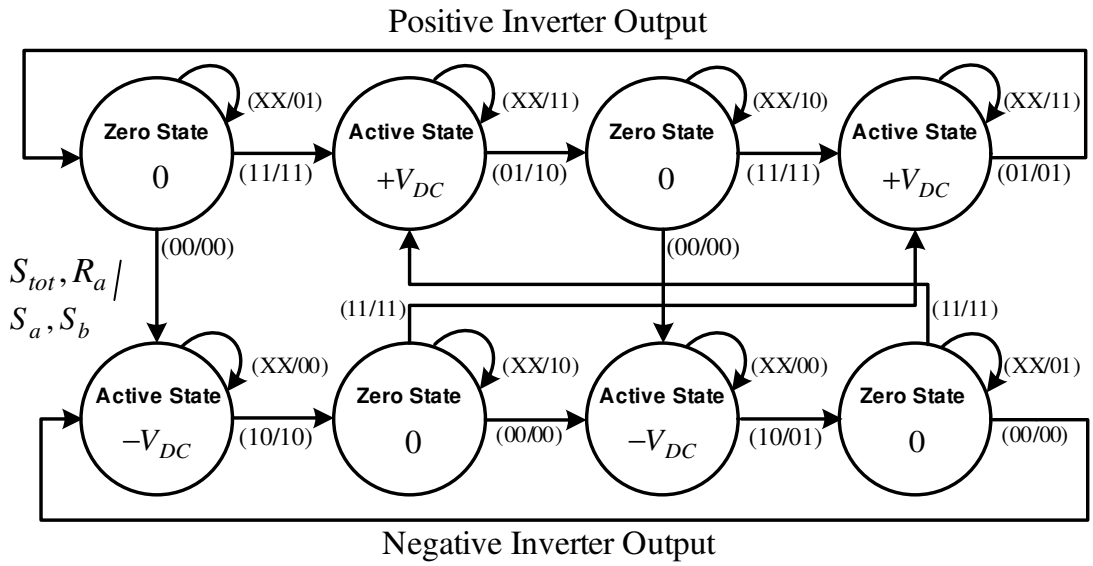


Figure 5.26: State machine diagram used to utilize the redundant zero state from the total switching signal S_{tot} (t) and the level change signal $R(t)$

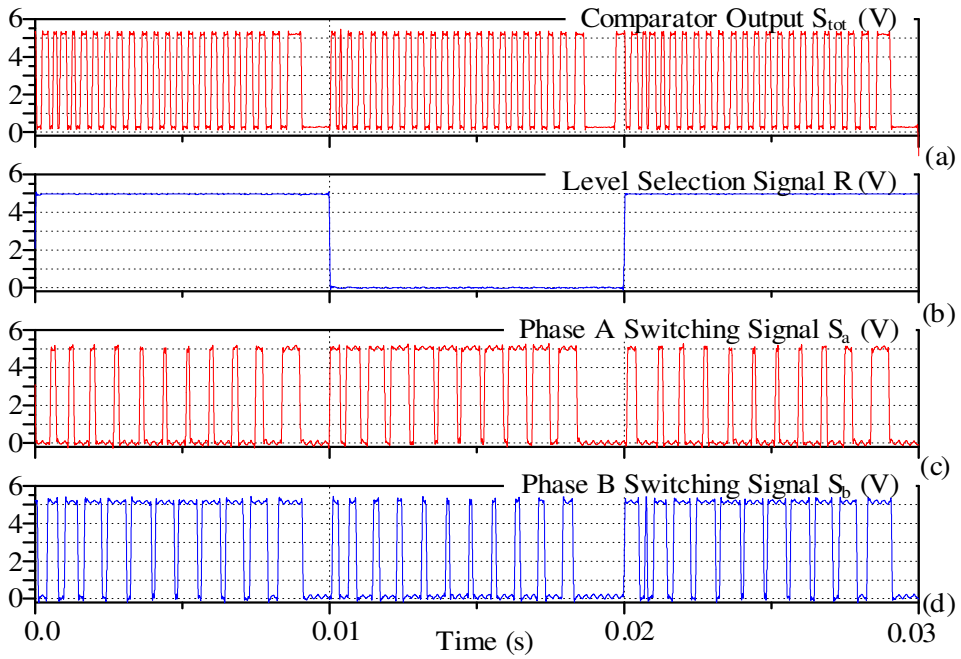


Figure 5.27: (a) (b) Equivalent PD switching signal (input to FSM) (c)(d) Generated output switching signal (Output from FSM)

polarity detect signal $R(t)$. The FSM utilizes the inverter ZERO states with round robin assignment to generate the inverter switching signals $S_a(t)$ and $S_b(t)$ as shown in Figure 5.27 (c)(d).

5.5 Hysteresis Band Clamping for Three-Level Variable Hysteresis Band Operation

For a three-level variable band hysteresis current regulator it is necessary to clamp the variable hysteresis band near two regions:

1) At the peak of the average inverter output voltage to avoid controller instability during overmodulation.
2) At the zero-crossings of the average inverter voltage to improve the inverter switching performance near the polarity change region.

The details of each of these band clamp regions are discussed in this section.

5.5.1 Managing the Overmodulation Region

For open-loop three-level PD PWM, overmodulation occurs when the peak of the sinusoidal modulation command exceeds the upper and the lower triangular carrier magnitudes ($V_{avg} > 1.0V_{DC}$) [11][16]. This distorts the linearity of the modulator operation and generates significant increased low frequency baseband distortion.

Section 3.4 has shown that for variable band hysteresis regulators there is some hazard with overmodulation. For a two-level VSI, this was solved by clamping the variable hysteresis band at a minimum value as the controller enters the overmodulation region. The same approach is now applied to a three-level VSI.

From equation (5.13) as the peak average voltage approaches V_{DC}, the variable hysteresis band approaches zero, viz:

$$I_h = I_{h\_\max}\left|V_{avg}(t)/V_{DC}\right|\left(1-\left|V_{avg}(t)/V_{DC}\right|\right) \qquad x \in A,B,C \qquad (5.23)$$
$$\xrightarrow{V_{avg}(t)=V_{DC}} I_h = 0$$

Figure 5.28 shows the consequence of this constraint, with high frequency switching occurring at the peak of the average inverter output voltage as the variable hysteresis band limit approaches zero and the controller loses stability.

However, extracting the fundamental component of the modified hysteresis switching signal ($S_{avg,OR}(t)$) allows the DSP to recognize the attempt to overmodulate the inverter when the cyclic average phase leg voltage starts to exceed the available bus voltage. In response, the DSP limits the reducing hysteresis band to

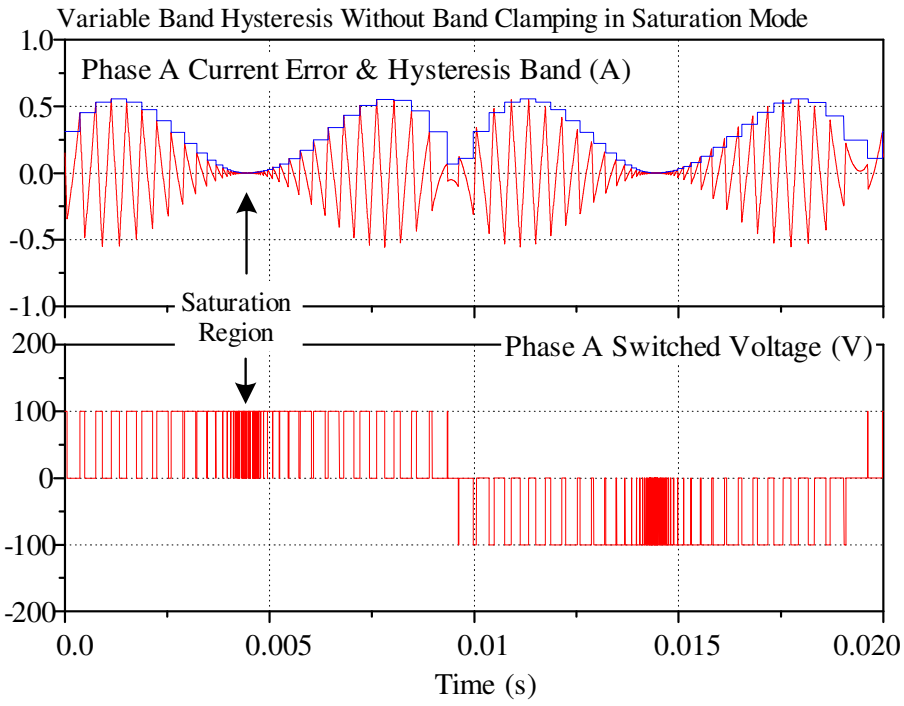


Figure 5.28: Simulated performance of variable band hysteresis current regulation without hysteresis band clamping, peak average inverter output voltage = 1.0*V_{DC}

a defined minimum value and allows the hysteresis regulators to go into an overmodulation response. The band clamping response is managed by the DSP, by heuristically ensuring that the variable hysteresis limit $I_h(t)$ calculated from equation (5.13) is constrained to a minimum value of approximately 20% of $I_{h,\max}$.

VSI switching naturally recommences after overmodulation as the per phase current error returns within the hysteresis bands and hits either the upper or lower variable hysteresis bands. Once switching recommences as the overmodulation situation reverses, the DSP smoothly ceases variable band limiting.

Figure 5.29 confirms the validity of the band limit clamping technique, for application to a single-phase leg NPC VSI. The hysteresis regulator smoothly progresses into overmodulation without any sign of high frequency switching. From this figure, it can also be seen how the regulator still tracks the reference current even in the nonlinear overmodulation region, with no sign of transient disturbance as the inverter enters and leaves this region of operation. Additionally the controller can easily differentiate between the region of overmodulation and the level change region since overmodulation occurs at the maximum peak of the average inverter voltage while polarity change occurs when the average inverter voltage is near zero.

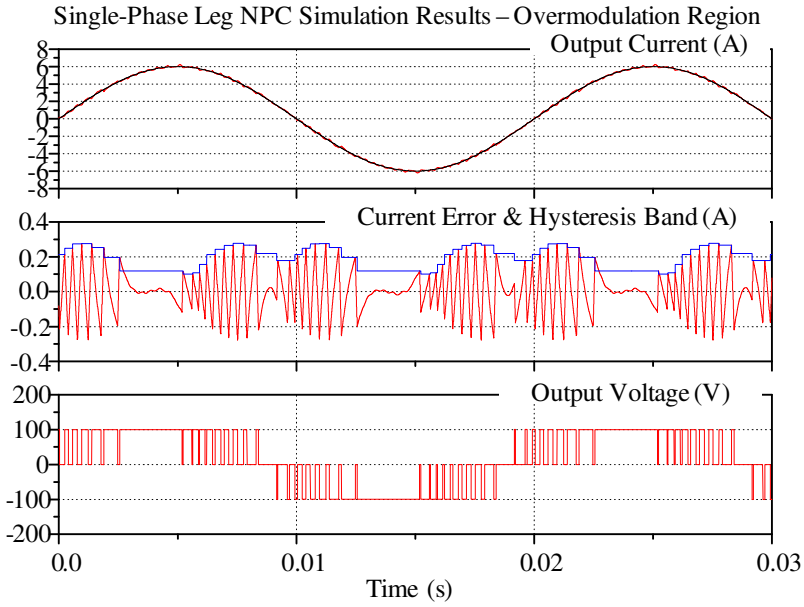


Figure 5.29: Simulated performance of variable band hysteresis current regulation with hysteresis band clamping, peak average inverter output voltage = 1.1*V_{DC}

5.5.2 Managing the Output Voltage Level Change Region

For the variable hysteresis band calculation of the three-level VSIs as per equation (5.13), there is an additional requirement for a band-clamping strategy near the zero-crossing of the average load voltage. As the average voltage becomes zero, the variable hysteresis band will also become zero, viz:

$$I_h = I_{h\_\max}\left|V_{avg}(t)/V_{DC}\right|\left(1-\left|V_{avg}(t)/V_{DC}\right|\right) \qquad x \in A,B,C \qquad (5.24)$$
$$\xrightarrow{V_{avg}(t)=0} I_h = 0$$

Figure 5.30(a)(b) illustrates the consequence of this constraint at the zero-crossing of the average inverter output voltage where the variable hysteresis band limit approaches zero and the controller starts high frequency switching in this region. There are also two additional phenomena to consider around this region where

1) Computation and sampling delay are imposed onto calculation of the average inverter output voltage.

2) The assumption of constant average voltage over one switching cycle is invalid.

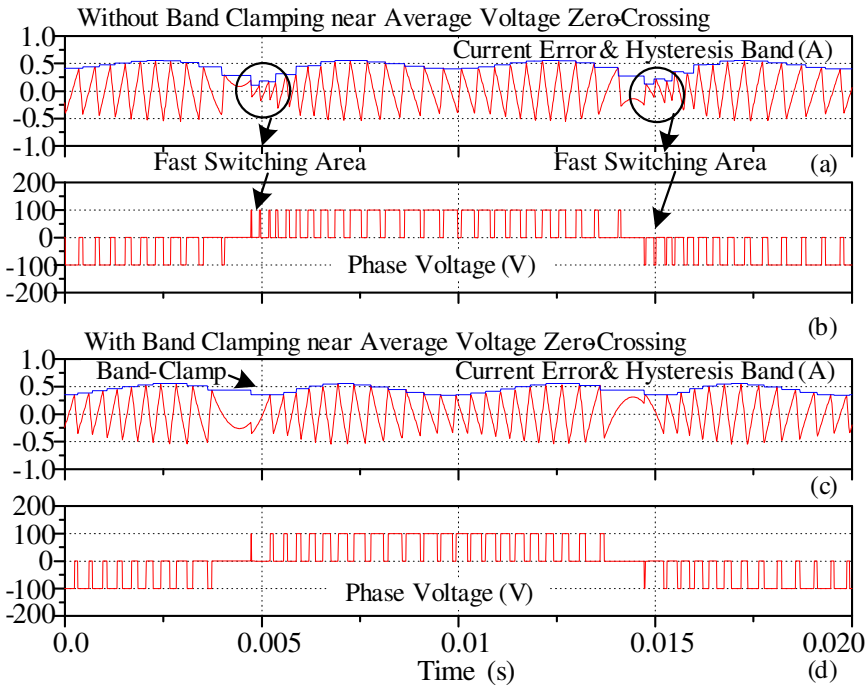


Figure 5.30: Simulated performance of variable band hysteresis current regulation showing the phase current error and voltage (a)(b) with hysteresis band clamping near average voltage zero-crossing (c)(d) without hysteresis band clamping near average voltage zero-crossing

These two factors cause early switching around the average voltage zero-crossing region, at a higher switching frequency than the target switching frequency as shown in Figure 5.30(a)(b). This will affect the phase and line to line voltage harmonic performance unless the band-clamp strategy is applied to the hysteresis band around this region, heuristically set to about 20% of the average inverter output voltage, as shown in Figure 5.30(c)(d). From these figures, the current error then loops back from the hysteresis boundary without actually hitting it, while the controller smoothly continues switching without difficulty after the polarity change transition.

Note that this response improves the switching frequency regulation around this region, but also causes the switching process to lose synchronism for a short time as shown in Figure 5.10.

Experimental results to confirm the effectiveness of the band clamping strategy for a three-level inverter are provided in chapter 6 as part of the three-phase system results.

5.6 Consolidated Simulation and Experimental Results

Throughout this chapter, selected simulation and experimental results have been presented in order to support the development of the theoretical analysis. These results are now re-presented in this section together with more extensive simulation and experimental results as a consolidated reference to show the operation of the new three-level hysteresis current regulator. Circuit parameters for the system are listed in Table 5.3.

The resultant performance of the new hysteresis current regulation approach has been verified for a three-level single-phase leg NPC inverter (Figure 5.31 (simulation) and Figure 5.32 (experiment)), a three-level single-phase leg FC inverter (Figure 5.35 (simulation) and Figure 5.36 (experiment)) and a three-level single-phase H-bridge (Figure 5.39 (simulation) and Figure 5.40 (experiment)). Results for a three-level fixed-band hysteresis controller are also shown for all of these topologies, to provide a baseline for comparison against the performance of the variable band hysteresis current regulator.

The figures show excellent output current tracking capability under both fixed and variable hysteresis band operation without any sign of DC tracking error. Also they show how the magnitude of the hysteresis variable band changes during the fundamental cycle, reaching a maximum value at the 45^{o} points on the fundamental cycle, and a minimum value at the peaks and zero crossings of the average inverter output voltage. During each zero crossing transition (polarity change), the variable band is clamped to a minimum value, to avoid calculating a zero hysteresis band value, which is not physically realisable. As seen in these figures, during the level

Table 5.3: Circuit parameters for the simulation implementation of the new hysteresis current regulation approach applied to a three-level single-phase leg NPC and FC inverter

Circuit Parameter		**Value**
Resistive load	(R) (Ω)	0.2
Inductive load	(L) (mH)	18
Target Switching frequency	(f_{sw}) (kHz)	2.5
Total DC bus voltage	($2V_{DC}$) (V)	200
Reference current	(I_{ref}) (A)	5
Back EMF frequency	(Hz)	50

change transition, the current error loops back from the hysteresis boundary without actually hitting it, while the controller continues switching without difficulty after the polarity change.

The figures also show a clear three-level switched phase voltage, where the distinct level transitions can be seen near the adjacent voltage levels. This confirms the effectiveness of the level selection strategy based on the prediction of the average voltage zero-crossing discussed in section 5.3, which avoids the erroneous polarity transitions that are typical of normal multiband hysteresis implementations.

Figure 5.36 also shows the balanced flying capacitor voltage to confirm the effectiveness of the active balancing strategy to maintain a balanced FC voltage using the finite state machine, without requiring additional voltage measurement.

Figure 5.33 (simulation) and Figure 5.34 (experiment) show the harmonic performance of the new hysteresis current regulator when applied to a three-level single-phase leg NPC inverter at the operating modulation depth of 0.9. The figures show how the use of a variable hysteresis band significantly improves the output voltage harmonic performance compared to a three-level fixed-band HCC. In simulation, the WTHD is reduced from 1.89% for a fixed-band HCC to 1.30% for the variable band HCC. The WTHD is further reduced to 1.19% by synchronizing the current error zero crossing to a fixed reference clock with a defined carrier and side band harmonics around the target switching frequency of 2500Hz. The same results are confirmed experimentally where the WTHD is reduced from 1.64% for a fixed-band HCC to 1.45% for the variable band HCC and to 1.32% with clock synchronisation. The final harmonic performance of the new three-level variable hysteresis band is essentially equivalent to that of harmonically optimum open-loop phase disposition (PD) PWM with a large carrier harmonic and substantially reduced side band components. This level of harmonic performance has significant advantages in terms of harmonic cancellation between the phase legs when applied to three-phase three-level VSIs topologies, as discussed in the next chapter.

Figure 5.37 (simulation) and Figure 5.38 (experiment) show the harmonic performance of the new hysteresis current regulator when applied to a three-level single-phase leg flying capacitor inverter at an operating modulation depth of 0.9. These figures show the same level of harmonic improvement as has been achieved for the NPC inverter, when using the new hysteresis current regulator for the FC inverter. Also, by comparing these figures, it can be seen that the harmonic

performance of the new hysteresis modulator is essentially identical irrespective of the three-level single-phase leg VSI topology.

Figure 5.41 (a) shows the experimental harmonic performance achieved by the new variable hysteresis band current regulator for a single-phase H-bridge. Figure 5.41 (b) confirms the harmonic cancellation that is achieved between the phase legs (phase leg A and B) using the proposed three-level hysteresis current regulator which has a very close match to the spectrum achieved by open loop asymmetrical double edge regular sampled PWM for the same topology [11]. This further confirms that an almost constant switching frequency has been achieved by the algorithm. Also from Figure 5.41 (b), the switching frequency of the line to line voltage is twice the phase voltage switching frequency and the suppression of the odd carrier multiplies and their sidebands can be clearly observed.

From these experimental and simulation results, it can be seen that the new three-level variable band hysteresis current regulator achieves excellent current tracking without any sign of DC tracking error. The output current error is precisely bounded within the variable hysteresis bands during the switching and the polarity change regions. The output voltage has a clear three-level switched phase voltage, with the distinct level transition near the adjacent voltage levels. For the NPC and FC inverter, the final harmonic performance of the controller is close to optimum open-loop PD PWM. For the H-bridge inverter, the final harmonic performance is close to the open-loop double edge asymmetrical PWM.

Simulation- Three-Level Neutral Point Clamped Inverter

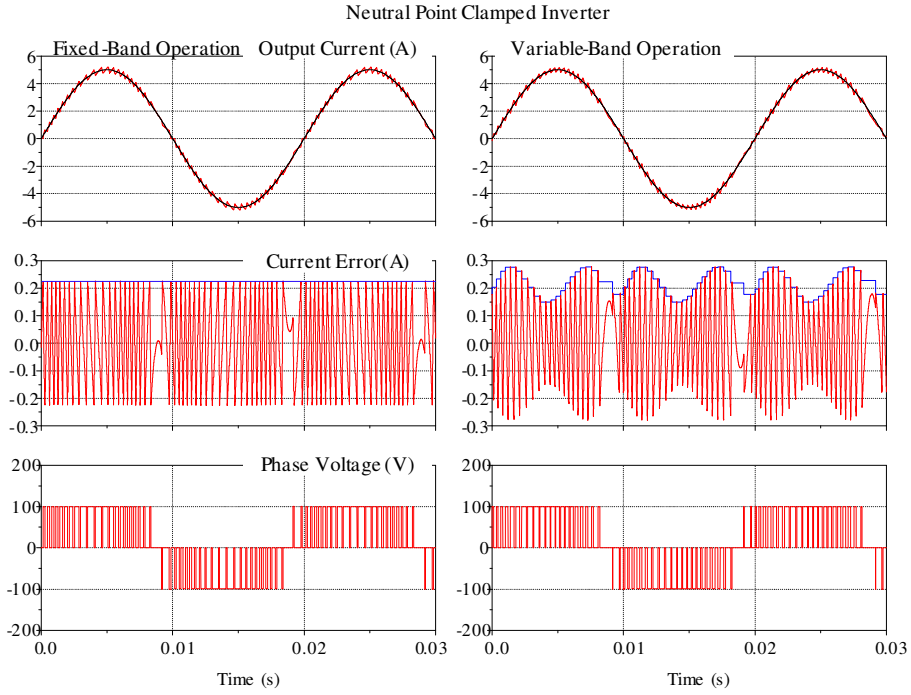


Figure 5.31: Single-phase leg simulation results for the new three-level hysteresis current regulator under fixed band and variable band operation (a) Output current (b) Current error and hysteresis band (c) three-level output voltage

Experiment- Three-Level Neutral Point Clamped Inverter

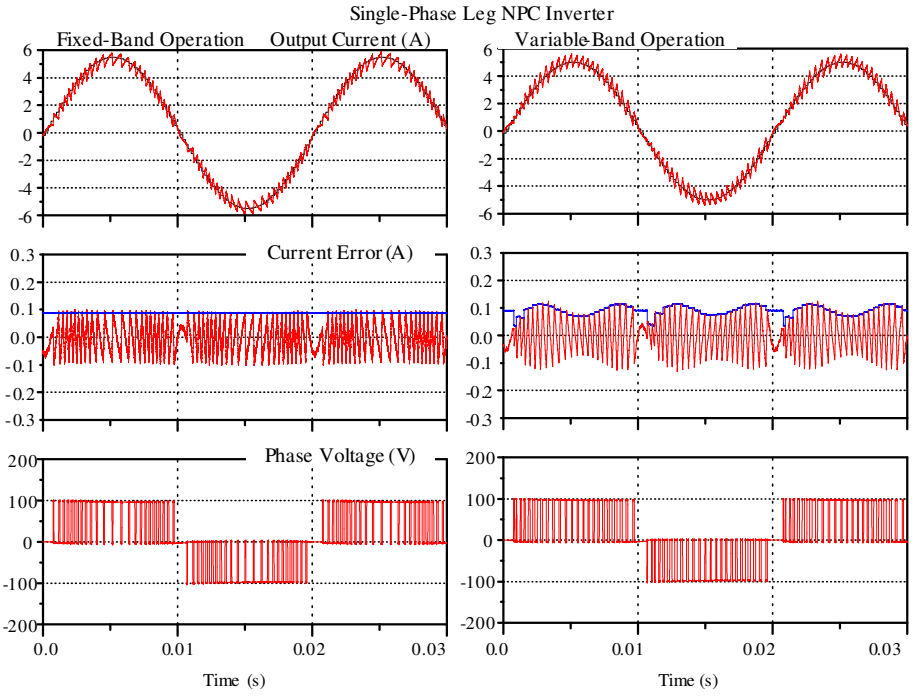


Figure 5.32: Single-phase leg experimental results for the new three-level hysteresis current regulator under fixed band and variable band operation (a) Output current (b) Current error and hysteresis band (c) three-level output voltage

Simulation- Three-Level Neutral Point Clamped Inverter

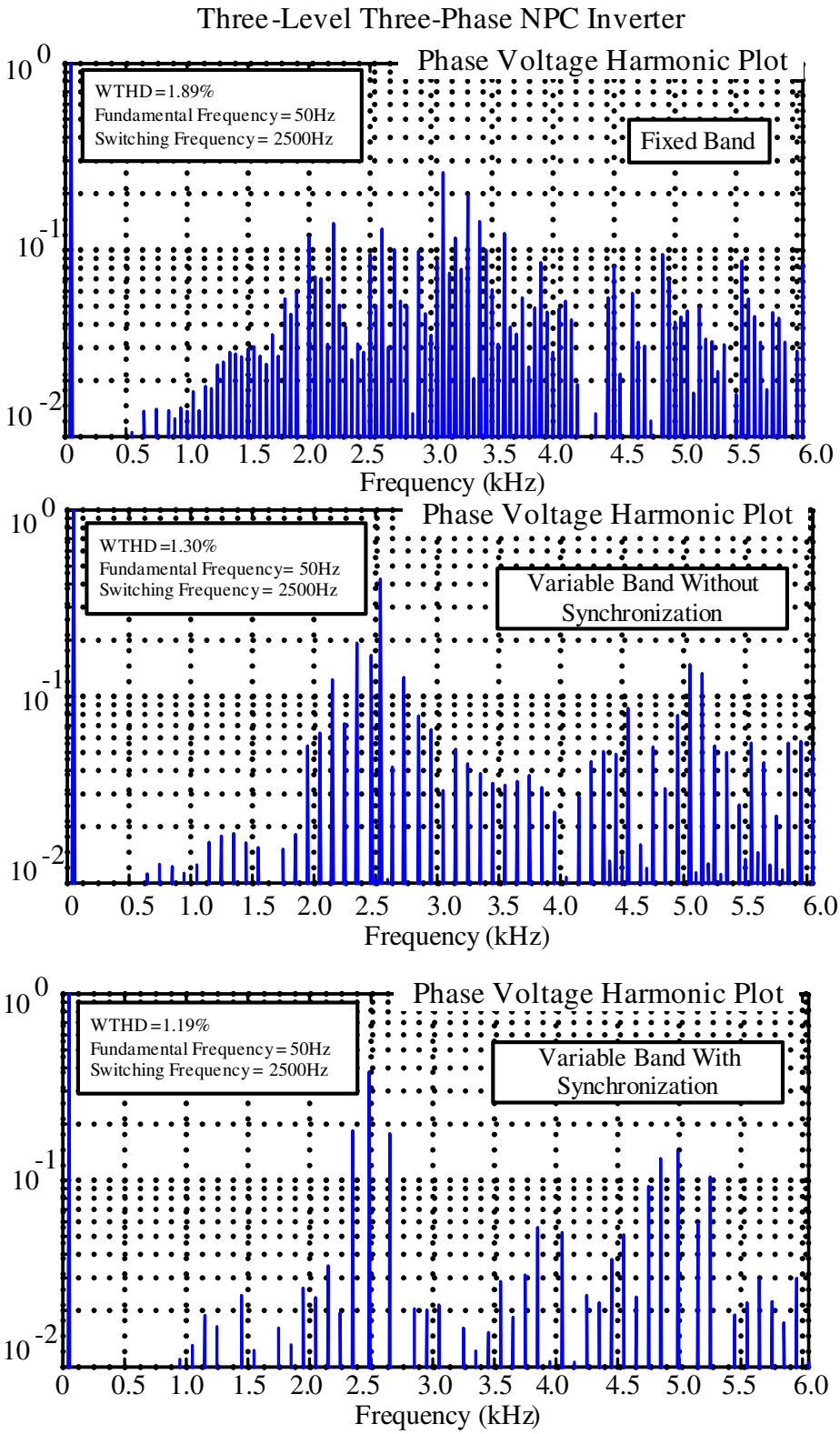


Figure 5.33: Phase leg voltage harmonics of the NPC inverter under hysteresis current regulation: (a) Fixed band (b) Variable band, (c) Fine-tuned variable band, (modulation depth = 0.9)

Experiment - Three-Level Neutral Point Clamped Inverter

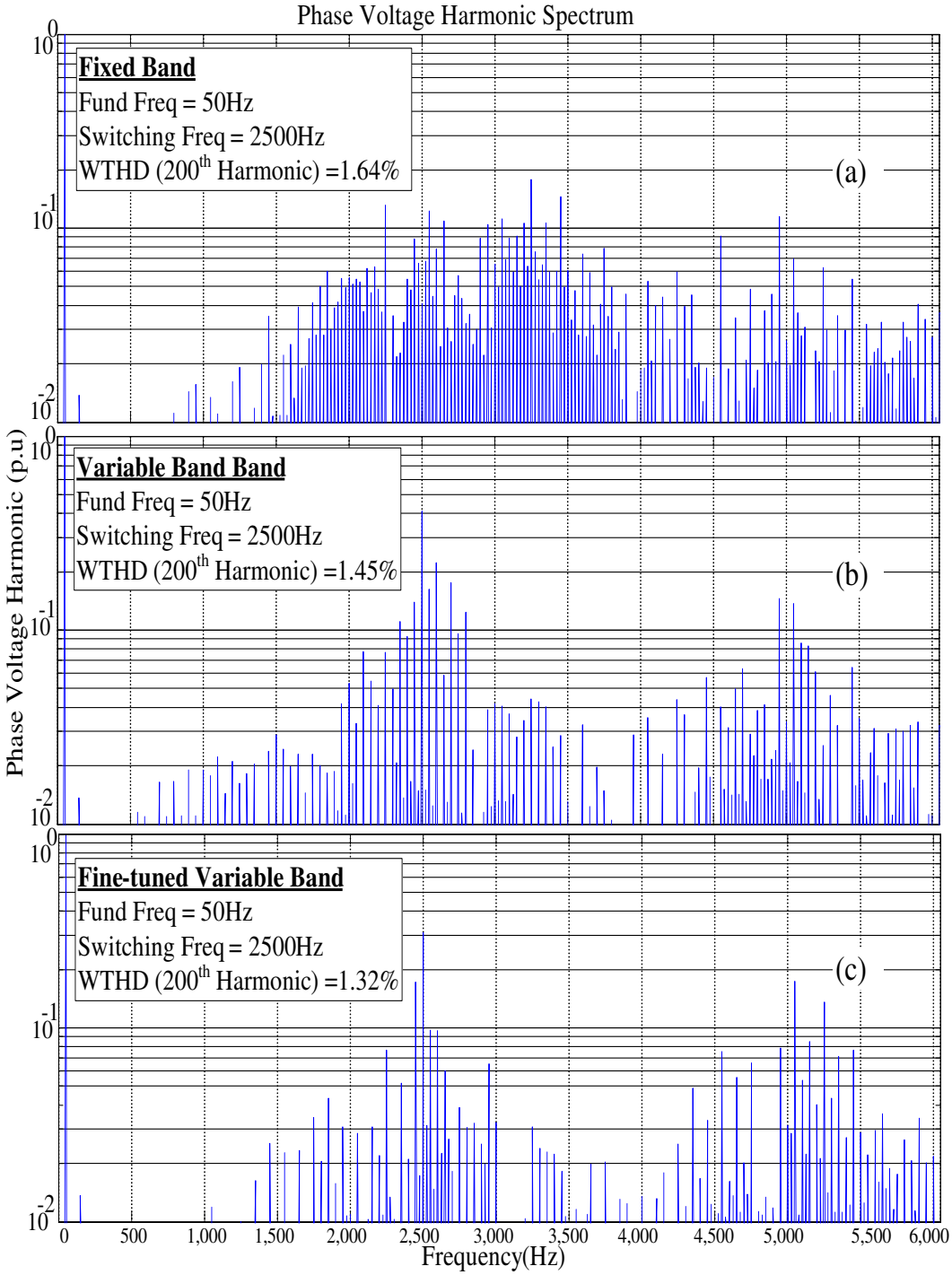


Figure 5.34: Phase voltage harmonics of the NPC inverter under hysteresis current regulation: (a) Fixed band (b) Variable band, (c) Fine-tuned variable band, Modulation depth = 0.9

Simulation - Three-Level Flying Capacitor Inverter

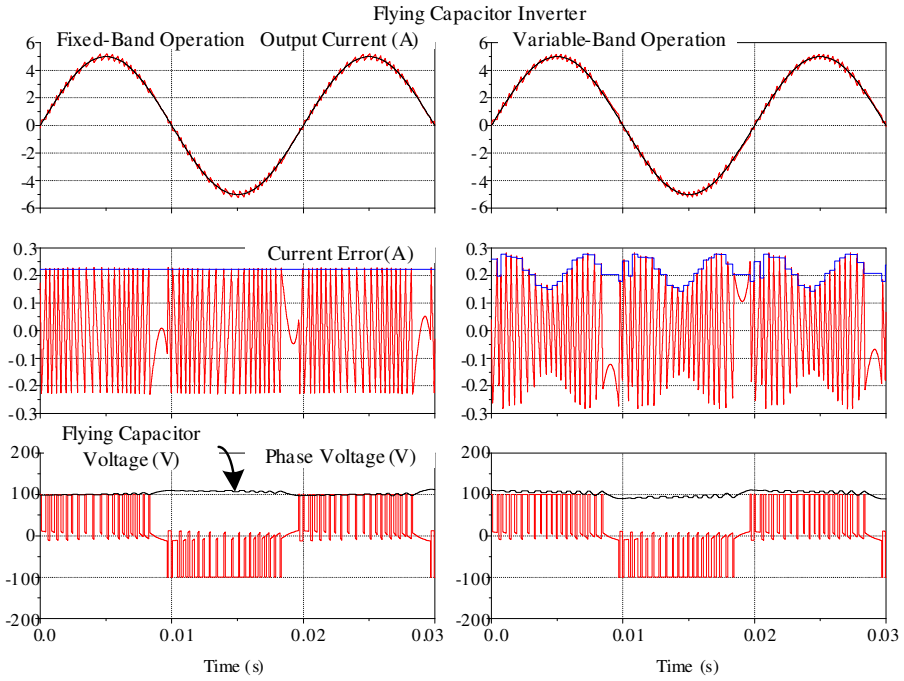


Figure 5.35: Single-phase leg simulation results for the new three-level hysteresis current regulator under fixed band and variable band operation (a) Output current (b) Current error and hysteresis band (c) three-level output voltage and FC voltage

Experiment- Three-Level Flying Capacitor Inverter

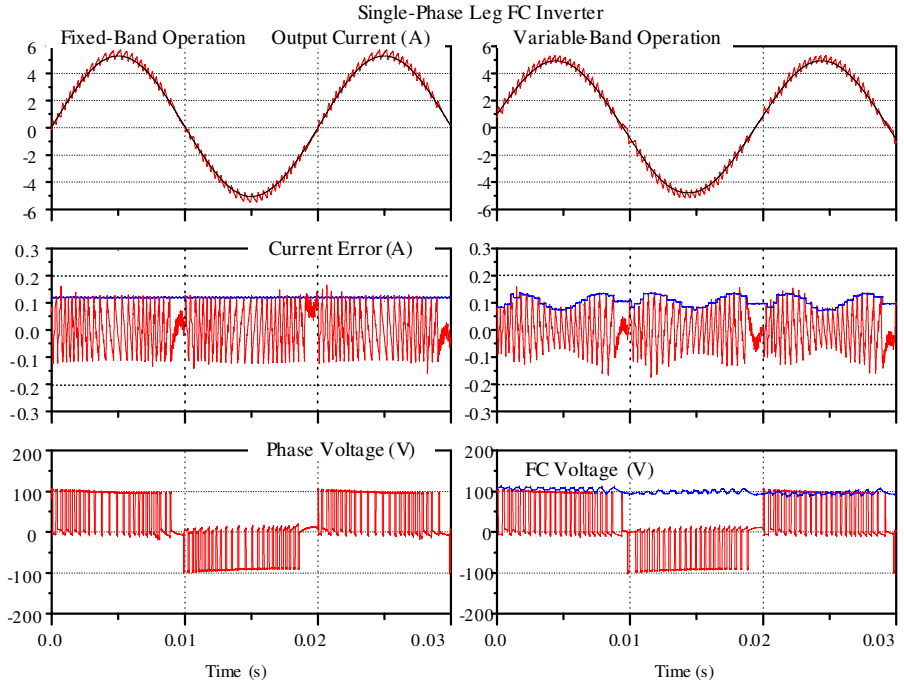


Figure 5.36: Single-phase leg experimental results for the new three-level hysteresis current regulator under fixed band and variable band operation (a) Output current (b) Current error and hysteresis band (c) three-level output voltage and FC voltage

Simulation- Three-Level Flying Capacitor Inverter

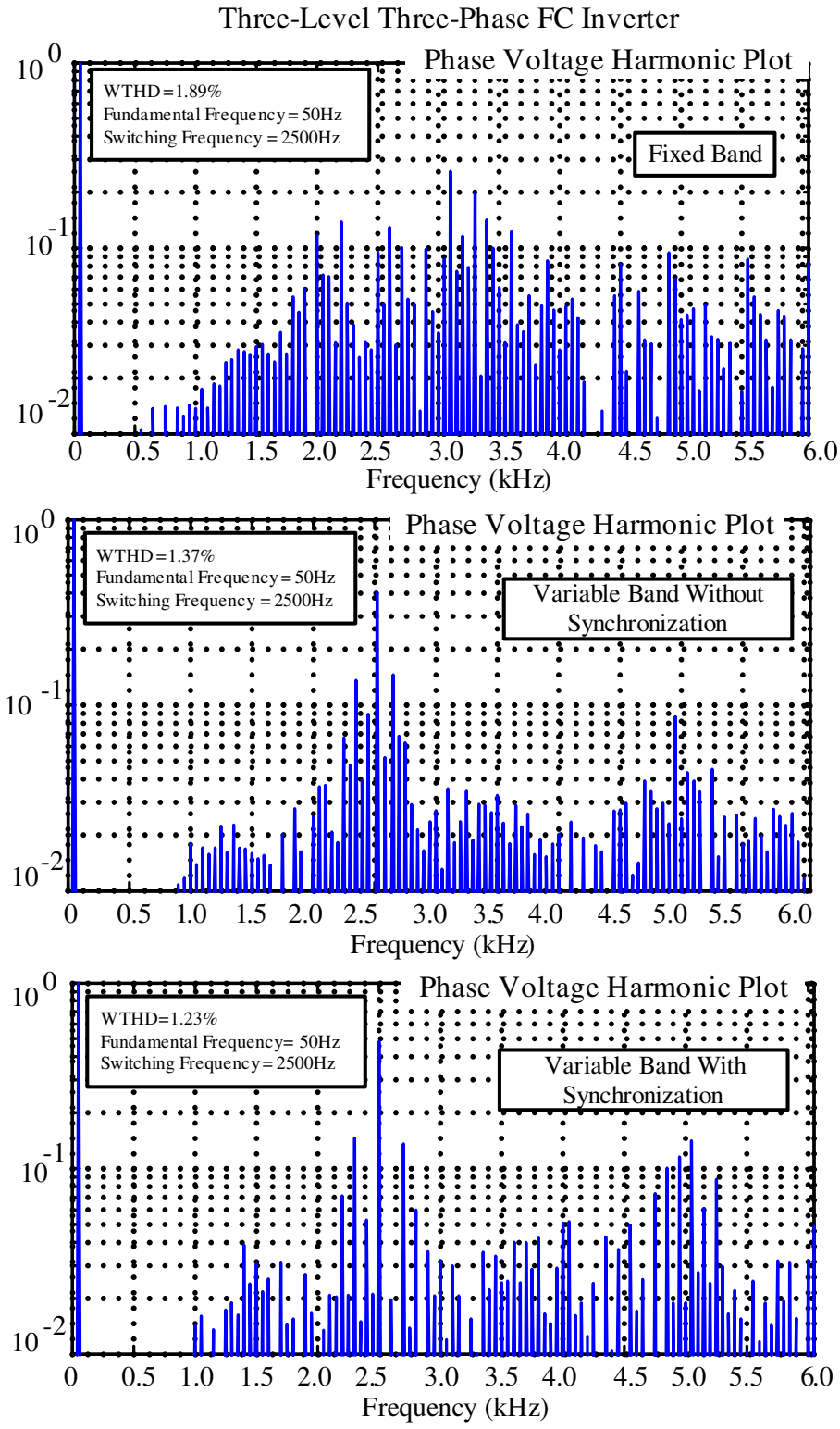


Figure 5.37: Phase leg voltage harmonics of the FC inverter under hysteresis current regulation: (a) Fixed band (b) Variable band, (c) Fine-tuned variable band, (modulation depth = 0.9)

Experiment- Three-Level Flying Capacitor Inverter

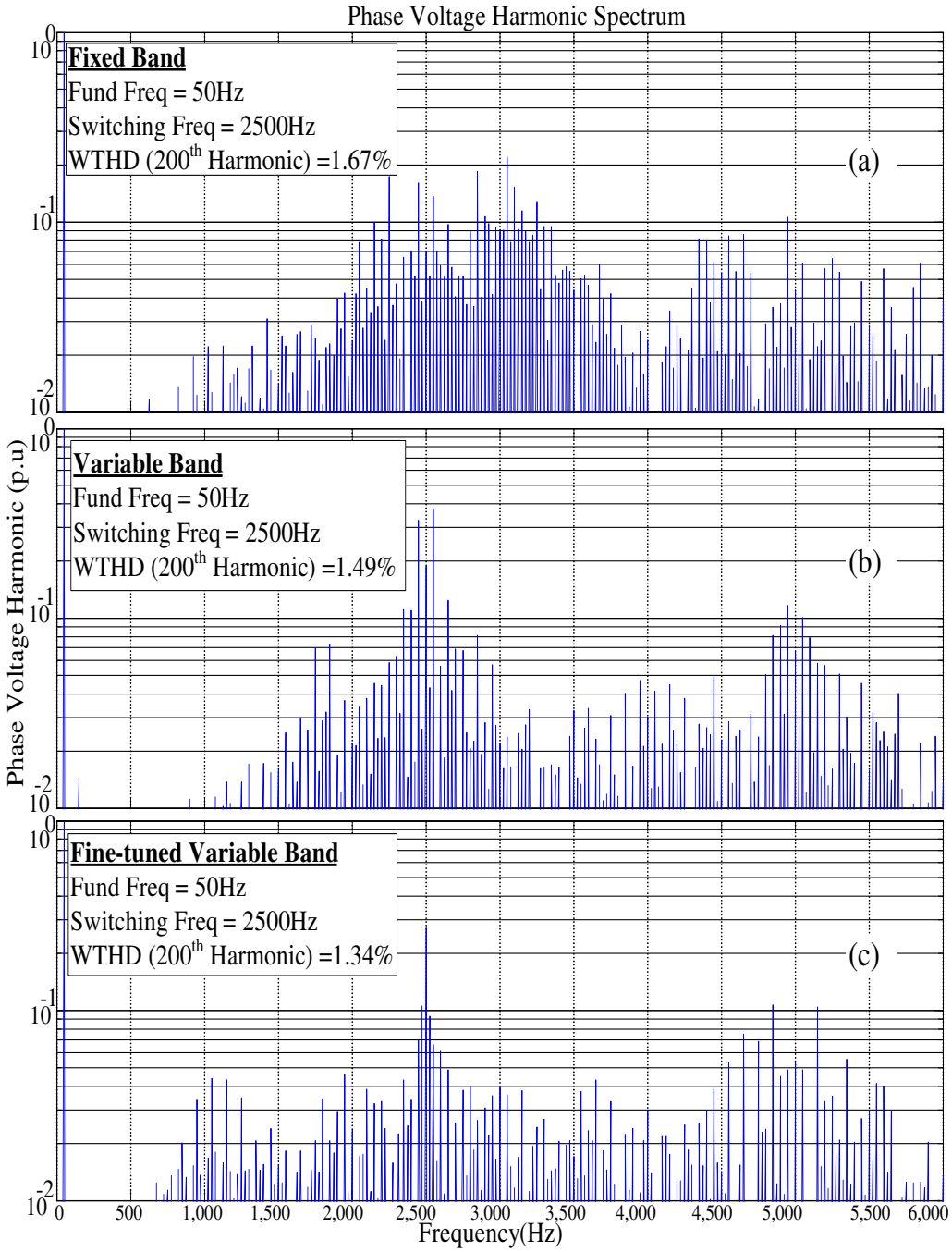


Figure 5.38: Phase voltage harmonics of the FC inverter under hysteresis current regulation: (a) Fixed band (b) Variable band, (c) Fine-tuned variable band, Modulation depth = 0.9

Simulation - Three-Level Single Phase H-Bridge

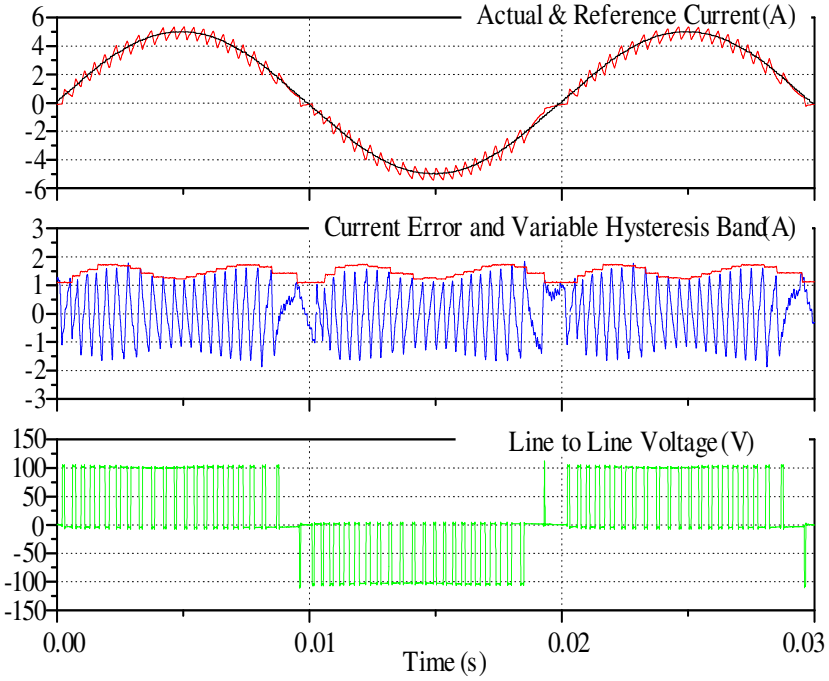


Figure 5.39: Single-phase leg simulation results for the new three-level hysteresis current regulator under fixed band and variable band operation (a) Output current (b) Current error and hysteresis band (c) three-level output voltage and FC voltage

Experiment- Three-Level Single Phase H-Bridge

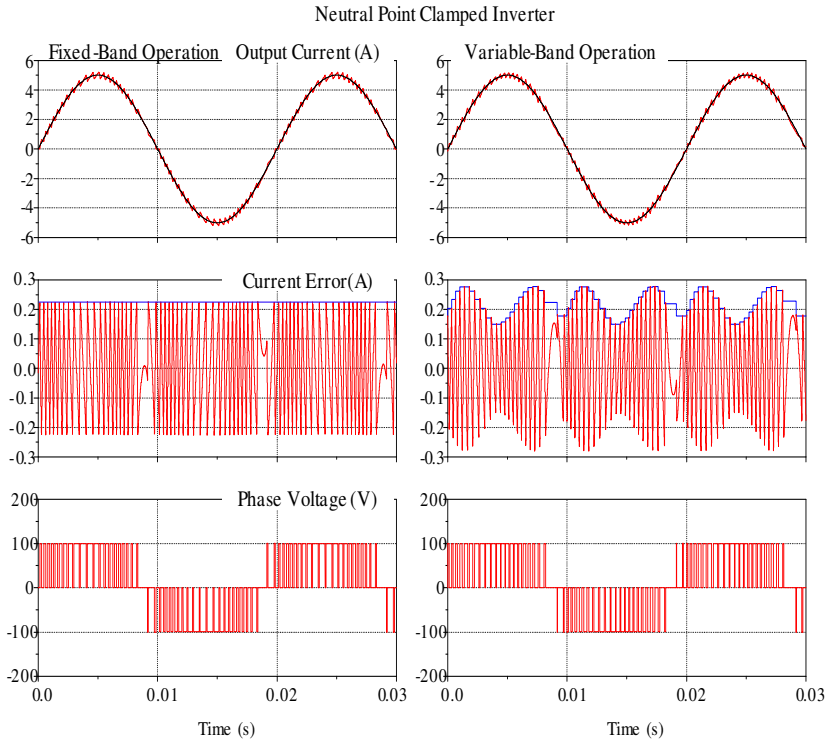


Figure 5.40: Single-phase leg experimental results for the new three-level hysteresis current regulator under fixed band and variable band operation (a) Output current (b) Current error and hysteresis band (c) three-level output voltage and FC voltage

Experiment- Three-Level Single Phase H-Bridge

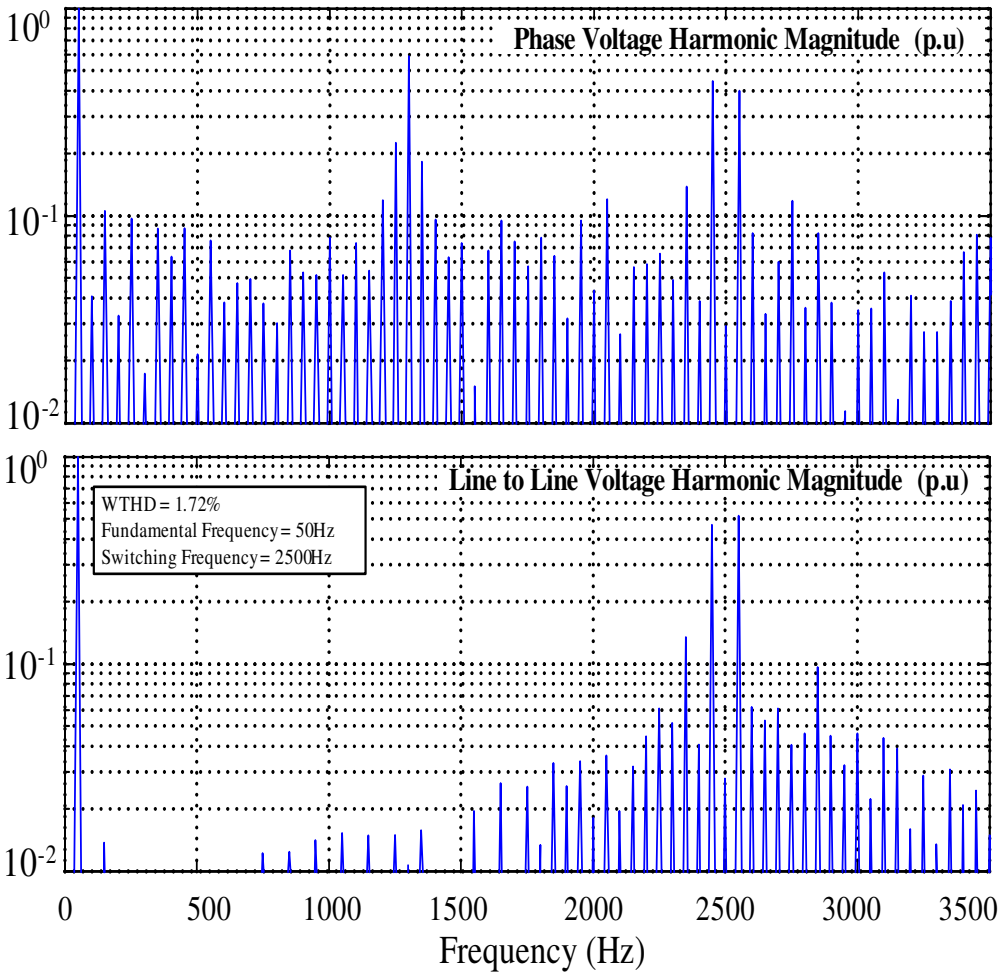


Figure 5.41: Harmonics performance of the three-level single phase H-bridge (a) phase leg voltage (b) line-line (output) voltage , Modulation depth = 0.9

5.7 Summary

This chapter has presented a new approach for hysteresis current regulation of a three-level single-phase VSI to maintain constant switching frequency. The three-level topologies used for this investigation were a single-phase H-bridge inverter, and a single-phase leg neutral point clamped (NPC), and flying capacitor (FC) inverter. The chapter extended the fundamental concepts of the two-level variable hysteresis band HCC as discussed in chapter 3, to the three-level VSI to develop a variable hysteresis band HCC with additional feedforward clock synchronisation.

Next, a novel technique was presented to detect the point of the phase voltage level change of the three-level VSI using the zero-crossing of the average inverter output voltage, so that only one hysteresis comparator is required per phase leg. This achieves more robust operation by eliminating steady state DC tracking errors and ensuring to select the appropriate output voltage level without requiring MB/MO hysteresis band arrangements and/or a current error derivative.

A logic decoder was developed to decode the switching signals of the single hysteresis comparator and the level change signal, to generate switching signals for the modulation of the NPC VSI. The pattern of the generated switching signals is similar to open-loop level shifted PWM. A finite state machine was implemented to generate the appropriate switching signals for modulation of the FC and H-bridge inverters using round robin assignment of the inverter zero state. The pattern of the generated switching signals is similar to open-loop phase shifted PWM.

The resultant performance achieved with this combined strategy is a harmonic performance similar to harmonically superior asymmetrical double-edge regular sampled PWM for a three-level single-phase H-bridge and naturally sampled PD-PWM for the three-level NPC and FC VSIs.

Finally, the operation of the controller was extended into the overmodulation region by clamping the variable hysteresis band to a minimum value in a similar way as presented in chapter 3. It was also found to be necessary to clamp the hysteresis band near the zero-crossing of the average inverter voltage to avoid high frequency switching within the level change region.

Simulation and experimental results for this system are included in this chapter to confirm the operation of the new hysteresis current regulation approach when applied to the three-level single-phase leg NPC, FC and single phase H-bridge VSIs.

Chapter 6

Hysteresis Current Regulation of a Three-Level Three-Phase Voltage Source Inverter

This chapter[1] applies the concepts of the variable band hysteresis controller presented in chapters 3, 4 and 5, to develop a new three-phase variable band hysteresis current regulator for a three-level VSI such as a NPC and FC inverter. The major extensions are:

- Section 6.2 applies the three-level variable hysteresis band concept of chapter 5 to a three-phase VSI by synchronously measuring the three-phase average inverter output voltages in a similar way as described in chapter 3. The common mode interacting current is then calculated and compensated of using hysteresis switching signals and the level change signals in a similar way as discussed in 4.2.2. The three-phase current error zero-crossings are synchronized to a fixed reference clock in a similar way as discussed in section 5.2.3, achieving with this combined strategy a performance that is very close to the harmonically superior open-loop PD PWM.
- Section 6.3 extends the operation of the proposed HCC into the overmodulation region in a similar way as described in section 3.4.
- Section 6.4 presents a novel active strategy to maintain a balanced neutral point voltage of the NPC inverter, by injecting the neutral point common mode current into the interacting current calculation.

[1] **Materials in this chapter were first published as**

1. Davoodnezhad, R.; Holmes, D. G.; McGrath, B. P.; "Three-Level Hysteresis Current Regulation for a Three Phase Neutral Point Clamped Inverter" *Power Electronics and Motion Control Conference (ECCE-EPE 2012*).
2. Davoodnezhad, R.; Holmes, D. G.; McGrath, B. P.; "Hysteresis current regulation of three phase flying capacitor inverter with balanced capacitor voltages," *Power Electronics and Motion Control Conference (IPEMC 2012).*

Simulation and experimental results are presented at the end of this chapter for various operating conditions under low and high pulse ratios.

6.1 Hysteresis Current Regulation of Three-level Three-Phase Voltage Source Inverters

This section develops the new constant frequency hysteresis current regulator for a three-level three-phase VSI, to achieve a harmonic performance that is very close to the open-loop PWM. The major differences between the development of the hysteresis current regulator for the three-phase three-level VSI and the single-phase leg three-level VSI in Figure 5.1 are as follows:

- Additional hysteresis current regulators and additional digital logic and finite state machine for phase legs B and C.
- Compensation for the common mode interacting current.
- Synchronisation of the three-phase current error zero-crossings to a fixed reference clock.
- Managing the regions of overmodulation and polarity change across all three phases.
- Balancing the neutral point voltage of the NPC inverter.

6.2 Extension to Three-Level Three-Phase VSI

Figure 6.1 shows the topology of a three-level three-phase neutral point clamped inverter and flying capacitor inverter feeding into a three-phase isolated series resistive and inductive load/filter with an AC back-EMF. From this figure, the load relationship for both the NPC and FC inverters can be written as:

$$V_x(t) = RI_x(t) + L\frac{dI_x(t)}{dt} + E_x(t) + U_0(t) \qquad x \in A, B, C \qquad (6.1)$$

Note that the only difference between equation (6.1) and the equivalent single-phase leg equation (5.1), is the additional load neutral point voltage $U_0(t)$.

Section 4.2.1 has presented mathematical analysis to express the load neutral point voltage in terms of the common mode interacting current. The common mode

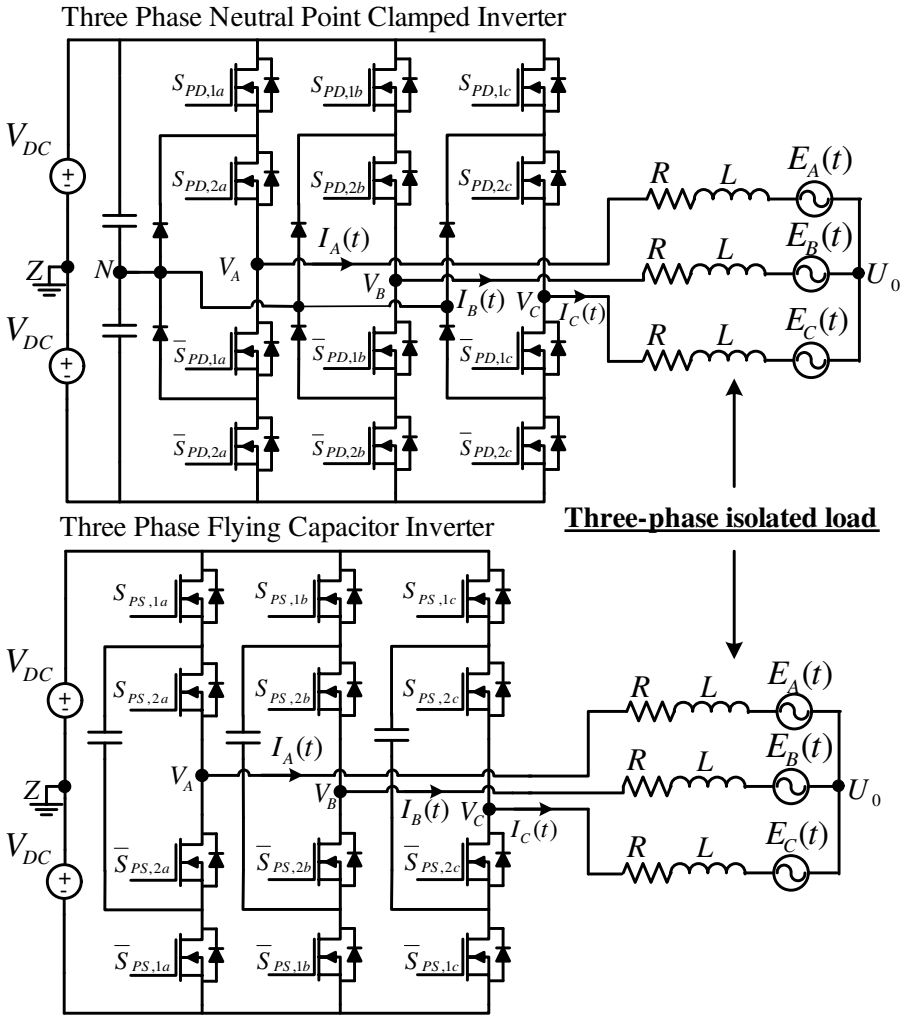


Figure 6.1: Topology of a three-phase NPC inverter and a three-phase FC inverter feeding a back-emf type load with series resistance and inductance

interacting current was then separated from the measured current for each phase and only the non-interacting current error component was used for the process of hysteresis comparison. Applying the same concept to the three-level three-phase VSI, the load equation can be restated as:

$$V_x(t) = RI^s_{x,3L}(t) + L\frac{dI^s_{x,3L}(t)}{dt} + E_x(t) \qquad x \in A,B,C \qquad (6.2)$$

if the load neutral point voltage is equated to $U_0(t) = -L\frac{d\gamma_{3L}(t)}{dt} - R\gamma_{3L}(t)$

where $I^s_{x,3L}(t) = I_x(t) - \gamma_{3L}(t)$ is the non-interacting component of the phase current.

In this form, equation (6.2) is identical to a single-phase hysteresis controlled system if the interacting current $\gamma_{3L}(t)$ is subtracted from the measured current before the hysteresis control process.

Using the same mathematical analysis as presented in 5.2.1 for the three-level single-phase leg VSI, the per phase switching ripple component $I^{s}_{x,r,3L}(t)$ can be separated into two switching time events over the positive and negative fundamental period to achieve a time varying three-level hysteresis band expression, as follows:

$$I_{x,h} = I_{h\_\max}\left|V_{x,avg}(t)/V_{DC}\right|\left(1-\left|V_{x,avg}(t)/V_{DC}\right|\right) \qquad x \in A,B,C \qquad (6.3)$$

where $I_{h,\max} = \dfrac{V_{DC}}{4Lf_{sw}}$ is the maximum allowable hysteresis band.

In this form, the three-phase hysteresis current regulator has three separate controllers, each of which compare their phase leg output current against a commanded reference. Each converter phase leg switches between 0 and $+V_{DC}$ or $-V_{DC}$ when the corresponding current error crosses the lower or higher (respectively) variable hysteresis band $I_{x,h}(t)$, depending on the polarity of the per phase average inverter voltage. The three-phase average inverter output voltages can also be measured in a similar way as described in section 5.2.2, using the modified hysteresis phase leg switching signals as follows:

$$S_{x,OR,avg}(t) = \left(\frac{T_{x,2}-T_{x,1}}{T_{x,3}-T_{x,1}} - 0.5\right) \qquad x \in A,B,C \qquad (6.4)$$

$$V_{x,avg}(t) = 2V_{DC} * S_{x,OR,avg}(t) \qquad x \in A,B,C \qquad (6.5)$$

where $T_{x,1}, T_{x,2}, T_{x,3}$ are the per phase switching instances shown in Figure 5.3.

6.2.1 Compensation of the Common Mode Interacting Current

6.2.1.1 Common Mode Current in Three-Level Three-Phase VSIs

Similar to the hysteresis current regulation of the two-level three-phase VSI presented in chapter 4, the next step is to calculate the common mode interacting current. From the mathematical analysis presented in section 4.2.2.1, the common mode interacting current can be directly calculated from the inverter three-level three-phase switched outputs as follows:

$$\gamma_{3L}(t) = -\frac{1}{3L}\int [V_A(t) + V_B(t) + V_C(t)]dt \tag{6.6}$$

Now, if this common mode current is subtracted from each phase's measured current, the new variable hysteresis band concepts developed in chapter 5 can be immediately applied to the three phase VSI.

6.2.1.2 Practical Implementation of Compensating the Common Mode Current

From equation (6.6), it is necessary to measure the three-level three-phase switched phase voltages to calculate the common mode interacting current. Section 4.2.2 developed a technique to calculate and compensate the common mode current using the three-phase gate signals since their switching patterns are essentially identical with their corresponding switched phase voltage.

The only difference for a three-level phase voltage is that it is necessary to know when the polarity of average inverter output voltage changes. As a result, the same technique can be readily extended to a three-level three-phase VSI to calculate the common mode interacting current. The three-level switched phase voltage is constructed with appropriate scaling and level shifting using the following information:

1) Three-phase modified hysteresis switching signals (feedback OR gate switching signals $S_{a,OR}(t), S_{b,OR}(t), S_{c,OR}(t)$): These are equivalent to the absolute normalized three-level three-switched phase voltages.
2) Three-phase level selection signals ($R_a(t), R_b(t), R_c(t)$): These signals identify when the switching pattern of the three-phase modified hysteresis switching signals belong to the switching group (0,+V_{DC}) or the switching group (0,-V_{DC}) of the per phase three-level output voltage.

Figure 6.2 shows the block diagram of the proposed strategy to calculate the common mode interacting current from the three-phase modified hysteresis switching signals ($S_{a,OR}(t), S_{b,OR}(t), S_{c,OR}(t)$) and the three-phase level selection signals ($R_a(t), R_b(t), R_c(t)$).

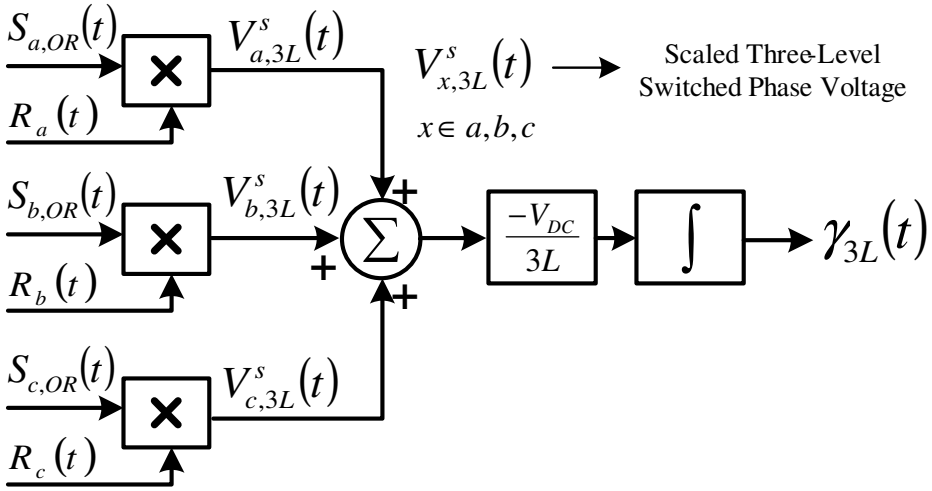


Figure 6.2: Measuring the common mode interacting current from the instantanous switching states of the three-phase modified hysteresis switching signals and three-phase level selection signals

During the time when the level selection signal is HIGH (+1), the corresponding scaled phase leg switched voltage is equivalent to the modified hysteresis switching signal and the average inverter voltage is positive.

During the time when the level selection signal is LOW (0) as per chapter 5, the corresponding scaled phase leg switched voltage is equivalent to the complement of the modified switching signal and the average inverter voltage is negative. This can be mathematically expressed as follows:

$$V_x(t) = +V_{DC} * \{2R_x(t) - 1\} * S_{x,OR} \qquad x \in A,B,C, \quad S_x \in 0,1, \quad R_x \in 0,1 \qquad (6.7)$$

where:

$V_x(t)$ is the three-level phase-leg switched voltage.

V_{DC} is the half of the total DC bus voltage.

$S_{x,OR}(t)$ is the phase modified hysteresis switching signal.

$R_x(t)$ is the phase level selection signal.

From equation (6.7), the common mode interacting current expression in (6.6) can now be stated as:

$$\gamma_{3L}(t) = -\frac{V_{DC}}{3L} \int \left[\begin{matrix} (R_a(t) * S_{a,OR}(t)) + (R_b(t) * S_{b,OR}(t)) \\ + (R_c(t) * S_{c,OR}(t)) \end{matrix} \right] dt \qquad \begin{matrix} S_x \in 0,1 \\ R_x \in -1,1 \\ x \in a,b,c \end{matrix} \qquad (6.8)$$

Equation (6.8) shows that the common mode interacting current can be easily measured using the three-phase modified switching signals and the level selection signals with appropriate gain and offset adjustment, as shown in Figure 6.2. Furthermore, the three-phase modified switching signals have already been used to measure the three-phase average inverter output voltages for calculation of the variable hysteresis bands. Hence, there is no need for additional voltage sensors to measure the three-switched phase voltages to calculate the common mode current.

Figure 6.3 provides simulation results to confirm the performance of compensating the common mode current in this way. Figure 6.3 (a)(b)(c) show the modified hysteresis switching signals and the level selection signals. Figure 6.3 (d)(e)(f) show the reconstituted three-phase three-level phase voltages. Figure 6.3(g) shows the phase A current error before compensation of the common mode current. As expected, the current error has a non-uniform switching behaviour caused by the common mode current.

Figure 6.3(h) shows the common mode interacting current calculated from equation (6.6) using the block diagram in Figure 6.2. Figure 6.3(i) shows the resultant non-interacting components of the phase A current error after common mode compensation, with the ripple caused only by the per phase leg switching processes. The figure confirms that calculation of the common mode current from equation (6.8) ensures to achieve a uniform triangular phase current error which can then be used for hysteresis comparison against a set of variable hysteresis bands to generate the per phase hysteresis switching signals. It also can be seen from this figure how the per phase current error freewheeling behaviour still occurs near the region of the output voltage level change with the proper compensation of the common mode current, and is then used by the DSP to generate the level selection signal.

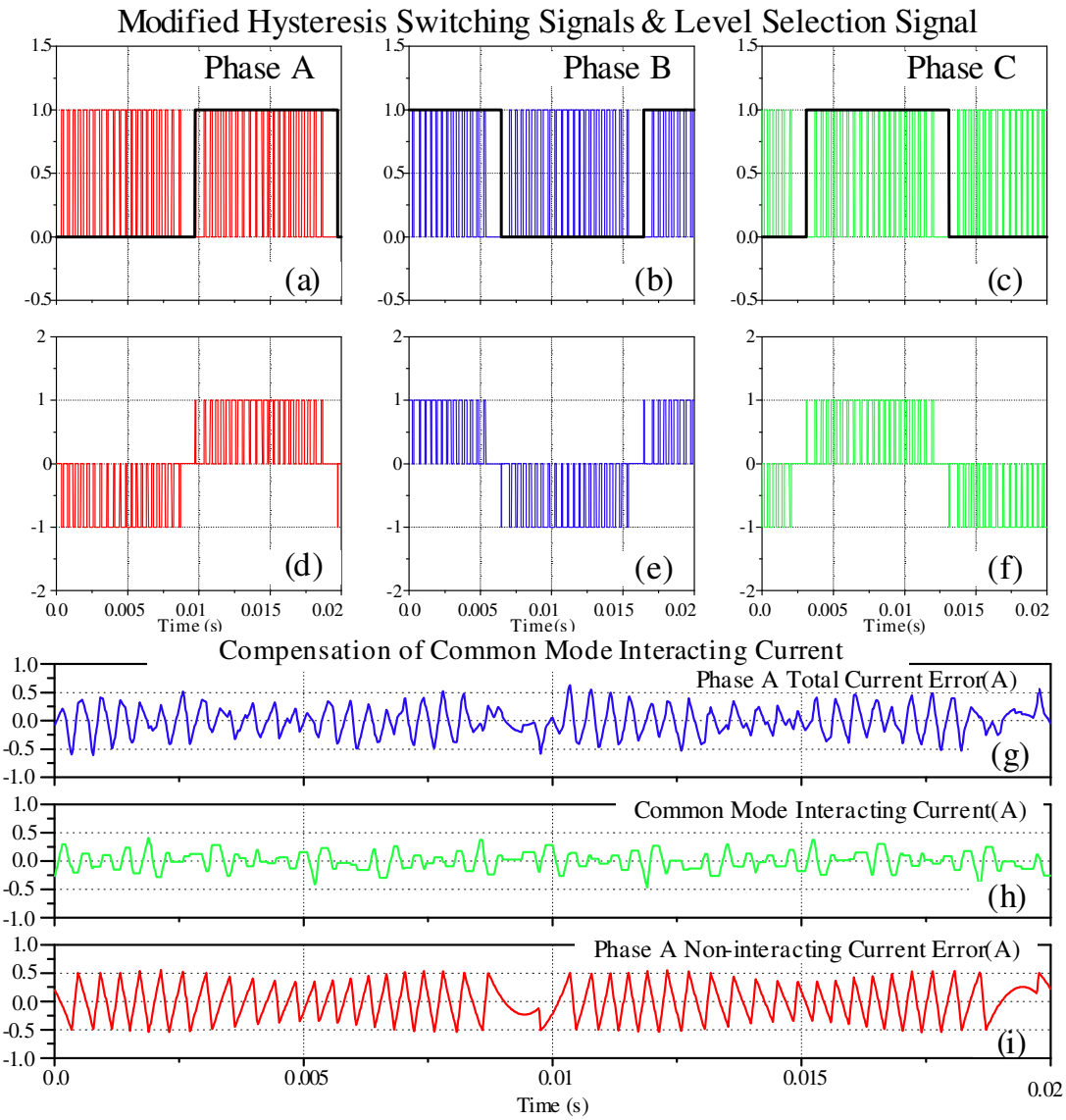


Figure 6.3: Simulation results for the ccompensation of the common mode interacting current for three-level three-phase VSIs. (a)-(c) three-phase modified hysteresis switching signals and level selection signals (d)-(f) reconstituted three-switched phase voltages (g) phase A total current error (h) common mode interacting current (i) phase A non-interacting current error (switching ripple component)

6.2.2 Synchronisation to a Fixed Reference Clock

Previous work by McGrath et. al [15][24][34] has shown that open-loop PD PWM ensures to achieve superior line-to-line harmonic performance in comparison to multilevel sine-triangle PWM such as PSC, POD and APOD strategies. This is because PD PWM places more harmonic energy into the carrier component of each phase leg and less energy into the sideband harmonics. Since this carrier harmonic is common mode across all three-phases, it will be cancelled and eliminated between the phase legs, while the side band harmonics do not cancel in the line-to-line harmonic spectra.

This has been further illustrated in Figure 6.4 where it shows the five-level line-to-line waveform for open-loop PD PWM strategy. The figure confirms that using PD PWM achieves five discrete line to line voltages levels where the switching groups (0,+V_{DC}), (+V_{DC},+2V_{DC}), (0,-V_{DC}) and (-V_{DC},-2V_{DC}) stay precisely within their boundary.

From this discussion, it can be recognised that three-phase open-loop PD equivalent PWM and its optimal harmonic performance are the target switching objectives for the new three-level three-phase variable band hysteresis current regulation approach. Figure 6.5 shows the synchronization process of current error zero-crossing with its active pulse placement over one switching period for fixed frequency PD PWM. The three-phase current errors cross zero at the peak and trough of the modulating triangular carrier waveform.

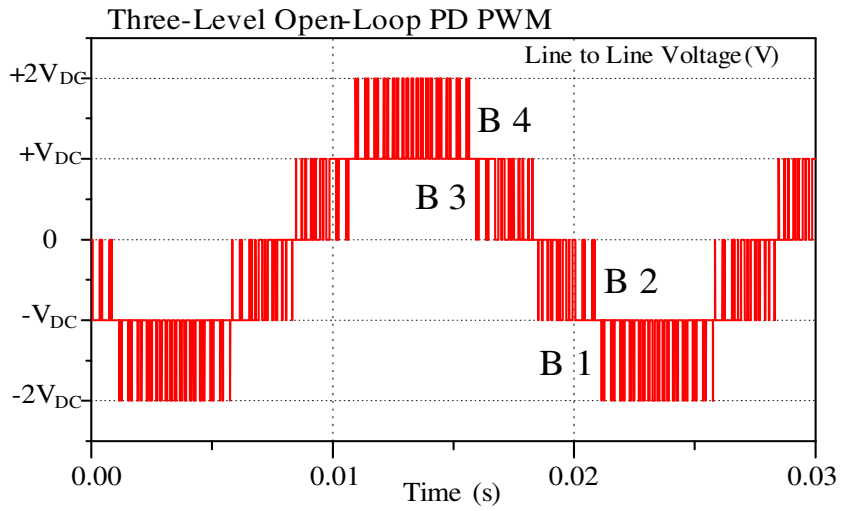


Figure 6.4: Line voltage waveform using (Top) open-loop POD PWM (Bottom) open-loop PD PWM

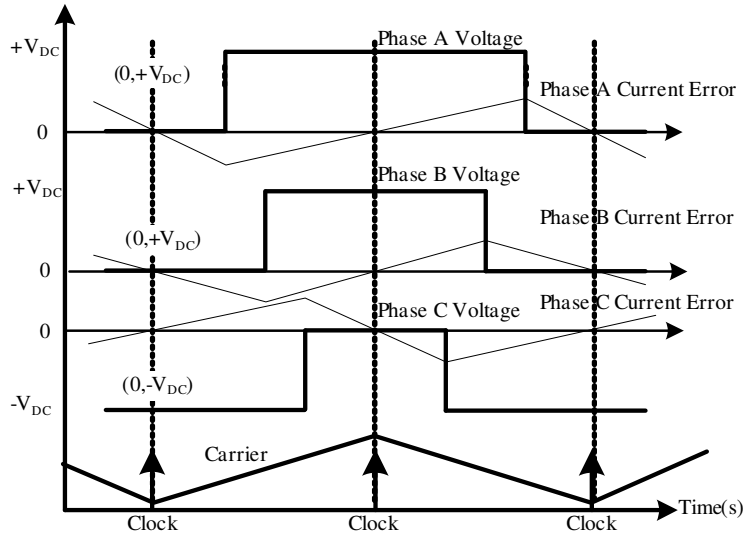


Figure 6.5: Balanced three phase current error zero crossings with centered three phase active pulses

Section 3.3 has already developed the fundamentals of synchronising the current error zero-crossing to a fixed reference clock as an additional feedforward step without directly measuring the current error zero-crossings. Section 5.2.3 applied this concept to the three-level VSIs where the synchronization was only implemented during the switching regions and it was suspended during the polarity change region.

This same synchronisation process can be readily applied to the three-phase VSI by calculating the appropriate band offset per phase leg, as follows:

$$I_{x,h,new} = \delta I_{x,h} + I_{x,h,old} \qquad x \in A,B,C \qquad (6.9)$$

where $I_{x,h,new}$ is the new calculated phase leg variable hysteresis band.

Figure 6.6 and Figure 6.7 confirm the simulated performance of the synchronization process showing the synchronised three-phase current error zero-crossings and the five-level PD equivalent line voltage.

In Figure 6.6, the variable hysteresis band is calculated using the actual inverter output voltage to eliminate the effect of sampling on the average inverter output voltage measurement. This figure confirms the achievement of five distinct output voltage levels with minor transitions between the switching groups ($+V_{DC}$,$+2V_{DC}$) and ($-V_{DC}$,$-2V_{DC}$). As discussed in section 5.2.3, these minor transitions are caused by the half cycle switching cycle delay until the current error is synchronized again with the DSP reference clock after the polarity change region.

Figure 6.7 shows the synchronization process considering the effect of sampling on the average inverter output voltage measurement, which is the case for the practical system. From this figure, it can be seen that there are more level transitions between the adjacent voltage levels. These level transitions are caused by three factors in the practical implementation of the proposed HCC, as follows:

- The sampling effect on the measurement of the average inverter output voltage. This becomes important when calculating the per phase variable hysteresis band using (6.3) which uses the assumption of constant average voltage over one switching cycle.
- The common mode interacting current which prevents the phase current error from achieving an absolute triangular form.
- The phase current error zero-crossing synchronizing to the DSP clock approximately two switching cycles after the polarity change region.

Considering these effects, the synchronization process still achieves a line-to-line voltage that is very close to open-loop PD PWM. The result is a significant improvement compared to the previously reported three-phase multilevel space vector based HCC [115][117][118][114].

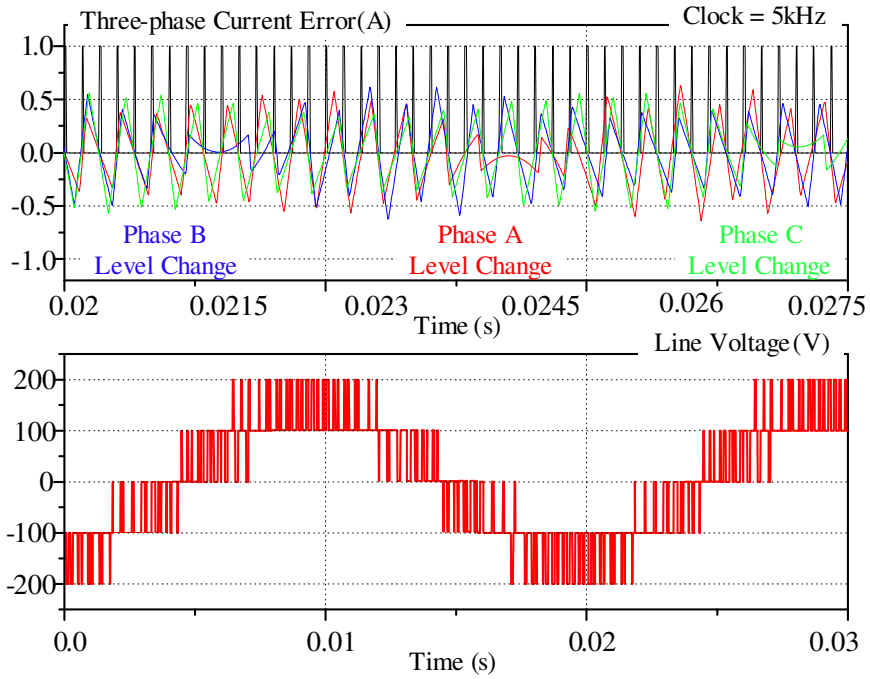


Figure 6.6: Balanced three phase current error zero crossings with PD equivalent line voltage, considering effect of sampling on average inverter output voltage for variable band calculation, Modulation Depth = 0.9

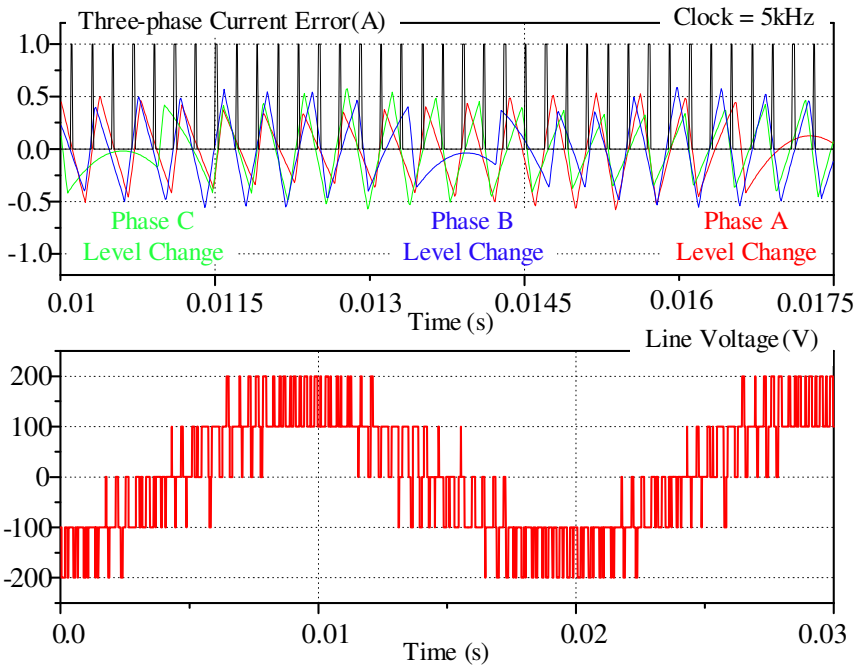


Figure 6.7: Balanced three phase current error zero crossings with PD equivalent line voltage, ignoring effect of sampling on average inverter output voltage for variable band calculation, Modulation Depth = 0.9

6.3 Band Clamping in a Three-Level Variable Hysteresis Band Operation

Section 5.5 has identified that for a three-level variable band hysteresis current regulator it is necessary to clamp the variable hysteresis band near two regions:

3) At the peak of the average inverter output voltage to avoid controller instability during overmodulation.
4) At the zero-crossings of the average inverter voltage to improve the inverter switching performance near the polarity change region.

Section 5.5 then proposed how to manage these constrains by clamping the variable hysteresis band at a minimum value about 20% of the maximum allowable hysteresis band.

The same concept can be immediately applied to a three-level three-phase VSI using the proposed hysteresis current regulator, where each phase leg clamps their corresponding variable hysteresis band as the VSI enters these regions.

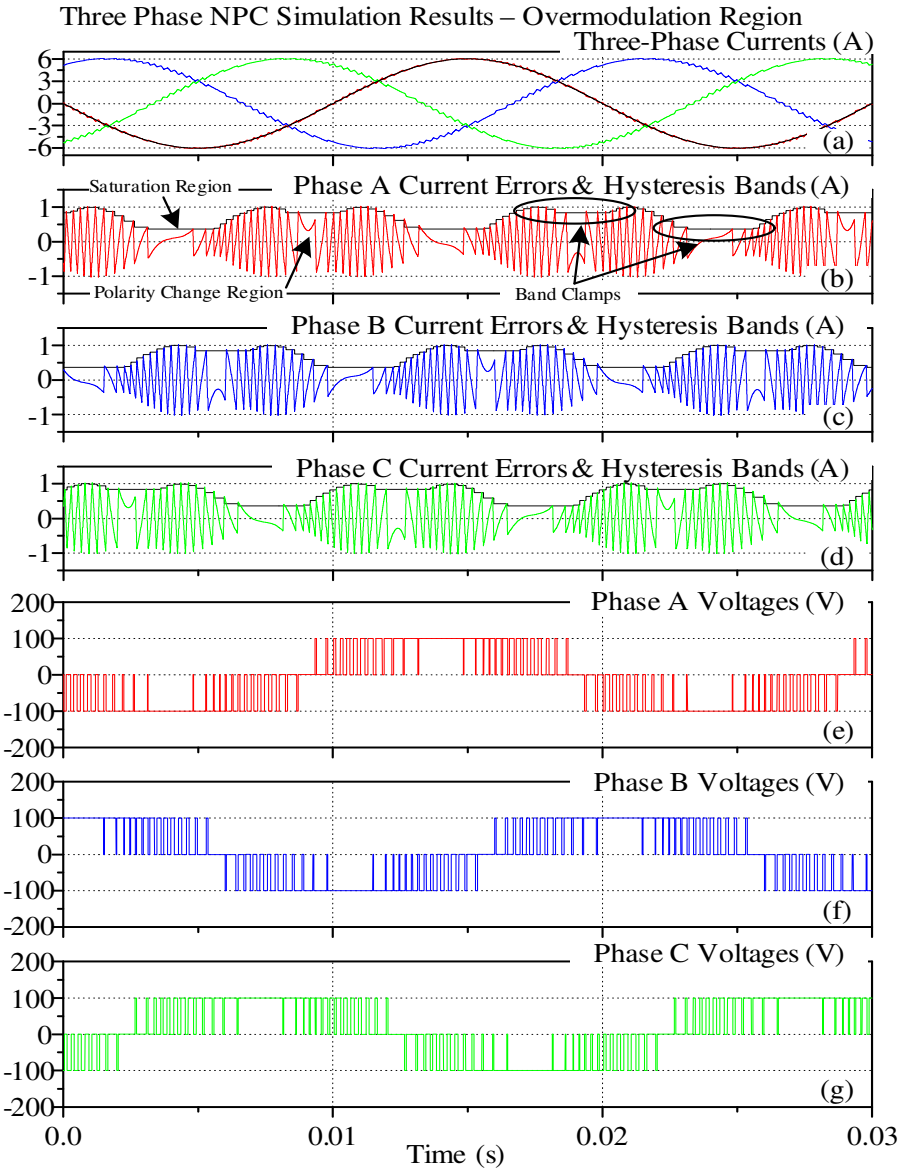


Figure 6.8: Simulated performance of three-level three-phase variable band hysteresis current regulation with hysteresis band clamping, peak average inverter output voltage = $1.2*V_{DC}$

Figure 6.8 confirms the validity of the band limit clamping technique for a three-level three-phase NPC VSI. From Figure 6.8(a), it can be seen that the multilevel three-phase hysteresis regulators smoothly progress into overmodulation. From this figure, it can also be seen how the regulators still track the reference currents even in the nonlinear overmodulation region, with no sign of transient disturbance as the inverter enters and leaves this region. Figure 6.8(b)(c)(d) show the three-phase current errors and the variable hysteresis bands. It can be seen that the controller can easily differentiate between the overmodulation region and the level change region since overmodulation occurs at the maximum peak of the average inverter voltage

while polarity change occurs when the average inverter voltage is near zero. In either case, the VSI switching recommences after the per phase current error returns to within the hysteresis bands. Figure 6.8(e)(f)(g) show the three-phase switched voltages. These figures confirm that the controller maintains stability during the overmodulation region without any sign of high frequency switching.

6.4 Neutral Point Voltage Balancing of the NPC Inverter

Recent work by Mohzani. et. al. [28] presented an analysis that describes the natural balancing behaviour of the NP voltage in terms of the time constants of each switching harmonic. This work showed that constant frequency open-loop PD PWM significantly improves the natural NP voltage balancing response.

The proposed three-level hysteresis current regulator with three-phase synchronization of the current error zero-crossings achieves a similar level of natural balancing of the NP voltage since it also maintains a constant switching frequency. Figure 6.9 provides experimental confirmation of the natural balancing response of this system, showing a slow restoration of the neutral point voltage of the NPC inverter to zero volts over a 20 second period from an initial unbalanced condition (the unbalanced clamp is released at 2.0 seconds). The response is determined by the modulation depth, DC bus capacitor values, and the available natural driving

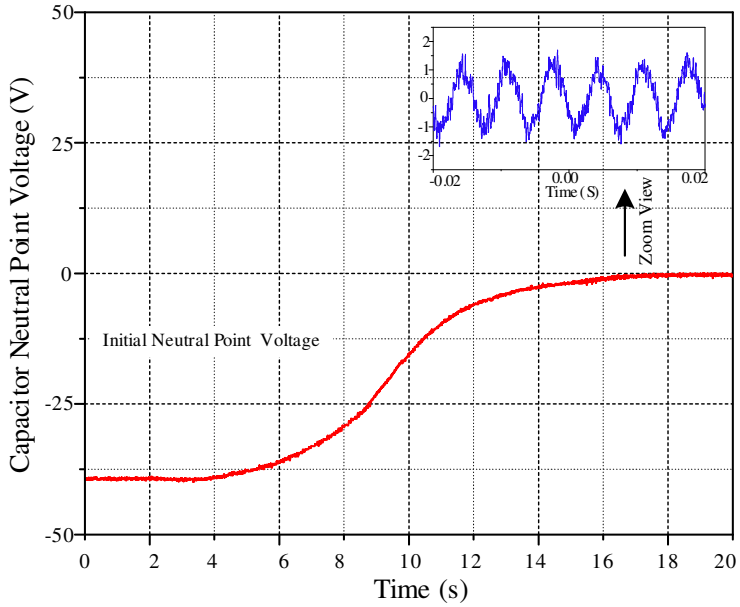


Figure 6.9: Experimental performance of the natural balancing of NPC neutral point voltage (without active balancing strategy)

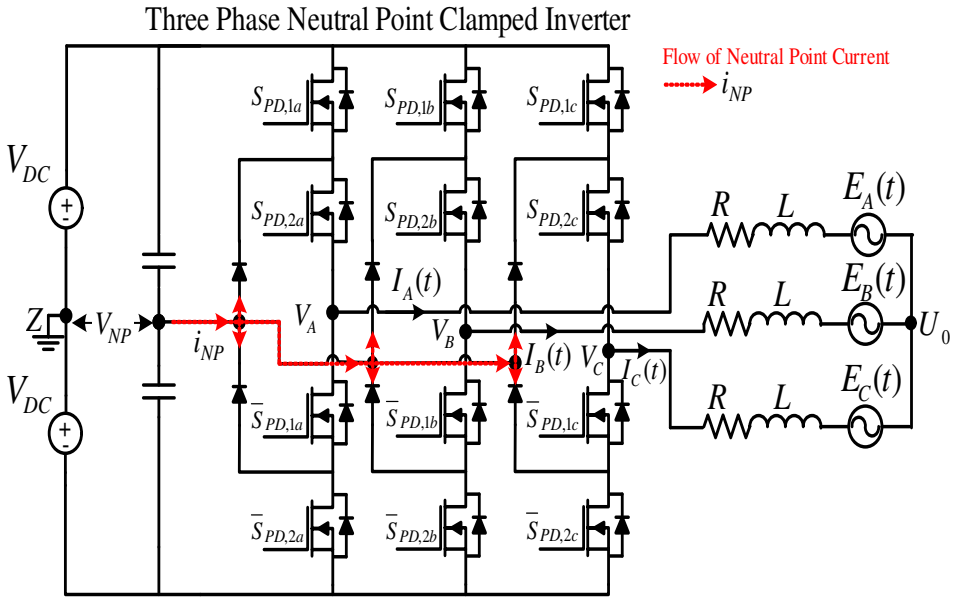


Figure 6.10: Topology of a three-level three-phase NPC VSI showing the flow of the common mode current generated by the bus capacitors neutral point voltage

response. It is essentially the same as the theoretical and experimental balancing response presented in [28] for open PD PWM, confirming the close match to PD PWM achieved by the variable band hysteresis regulation strategy.

In order to improve the balancing response of the NP voltage, a novel active balancing strategy is now proposed. Figure 6.10 illustrates a three-level three-phase NPC inverter that shows how the capacitor neutral point current flows through the per phase series connected diodes. This current is essentially generated by the neutral point voltage of the bus capacitors, which is a common mode voltage across all three phases. Hence the three-phase load relationship as per equation (6.1) can be restated to include this effect as:

$$V_x(t) = RI_x(t) + L\frac{dI_x(t)}{dt} + E_x(t) + U_0'(t) \qquad x \in A,B,C \qquad (6.10)$$

where $U_0'(t)$ is the modified load neutral point voltage expression and can be restated as follows:

$$U_0'(t) = \underbrace{\frac{V_A(t)+V_B(t)+V_C(t)}{3}}_{U_0(t)} + V_{NP}(t) \qquad x \in A,B,C \qquad (6.11)$$

where $V_{NP}(t)$ is the neutral point voltage.

Now using a similar mathematical analysis as presented in section 6.2, the new modified common mode interacting current $\gamma'_{3L}(t)$ can be expressed as:

$$\gamma'_{3L}(t) = -\frac{1}{3L}\int[(V_A(t)+V_B(t)+V_C(t))+3V_{NP}(t)]dt$$
$$\gamma'_{3L}(t) = \gamma_{3L}(t)+i_{NP}(t) \tag{6.12}$$

Equation (6.12) shows how the common mode NP voltage can be integrated in a similar way as the three-phase voltages to calculate the common mode current caused by the NP voltage. In an ideal balanced condition, the NP voltage is zero and hence the NP current is zero. However as this voltage starts to drift from zero, the generated NP current is subtracted from the total phase leg current and forces the proposed hysteresis regulator to generate the appropriate switching signals to restore the NP voltage back to zero.

Figure 6.11 shows the implementation block diagram of the active balancing strategy where the scaled neutral point voltage is added to the individual scaled three phase voltages before calculating the modified common mode interacting current.

Figure 6.12 shows the experimental NP voltage balancing response achieved using this new active balancing strategy, where once again the unbalanced voltage clamp is released at 2 seconds. From this figure, it can be clearly seen that the balancing response is significantly improved, becoming nearly 12 times faster in comparison to natural balancing response of the NP voltage.

On reflection, it can be seen how implementing the active NP balancing strategy in this way is similar to the injection of a third harmonic voltage into the common mode interacting current as described in section 4.3.1.2, again recognising that all of these voltage terms are common mode effects across all three-phase voltages. From this understanding it can be concluded any common mode effect can be incorporated into the hysteresis modulation process by adding it to the common mode interacting current in a similar way as the common mode neutral point voltage $U_0(t)$.

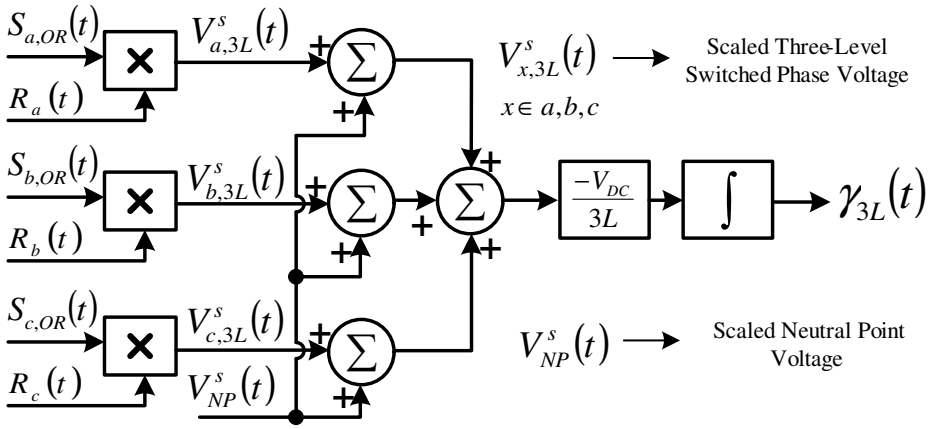


Figure 6.11: Block diagram of implementing the active balancing strategy

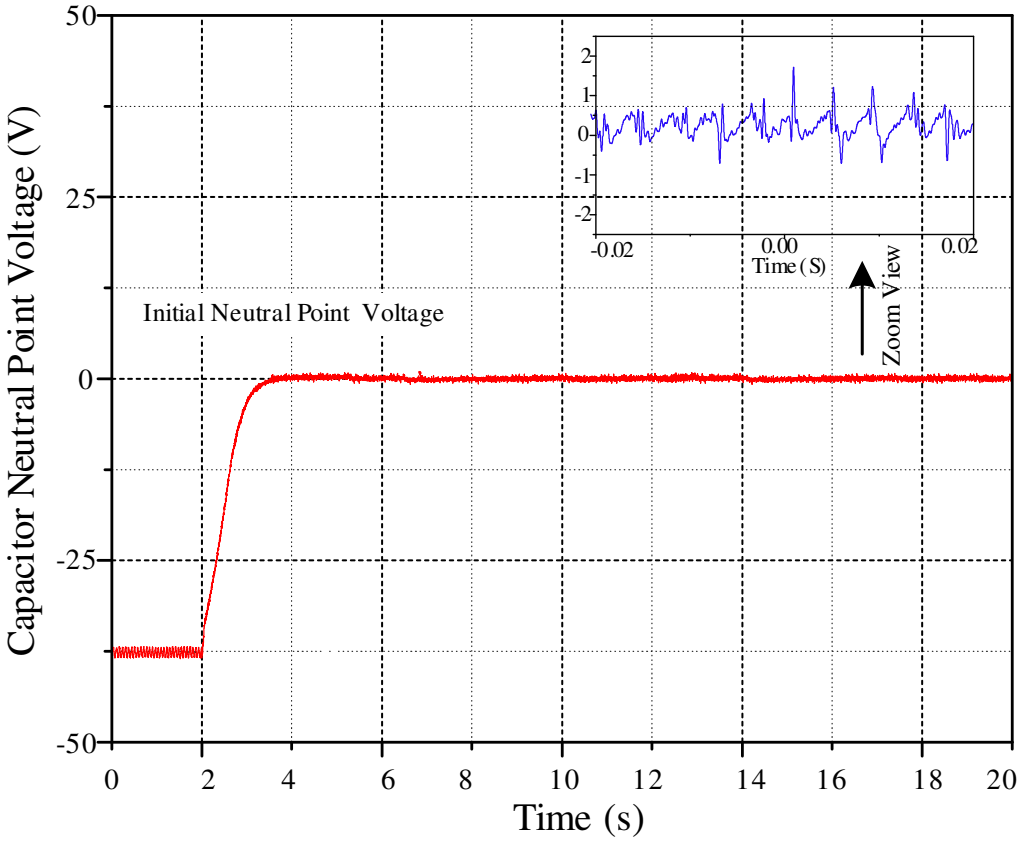


Figure 6.12: Experimental performance of the natural balancing of NPC neutral point voltage (with active balancing strategy)

6.5 Implementation of the New Three-Level Three-Phase Hysteresis Current Regulator

So far, this chapter has presented the development stages of the new three-phase hysteresis current regulator with some simulation and experimental results to confirm its performance under various operating conditions. Figure 6.13 shows the full structure of the new three-level three-phase variable band hysteresis current regulation system for both the NPC and FC inverters.

The three-phase NPC inverter employs three separate hysteresis current regulators with the additional logic circuitry per phase leg as shown in Figure 5.16. The three-phase FC inverter employs three separate hysteresis current regulators with the additional finite state machine per phase leg as shown in Figure 5.22. Section 6.6 begins by presenting consolidated results for the three-phase NPC inverter and it then presents similar results for the three-phase FC inverter.

The primary hysteresis comparison process is implemented using a conventional op-amp summing junction to calculate the current errors $e_A(t)$, $e_B(t)$ and $e_C(t)$, and to subtract the common mode interacting current $\gamma_{3L}(t)$ for the per phase current error. The resultant errors are then fed into three variable threshold analogue comparators (one comparator per phase leg) to create the three-phase hysteresis switching signals ($S_{a,tot}(t)$, $S_{b,tot}(t)$, $S_{c,tot}(t)$).

The logic decoder is implemented using the CPLD where it generates the three-phase modified hysteresis switching signals ($S_{a,OR}(t)$, $S_{b,OR}(t)$, $S_{c,OR}(t)$). The DSP uses these switching signals to extract the three-phase average inverter voltages for the calculation of three- phase variable hysteresis bands. Additionally, the DSP generates the level selection signals from the prediction of the average voltage zero-crossing.

The three-phase modified hysteresis switching signals and the level selection signals are used by the logic decoder to generate the switching signals for the modulation of the NPC inverter. The three-phase modified hysteresis switching signals and the level selection signals are fed into the CPLD to implement the FSM described in section 5.4.2 to generate the appropriate switching signals for the modulation of the FC inverters. The CPLD is also responsible to apply the dead-time to the NPC and FC switching signals.

The common mode interacting current is then calculated using an additional DAC where it outputs the three-phase level selection signals as multiplexed scaled positive and negative representations of the DC bus voltage. A multiplexer is used to generate the scaled three phase three-level switched phase voltages for the calculation of the common mode interacting current using equation (6.8).

The NP active balance strategy is implemented first by measuring the NP voltage using two additional voltage sensors to calculate the positive and the negative DC bus voltage with respect to the NP voltage. A spare DAC channel is then used to output the scaled NP voltage as the reference ZERO voltage for the generation of the three-switched phase voltages before calculation of the common mode current from equation (6.8).

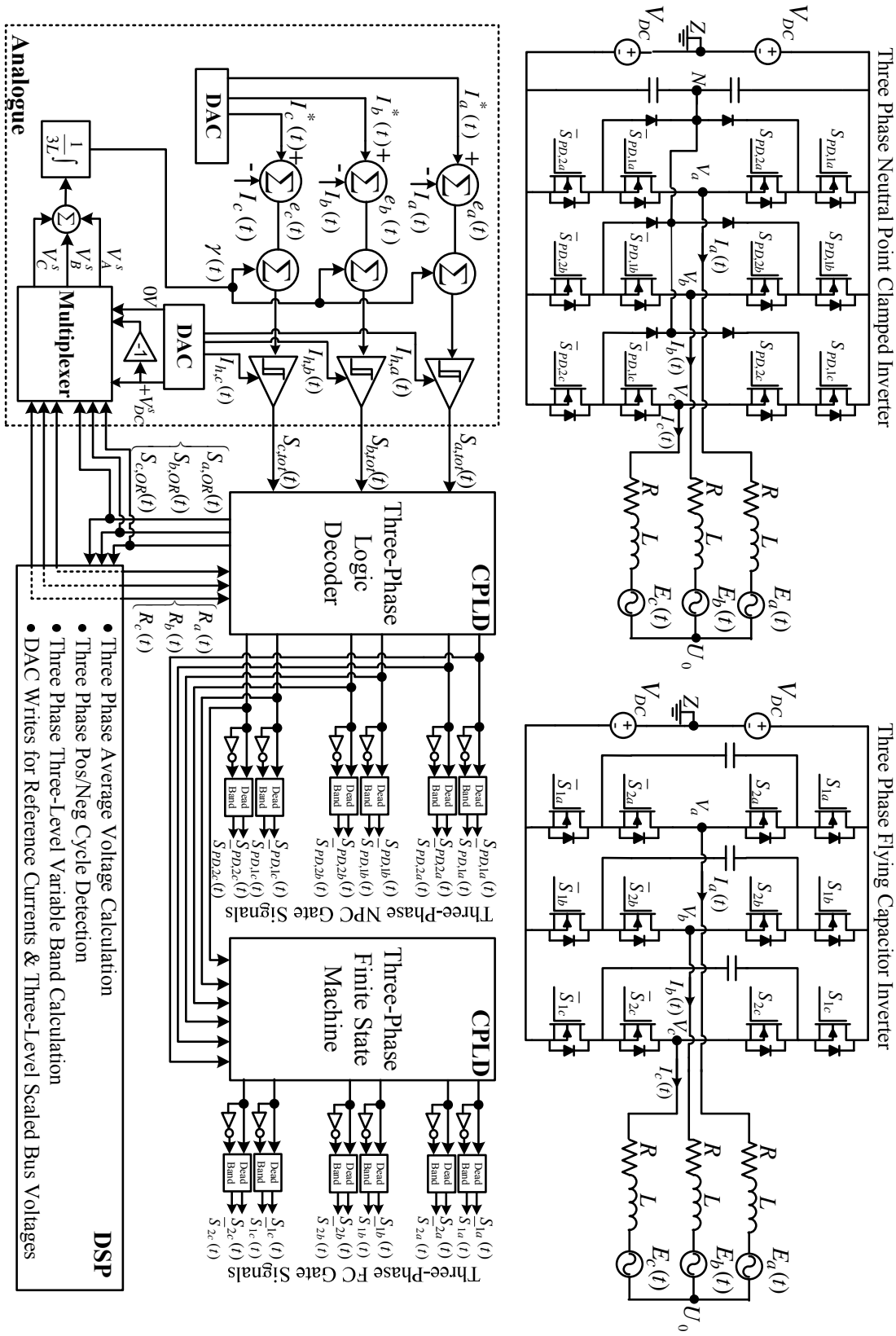


Figure 6.13: Three phase three-level NPC inverter (left) and three phase three-level FC inverter (right) feeding backemf type load using the proposed three-level three-phase variable band hysteresis controller structure

6.6 Consolidated Simulation and Experimental Results

Throughout this chapter, selected simulation and experimental results have been presented in order to support the development of the theoretical analysis. This section now provides consolidated simulation and experimental results for the operation of the proposed hysteresis current regulation approach for the three-level three-phase VSI with the circuit parameters listed in Table 6.1.

Figure 6.15 to Figure 6.26 provide results for the three-phase NPC VSI and Figure 6.27 to Figure 6.38 provide results for the three-phase FC VSI.

Figure 6.14 shows experimental results for the compensation of the common mode interacting current in a three-level three-phase system. This figure shows the three-phase scaled switched phase and the three-phase bipolar level selection signals. It also shows the measured common mode interacting current. The figure shows the calculation of the common mode current and compensation for this effect, to achieve the clean triangular phase A current error which is subtracted from each phase current error before making the hysteresis switching decisions.

Figure 6.15 (simulation) and Figure 6.16 (experiment) show the steady-state operation of the new three-level hysteresis current regulation approach applied to the three-phase neutral point clamped inverter. These figures show the three-phase output currents, which confirm the excellent current tracking capability of the proposed hysteresis regulator. It also shows the three-phase current errors, variable hysteresis bands, and phase voltages. It can be seen how the magnitude of the hysteresis variable band changes during the fundamental cycle, reaching a maximum value at the 45^{o} points on the fundamental cycle, and a minimum value at the peaks and zero crossings of the fundamental. During each zero crossing transition, the

Table 6.1: Circuit parameters for the simulation implementation of the new hysteresis current regulation approach applied to a three-level single-phase leg NPC and FC inverter

Circuit Parameter		**Value**
Resistive load	(R) (Ω)	16
Inductive load	(L) (mH)	18
Target Switching frequency	(f_{sw}) (kHz)	2.5
Total DC bus voltage	($2V_{DC}$) (V)	200
Reference current	(I_{ref}) (A)	5
Back EMF frequency	(Hz)	50

variable band is clamped to a minimum value, to avoid calculating a zero hysteresis band. Also it shows how during this transition, the current error loops back from the hysteresis boundary without actually hitting it, while the controller smoothly continues switching without difficulty after polarity change transition.

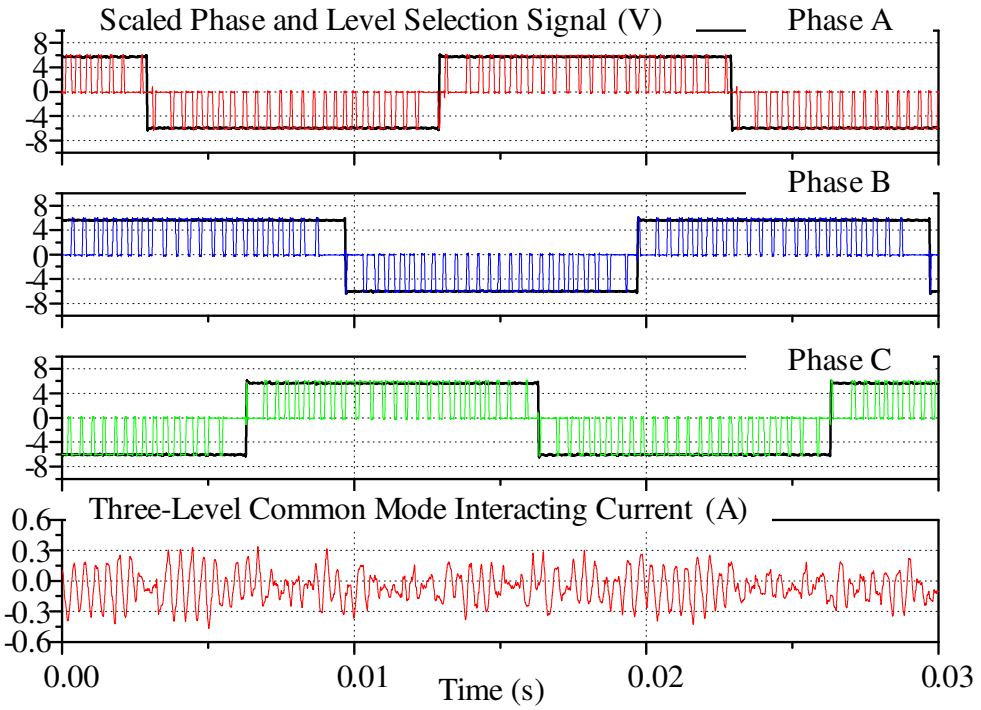


Figure 6.14: Compensation of the common mode interacting current (a) phase A scaled phase voltage and its bipolar level selection signal (b) phase B scaled phase voltage and its bipolar level selection signal (c) phase C scaled phase voltage and bipolar level selection signal (d) three-level common mode interacting current

Figure 6.17 (simulation) and Figure 6.18 (experiment) show the output phase voltage harmonic spectrum for the fixed band, variable band and variable band hysteresis controllers with synchronized current errors for the three-phase three-level NPC inverter. The figures show how the use of new variable hysteresis band strategy significantly improves the output voltage harmonic performance compared to a three-level fixed-band HCC. In simulation, the WTHD is reduced from 1.89% for a fixed-band HCC to 1.30% for the variable band HCC, and is further reduced to 1.19% by synchronizing the current error zero crossing to a fixed reference clock with a defined carrier and side band harmonics around the target switching frequency of 2500Hz. The same results are confirmed in experiment where the WTHD is reduced from 1.64% for a fixed-band HCC to 1.45% for the variable band HCC and to 1.32% with synchronization to fixed clock. The harmonic performance of the new three-level variable hysteresis band is essentially equivalent to that of the harmonically superior open-loop phase disposition (PD) PWM.

Three-Level Three-Phase Neutral Point Clamped Inverter

Simulation - Steady-State Operation

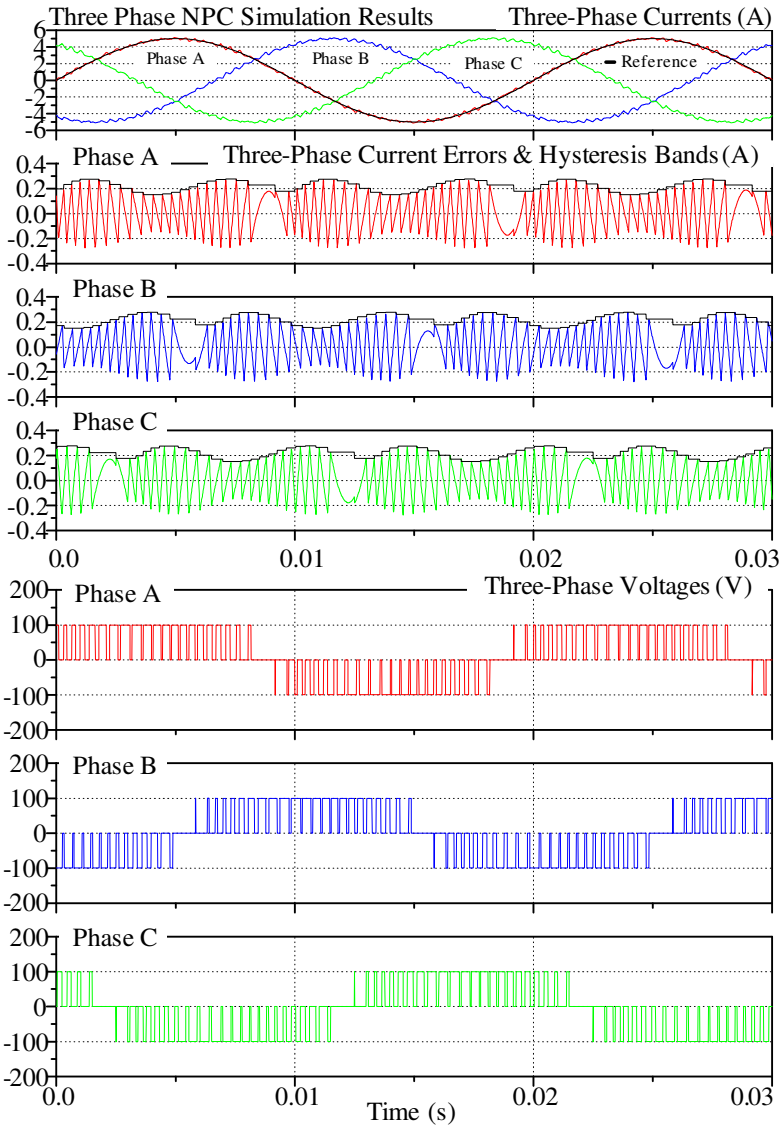


Figure 6.15: Steady-state operation of the proposed three-level hysteresis current regulation approach applied to three-phase NPC inverter showing three-phase output currents, current errors and variable hysteresis bands, phase voltages, Modulation depth=0.9

Three-Level Three-Phase Neutral Point Clamped Inverter

Experiment - Steady-State Operation

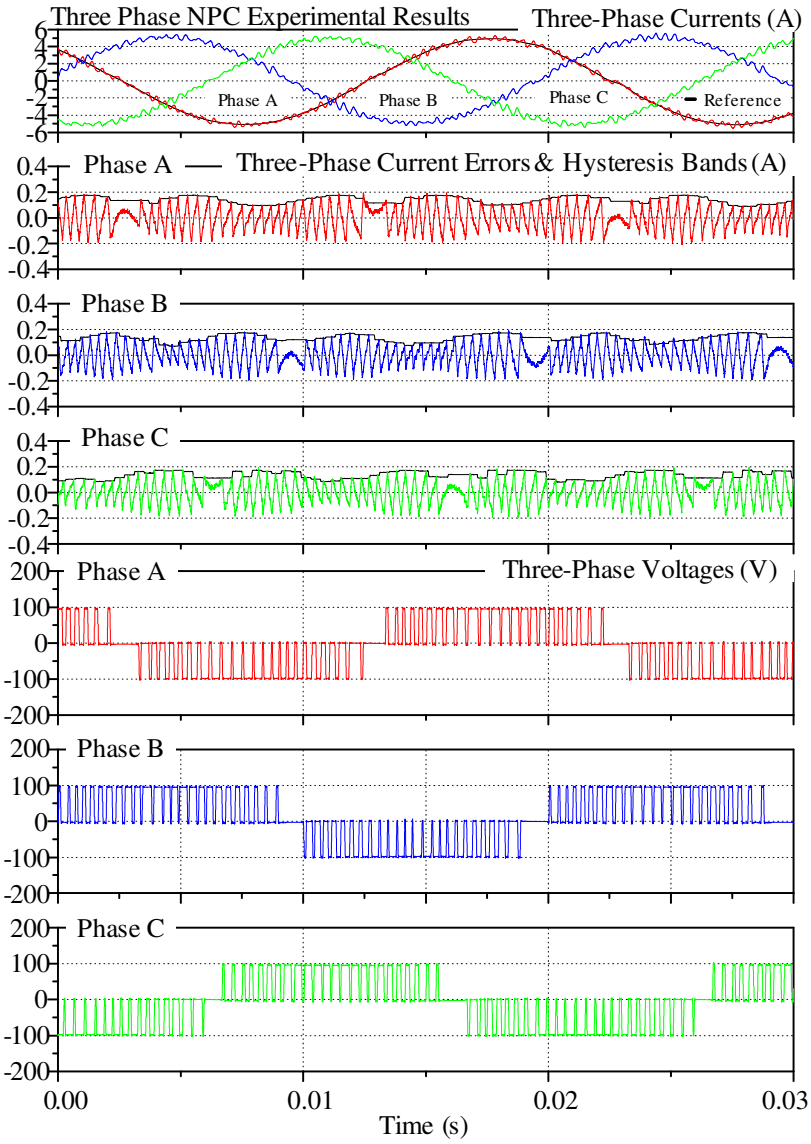


Figure 6.16: Steady-state operation of the proposed three-level hysteresis current regulation approach applied to three-phase NPC inverter showing three-phase output currents, current errors and variable hysteresis bands and phase voltages, Modulation depth=0.9

Simulation - Phase Voltage Harmonic Performance

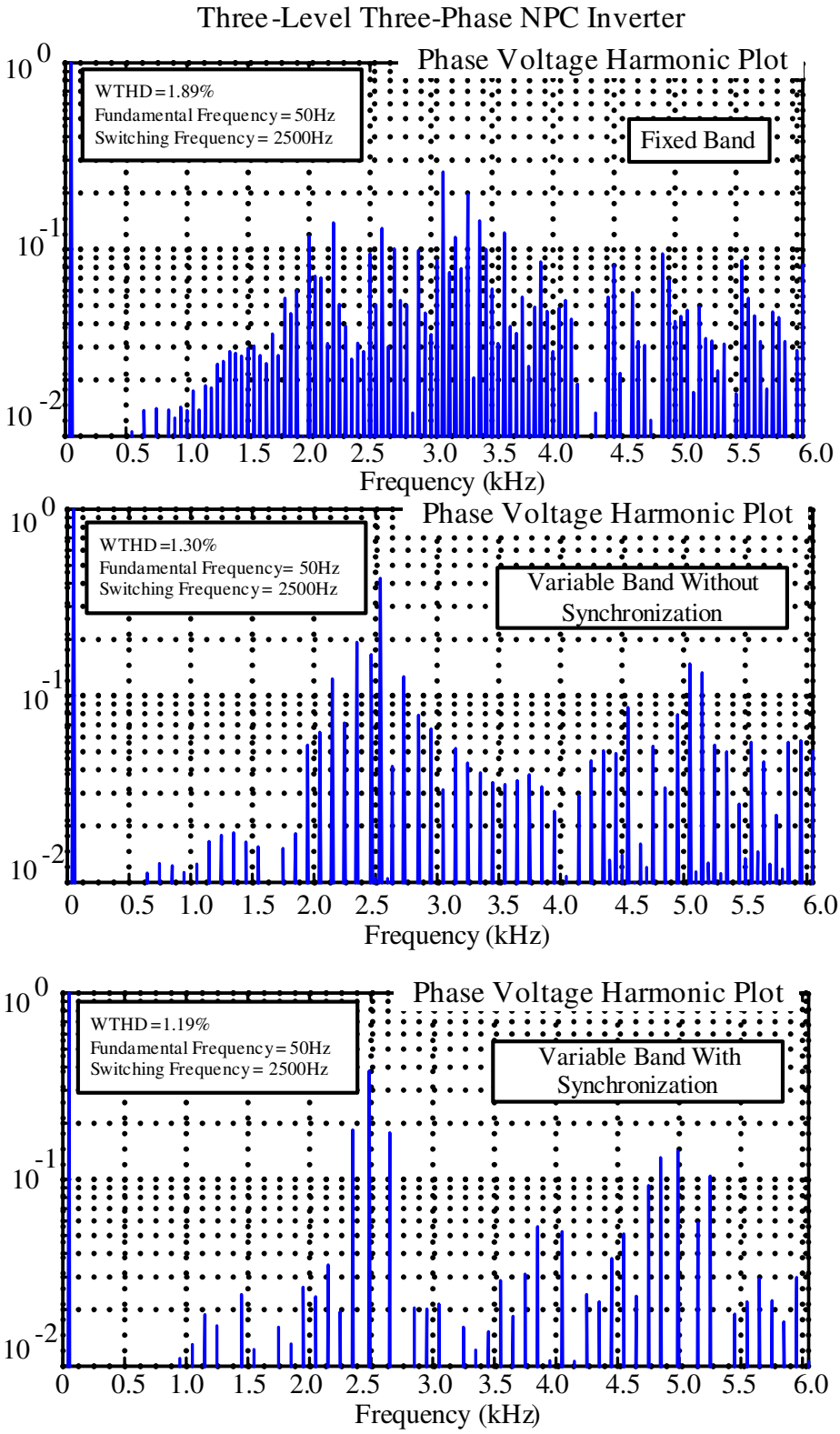


Figure 6.17: Phase leg voltage harmonics of the NPC inverter under hysteresis current regulation: (a) Fixed band (b) Variable band, (c) Fine-tuned variable band, (modulation depth = 0.9)

Experiment - Phase Voltage Harmonic Performance

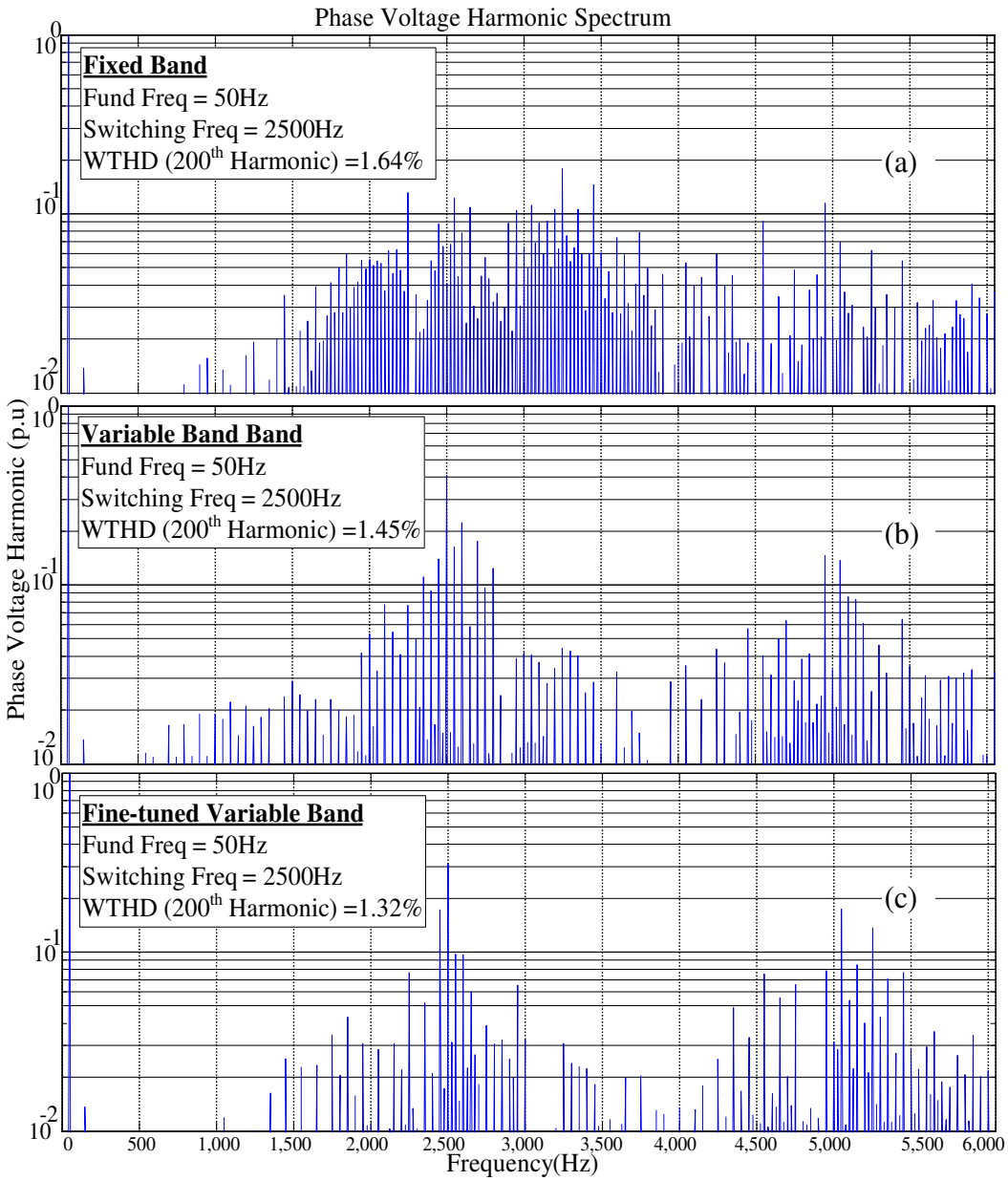


Figure 6.18: Phase voltage harmonics of the NPC inverter under hysteresis current regulation: (a) Fixed band (b) Variable band, (c) Fine-tuned variable band, Modulation depth = 0.9

Figure 6.19 (simulation) and Figure 6.20 (experiment) show the three-phase line-to-line voltages of the NPC inverter. These figures confirm the characteristic five-level line-to-line switching of PD modulation with a minor leakage between the levels. The inaccuracy occurs during the adjacent voltage level transitions, where the controller has to clamp the variable hysteresis above zero discussed in section 5.5 to maintain switching control during the transition.

Figure 6.21 (simulation) and Figure 6.22 (experiment) show the equivalent open-loop PD PWM line to line voltage harmonic plot for the NPC inverter. From this figure, it can be clearly seen that the carrier harmonic is cancelled in the line to line voltage with a significant WTHD reduction for a fine-tuned variable hysteresis regulator in comparison to a fixed-band hysteresis regulator.

Figure 6.23 (simulation) and Figure 6.24 (experiment) show the transient response achieved for a 100% step-up change in reference current for the NPC inverter, which confirms that the variable hysteresis band addition has not compromised the excellent dynamic response of the conventional hysteresis control [51].

Figure 6.25 (simulation) and Figure 6.26 (experiment) show further studies carried out in simulation for the modulation depth of 110% to investigate the operation of the proposed hysteresis current regulation approach in overmodulation using the band clamp strategy. It can be seen that the controller maintains excellent current control capability during overmodulation. More importantly, the controller still maintains an excellent balanced neutral point voltage for the NPC inverter. Also note in Figure 6.26 (experiment), the effect of switching ripple in the per phase current error freewheeling in the overmodulation region. This ripple is caused by both the common mode interacting current since the other two phase legs are still switching, and also the sampling effect on the phase output current reference.

For the FC inverter, Figure 6.27 to Figure 6.38 extend the simulation and experimental results to verify the excellent performance of the new three-phase three-level hysteresis current regulator when applied to the FC VSI.

Also note in Figure 6.28 and Figure 6.38 how the controller maintains an excellent balanced flying capacitor voltage both at the modulation depths of 0.9 and 1.1 respectively.

Simulation - Three-Phase Line-to-Line Voltages

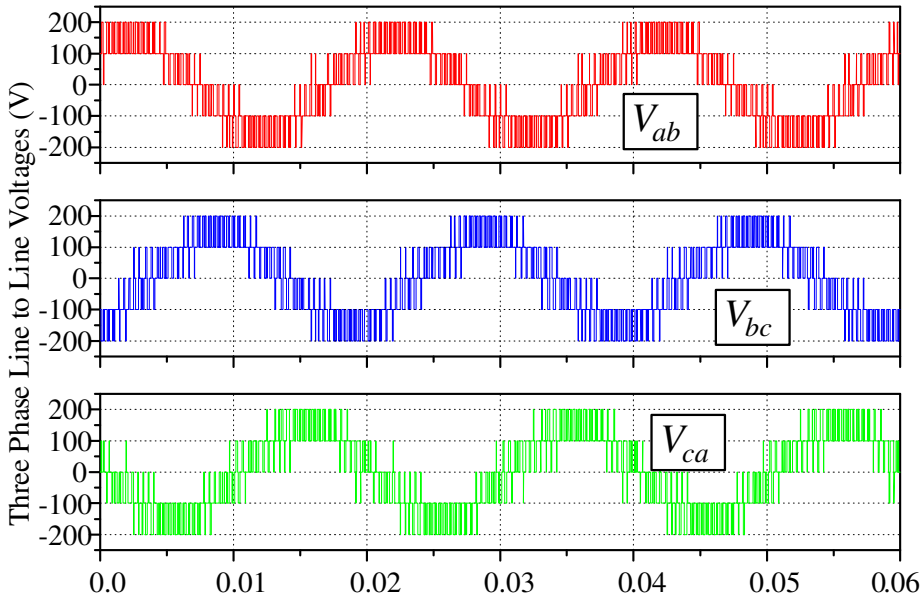


Figure 6.19: PD equivalent three-phase line-to-line voltages of NPC inverter

Experiment - Three-Phase Line-to-Line Voltages

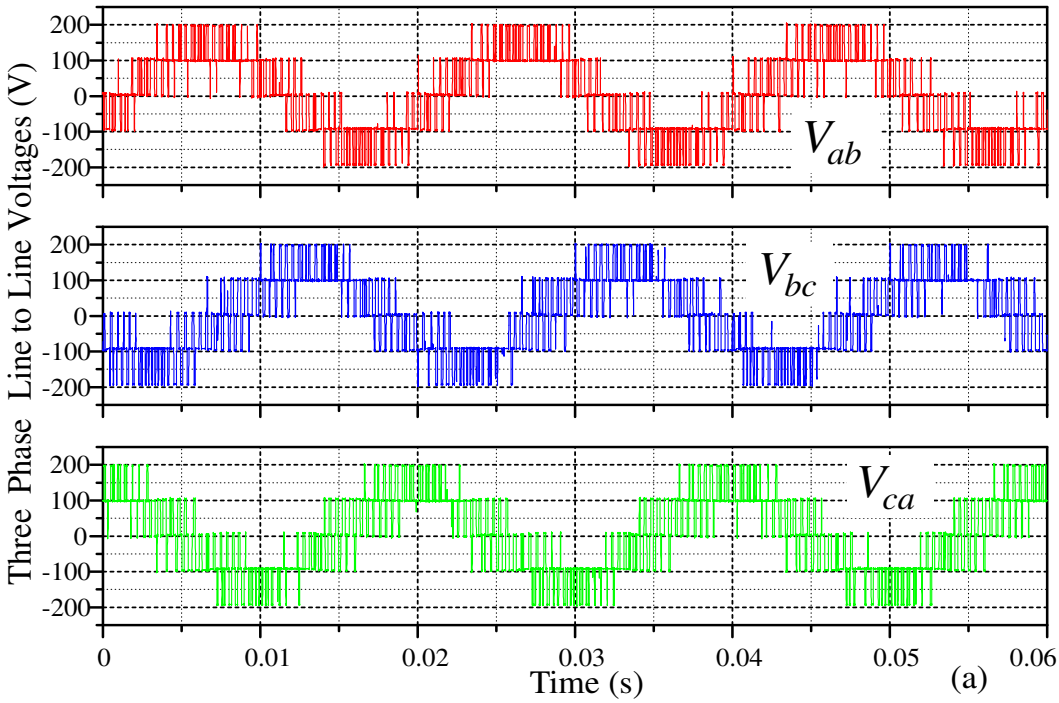


Figure 6.20: PD equivalent three-phase line-to-line voltages of NPC inverter

Simulation - Three-Phase Line-to-Line Harmonics

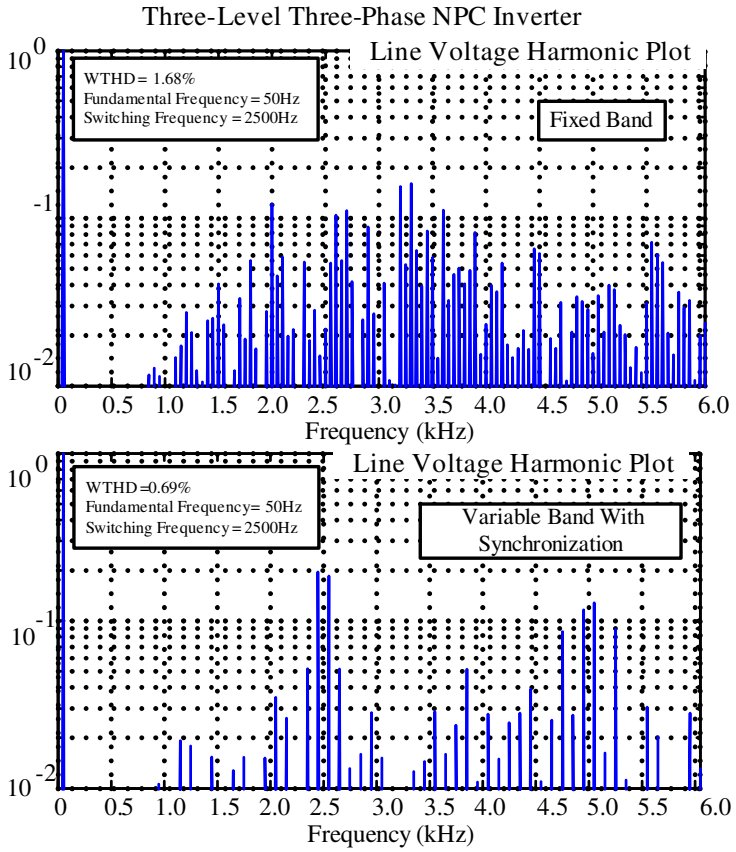


Figure 6.21: Line voltage harmonics of the NPC inverter under hysteresis current regulation: (a) Fixed band (b) Fine-tuned variable band, Modulation depth = 0.9

Experiment - Three-Phase Line-to-Line Harmonics

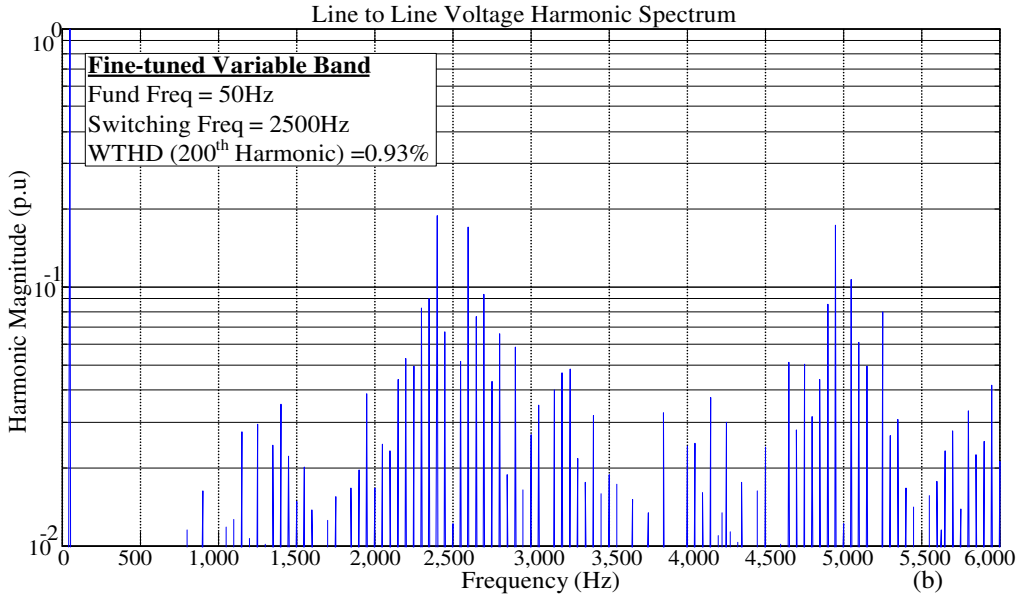


Figure 6.22: Line voltage harmonics of the NPC inverter under fine-tuned variable band, Modulation depth = 0.9

Simulation - Transient Operation

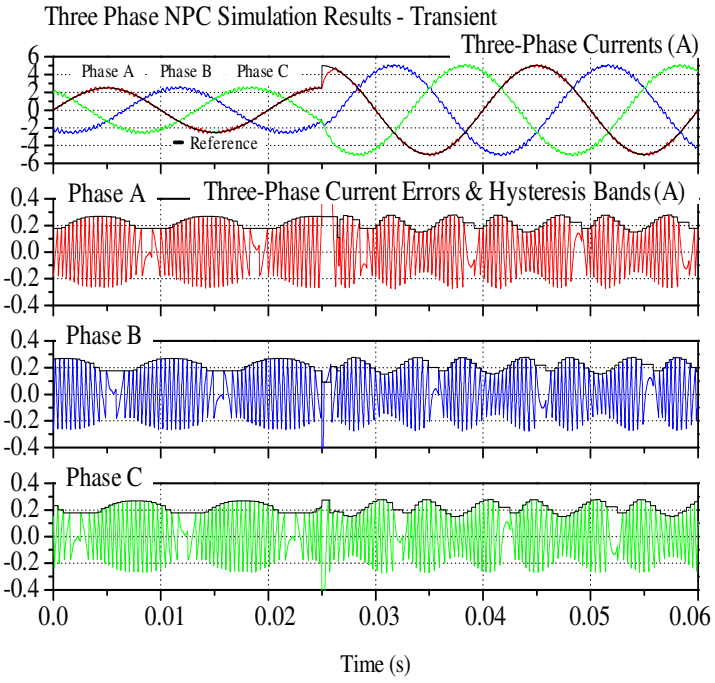


Figure 6.23: Transient operation of the proposed three-level hysteresis current regulation approach applied to three-phase NPC inverter showing three-phase output currents, current errors and variable hysteresis bands

Experiment - Transient Operation

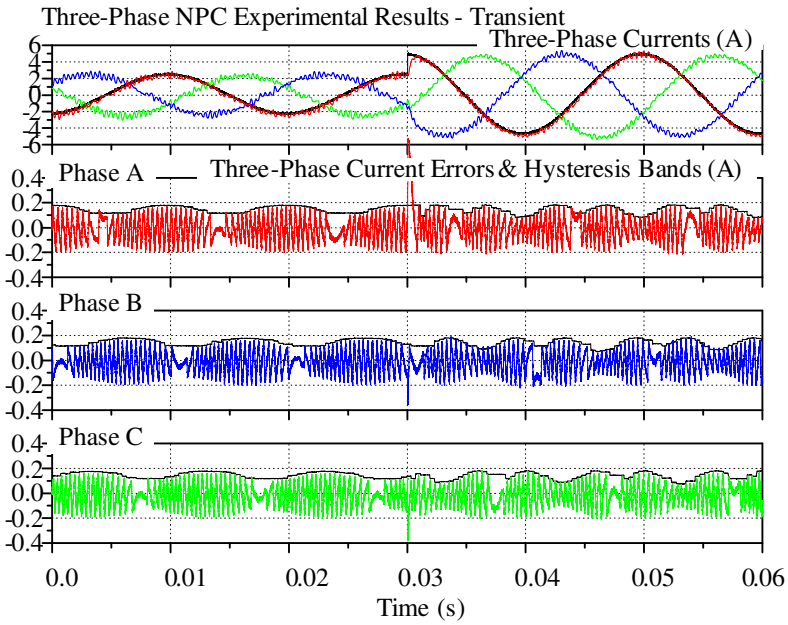


Figure 6.24: Transient operation of the proposed three-level hysteresis current regulation approach applied to three-phase NPC inverter showing three-phase output currents, current errors and variable hysteresis bands

Simulation - Overmodulation Operation

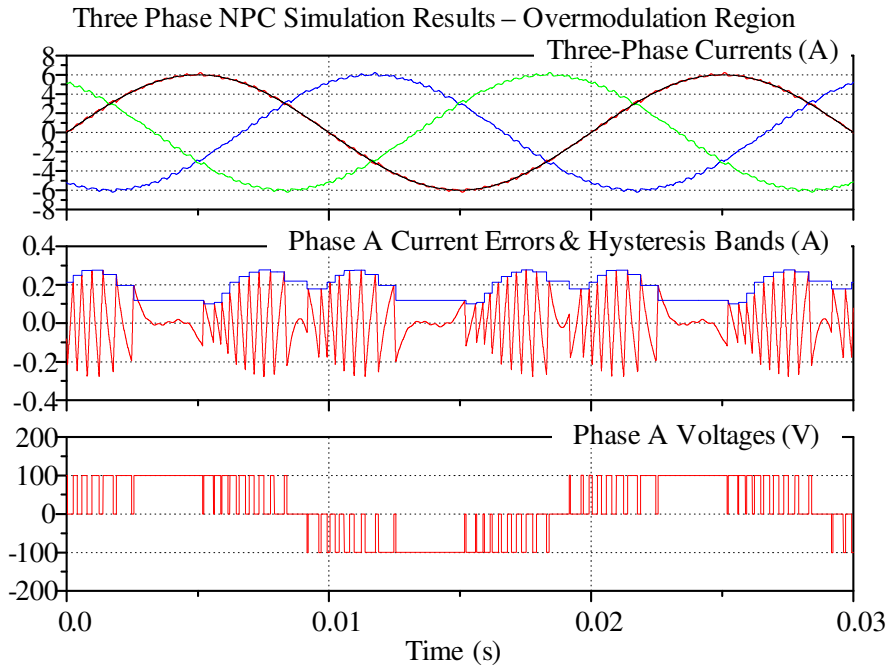


Figure 6.25: Overmodulation operation of the proposed three-level hysteresis current regulation approach applied to three-phase NPC inverter showing three-phase output currents, phase A current error, variable hysteresis band and phase voltage, Modulation depth=1.2

Experiment - Overmodulation Operation

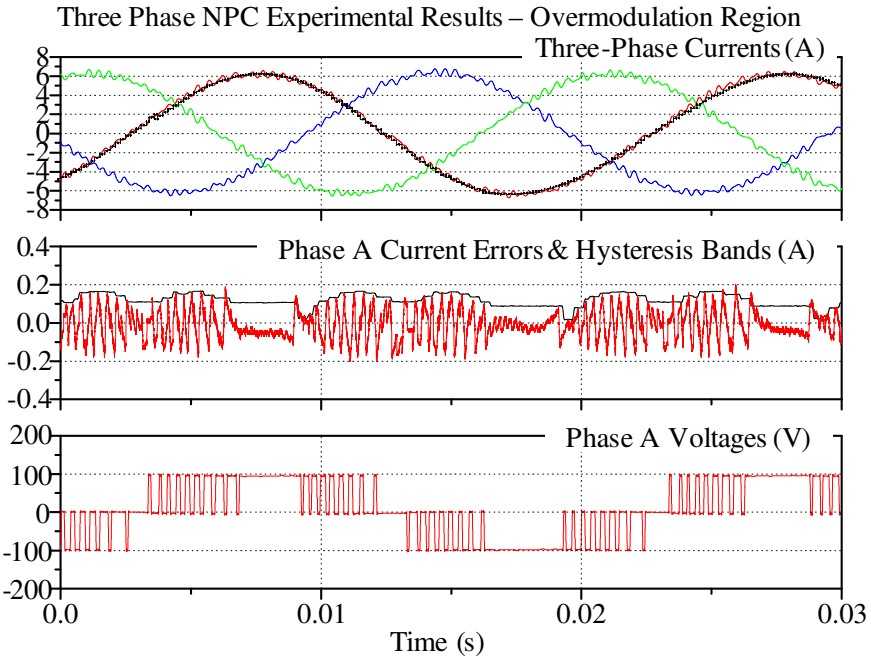


Figure 6.26: Overmodulation operation of the proposed three-level hysteresis current regulation approach applied to three-phase NPC inverter showing three-phase output currents, phase A current error, variable hysteresis band and phase voltage, Modulation depth=1.2

Three-Level Three-Phase Flying Capacitor Inverter

Simulation - Steady-State Operation

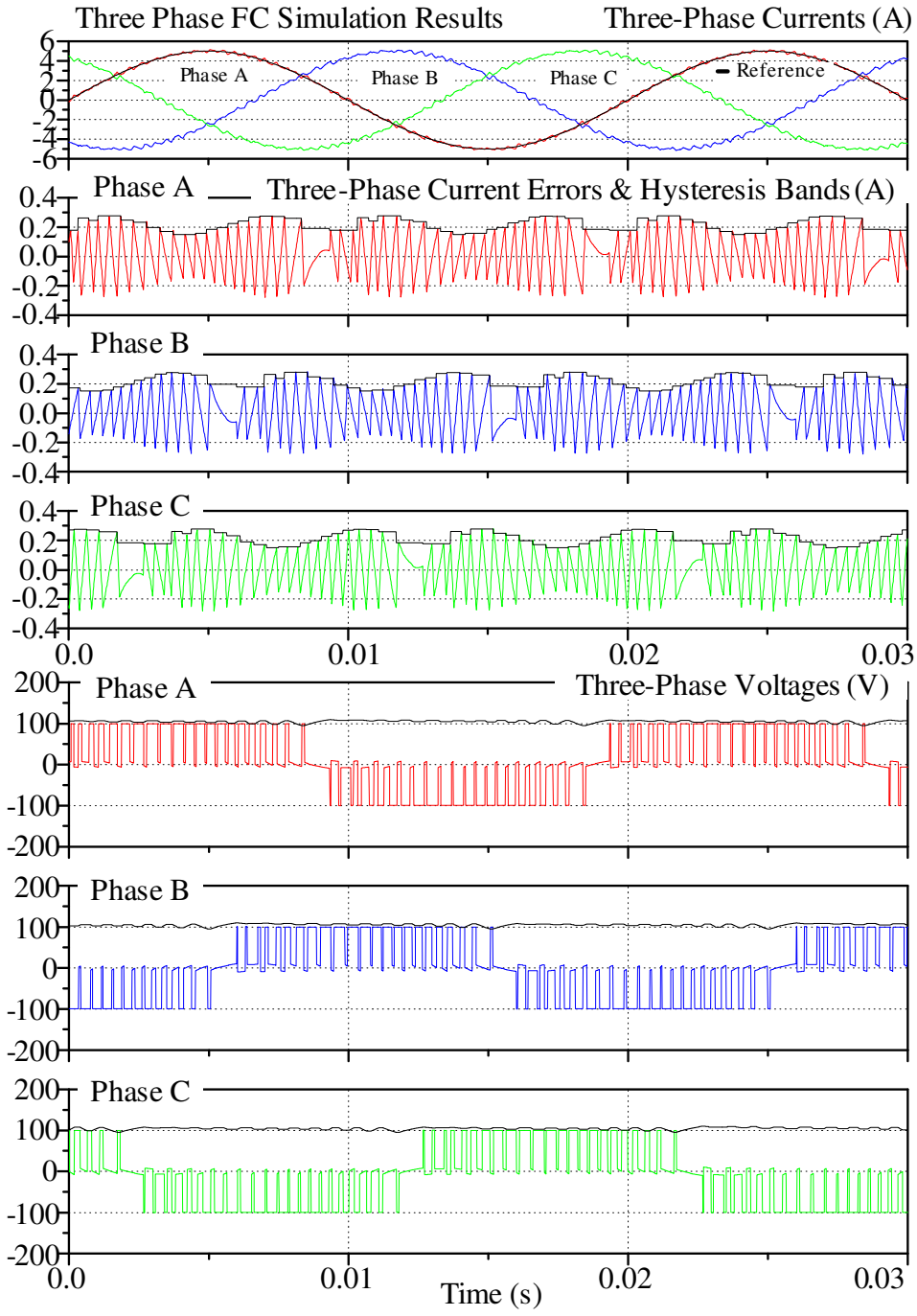


Figure 6.27: Steady-state operation of the proposed three-level hysteresis current regulation approach applied to three-phase FC inverter showing three-phase output currents, current errors and variable hysteresis bands, phase & FC voltages, Modulation depth=0.9

Three-Level Three-Phase Flying Capacitor Inverter

Experiment - Steady-State Operation

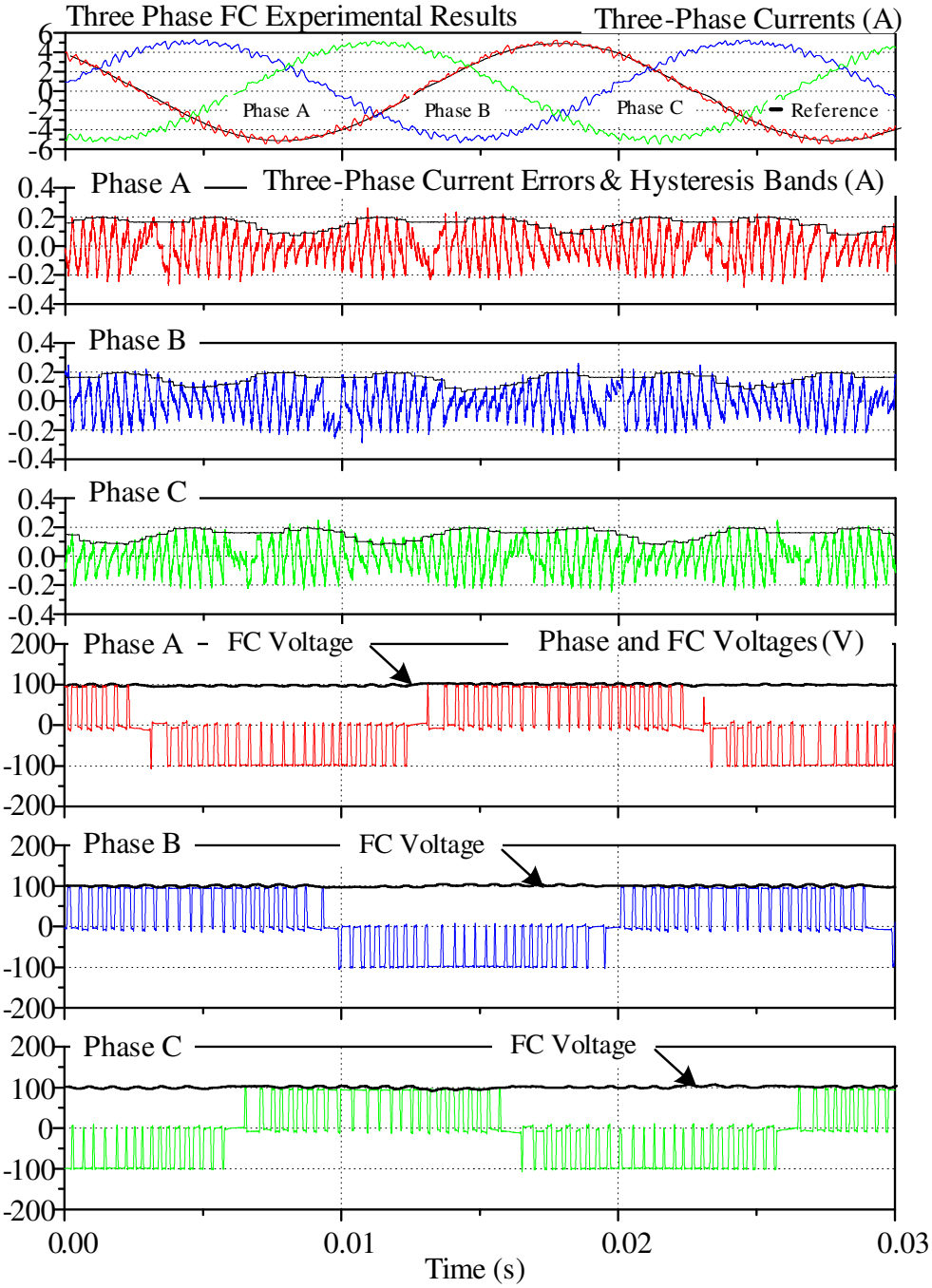


Figure 6.28: Steady-state operation of the proposed three-level hysteresis current regulation approach applied to three-phase FC inverter showing three-phase output currents, current errors and variable hysteresis bands, phase & FC voltages, Modulation depth=0.9

Simulation - Phase Voltage Harmonic Performance

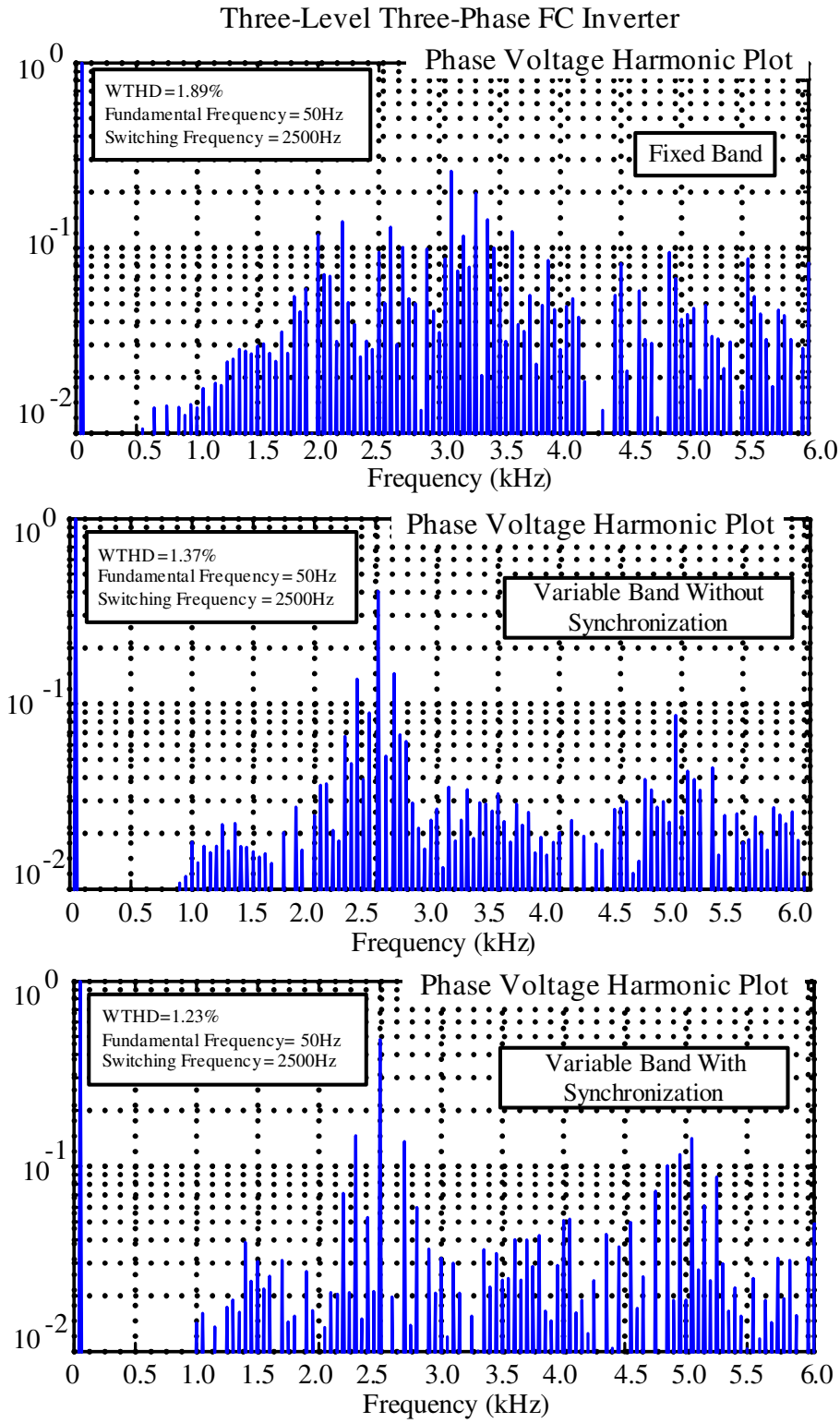


Figure 6.29: Phase voltage harmonics of the FC inverter under hysteresis current regulation: (a) Fixed band (b) Variable band, (c) Fine-tuned variable band, Modulation depth = 0.9

Experiment - Phase Voltage Harmonic Performance

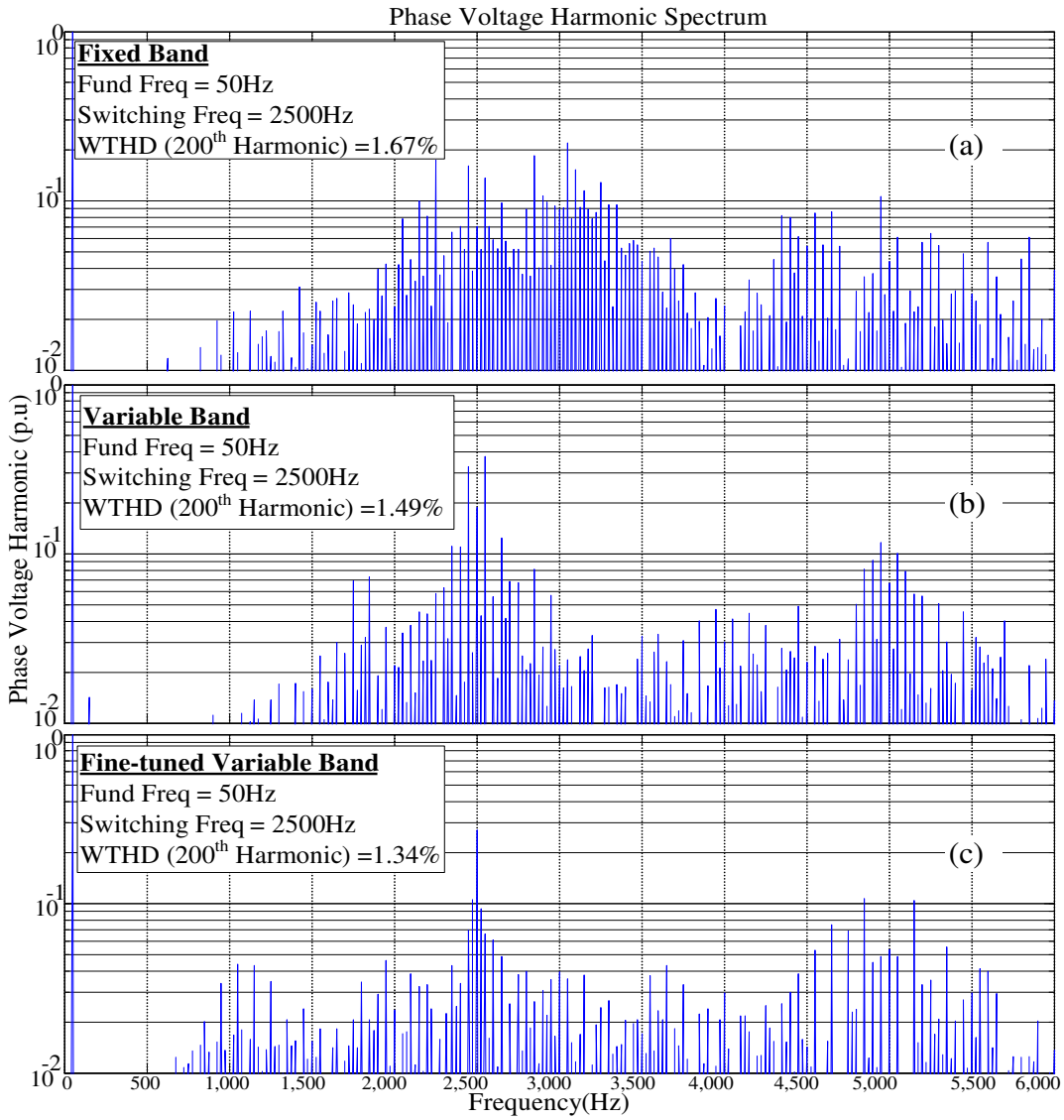


Figure 6.30: Phase voltage harmonics of the FC inverter under hysteresis current regulation: (a) Fixed band (b) Variable band, (c) Fine-tuned variable band, Modulation depth = 0.9

Simulation - Three-Phase Line-to-Line Voltages

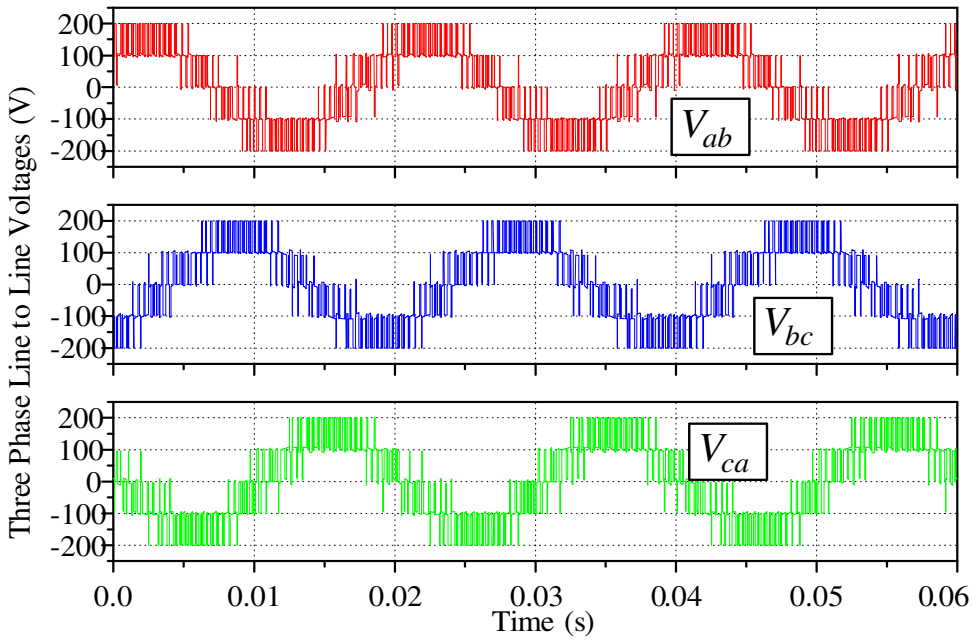


Figure 6.31: PD equivalent three-phase line-to-line voltages of FC inverter

Experiment - Three-Phase Line-to-Line Voltages

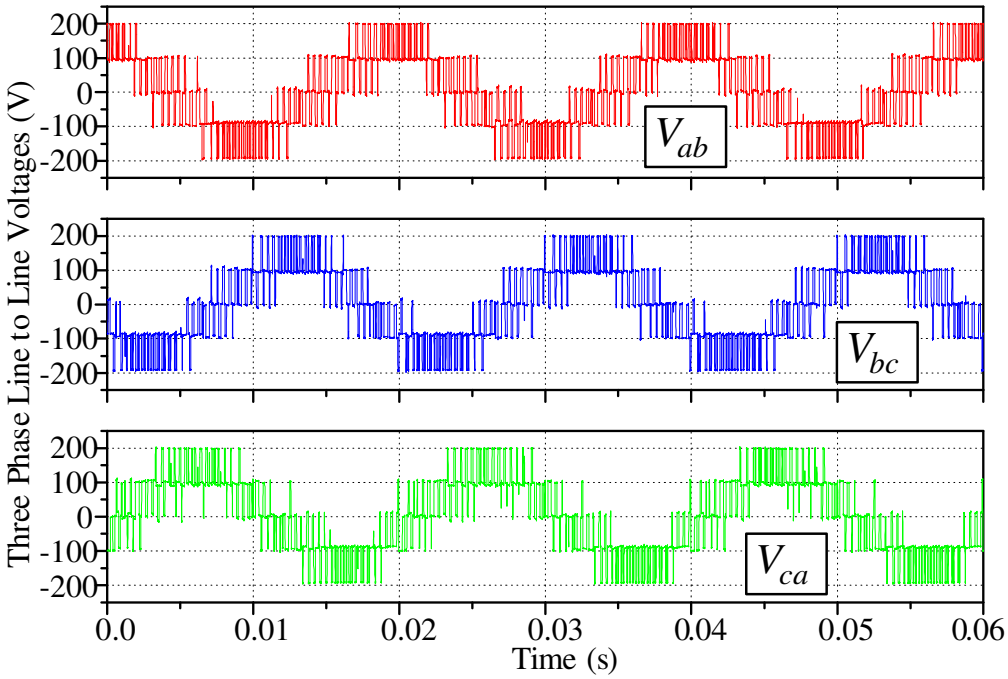


Figure 6.32: PD equivalent three-phase line-to-line voltages of FC inverter

Simulation - Three-Phase Line-to-Line Harmonics

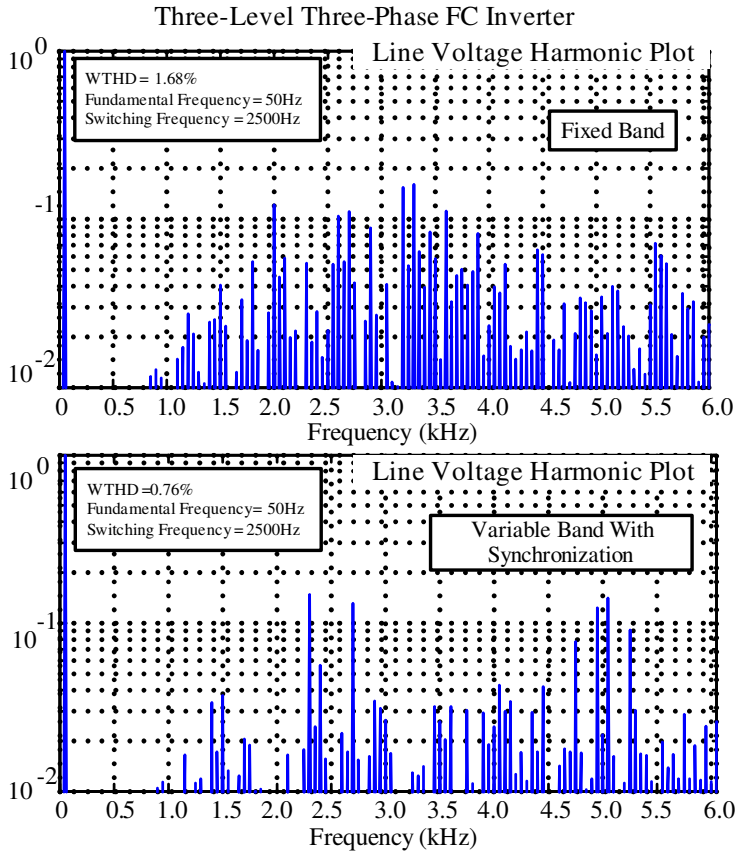


Figure 6.33: Line voltage harmonics of the FC inverter under hysteresis current regulation: (a) Fixed band (b) Fine-tuned variable band, Modulation depth = 0.9

Experiment - Three-Phase Line-to-Line Harmonics

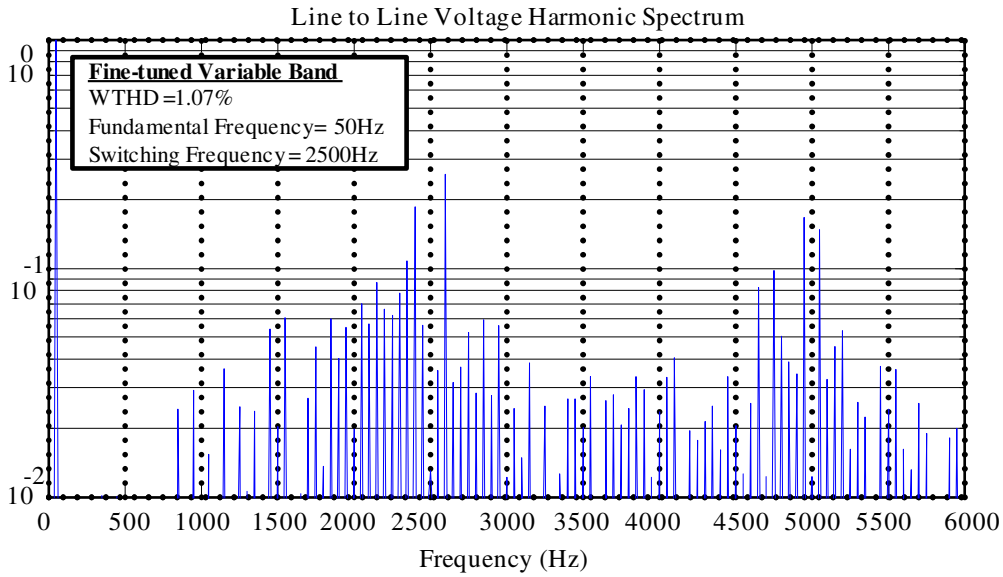


Figure 6.34: Line voltage harmonics of the FC inverter under hysteresis current regulation: (a) Fixed band (b) Fine-tuned variable band, Modulation depth = 0.9

Simulation - Transient Operation

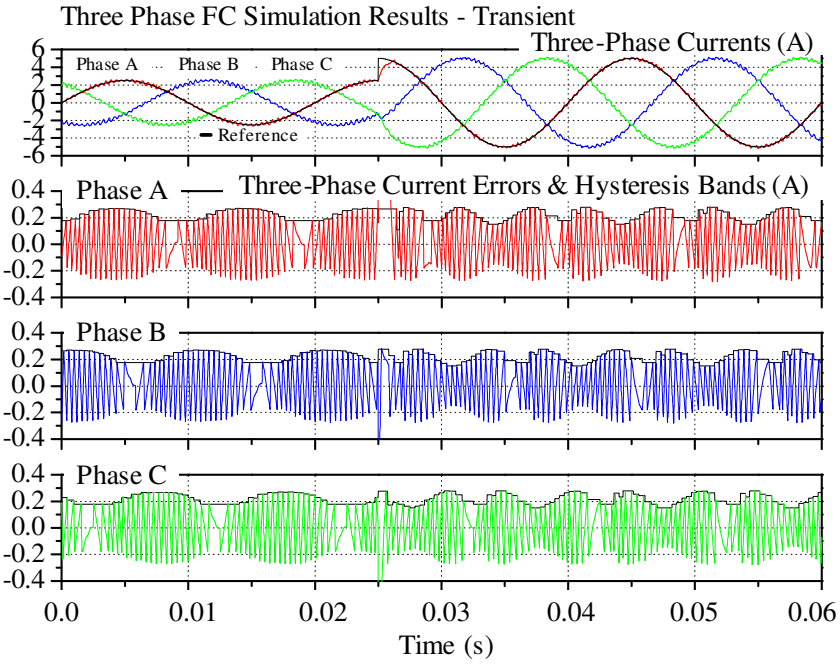


Figure 6.35: Transient operation of the proposed three-level hysteresis current regulation approach applied to three-phase FC inverter showing three-phase output currents, current errors and variable hysteresis bands

Experiment - Transient Operation

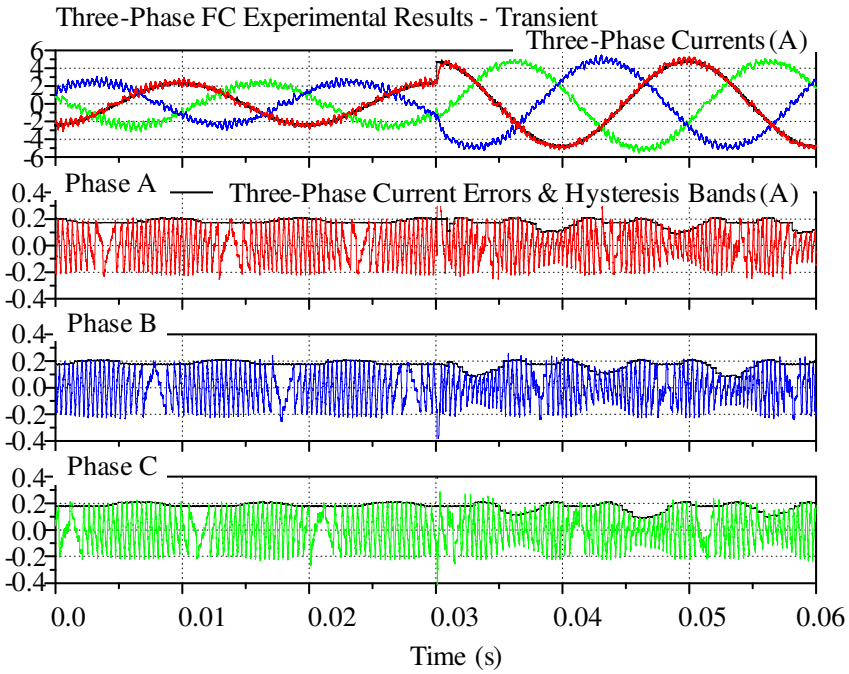


Figure 6.36: Transient operation of the proposed three-level hysteresis current regulation approach applied to three-phase FC inverter showing three-phase output currents, current errors and variable hysteresis bands

Simulation - Overmodulation Operation

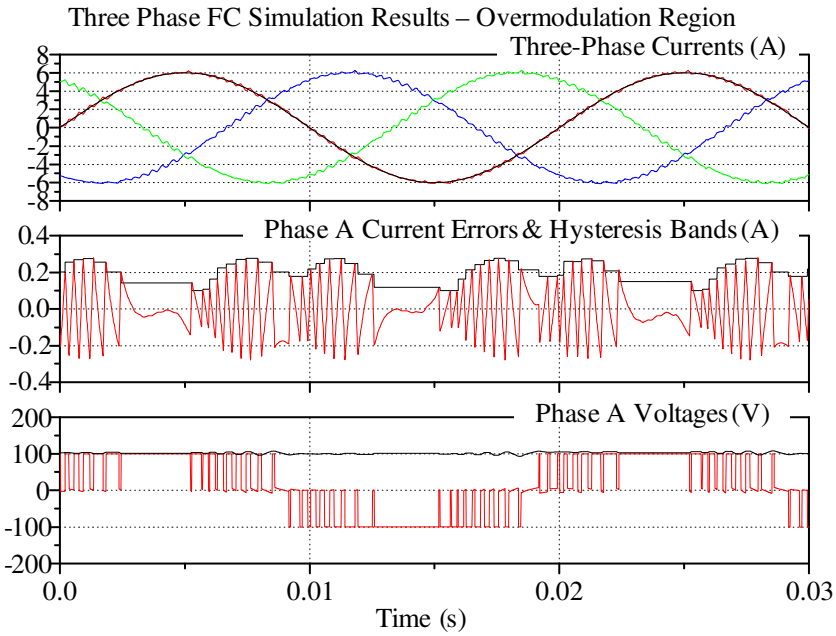


Figure 6.37: Overmodulation operation of the proposed three-level hysteresis current regulation approach applied to three-phase FC inverter showing three-phase output currents, phase A current error, variable hysteresis band and phase voltage, Modulation depth=1.2

Experiment - Overmodulation Operation

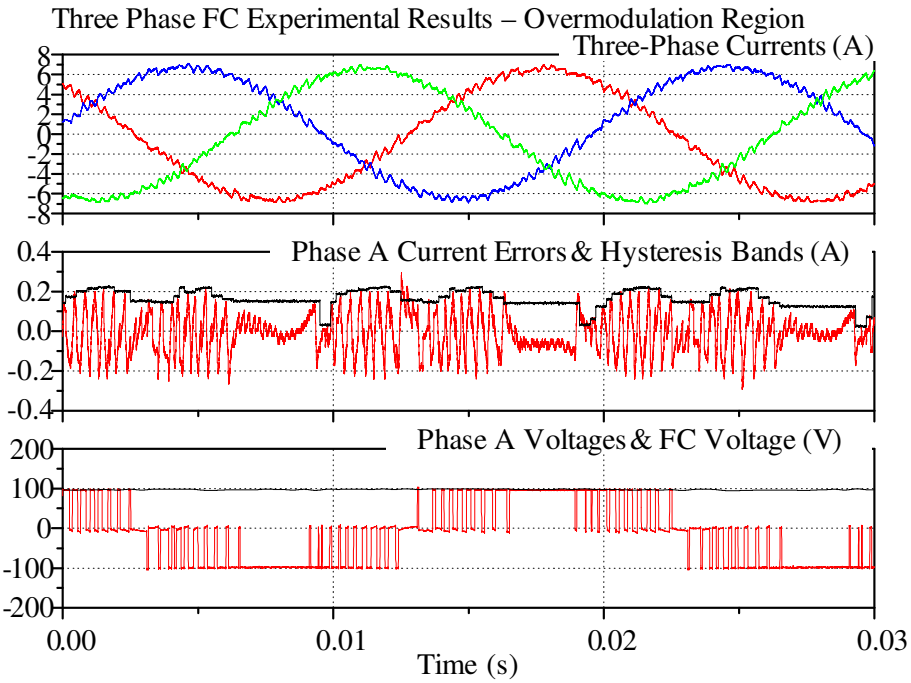


Figure 6.38: Overmodulation operation of the proposed three-level hysteresis current regulation approach applied to three-phase FC inverter showing three-phase output currents, phase A current error, variable hysteresis band and phase voltage, Modulation depth=1.2

Operation under Low Pulse Ratio

Further investigation has been carried out experimentally to verify the operation of the three-level hysteresis current regulator under a low pulse ratio of 16. Figure 6.39 and Figure 6.40 show the steady-state operation of three-level hysteresis current controller applied to NPC and FC inverters respectively. The circuit parameters used to conduct these experiments remain the same as those listed in Table 6.1 with an operating switching frequency of approximately 800 Hz. These figures confirm an excellent operating performance showing the three-phase output currents, phase A current error and variable hysteresis band, phase voltage and the PD equivalent line to line voltage. Figure 6.41 shows the harmonic performance of phase and line voltages that confirm the constant operating frequency achieved by the proposed hysteresis regulator.

Experiment - Three-Level Three-Phase NPC Inverter

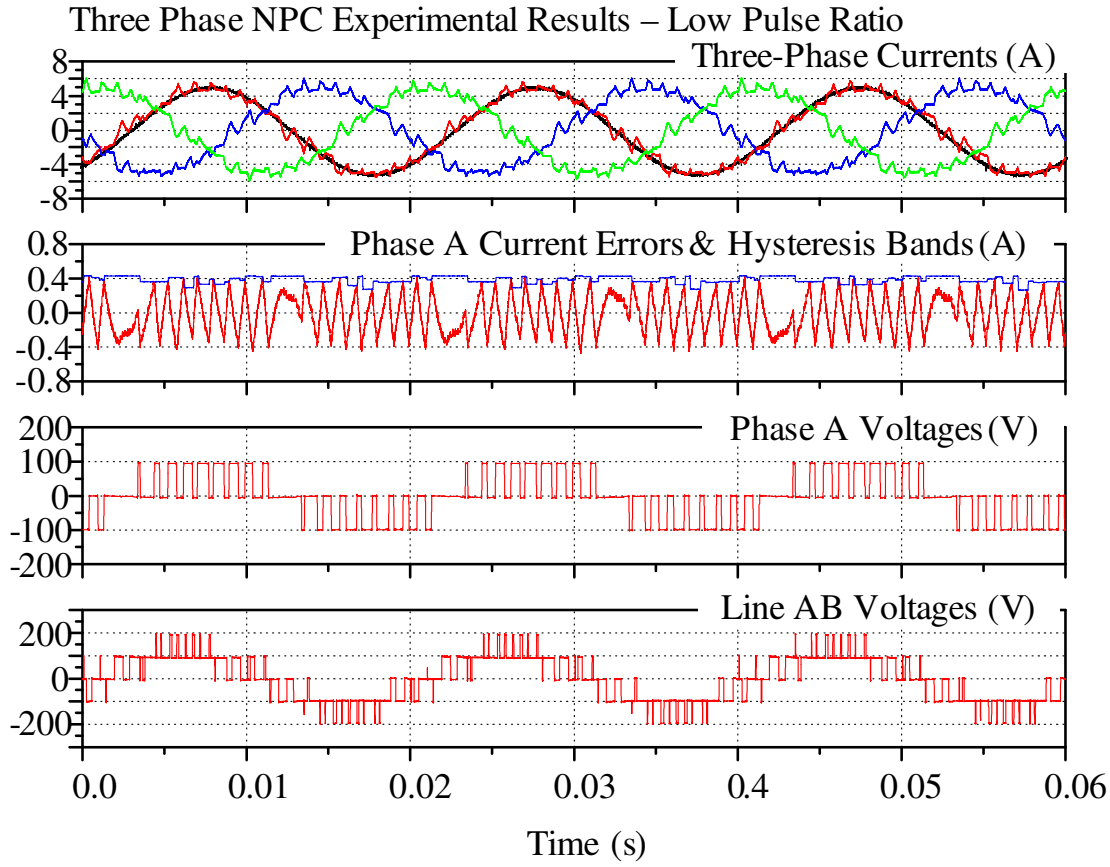


Figure 6.39: Steady-state operation of the proposed three-level hysteresis current regulation approach applied to three-phase NPC inverter showing three-phase output currents, phase A current error and variable hysteresis band, phase & line voltages, Pulse ratio = 16 , Modulation depth=0.9

Operation under Low Pulse Ratio (PR=16)

Experiment - Three-Level Three-Phase Flying Capacitor Inverter

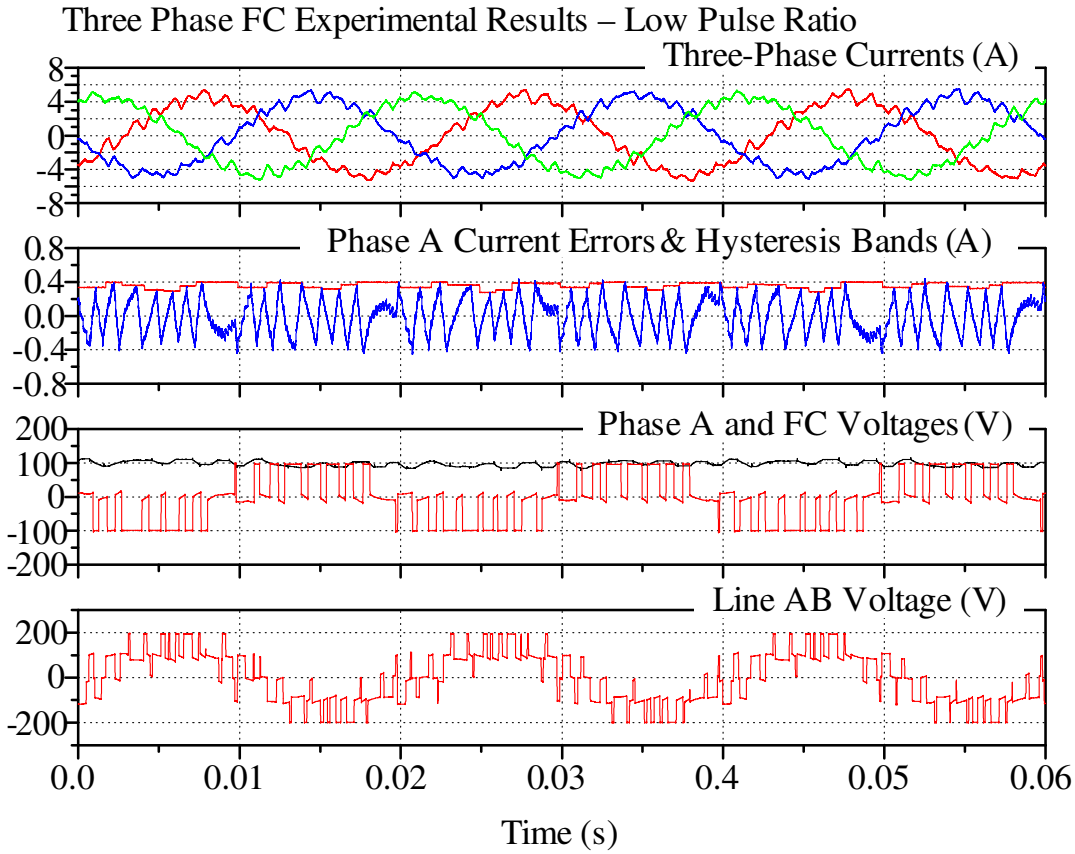


Figure 6.40: Steady-state operation of the proposed three-level hysteresis current regulation approach applied to three-phase FC inverter showing three-phase output currents, phase A current error and variable hysteresis band, phase, FC& line voltages, Pulse ratio = 16 , Modulation depth=0.9

Operation under Low Pulse Ratio

Experiment - Harmonic Performance

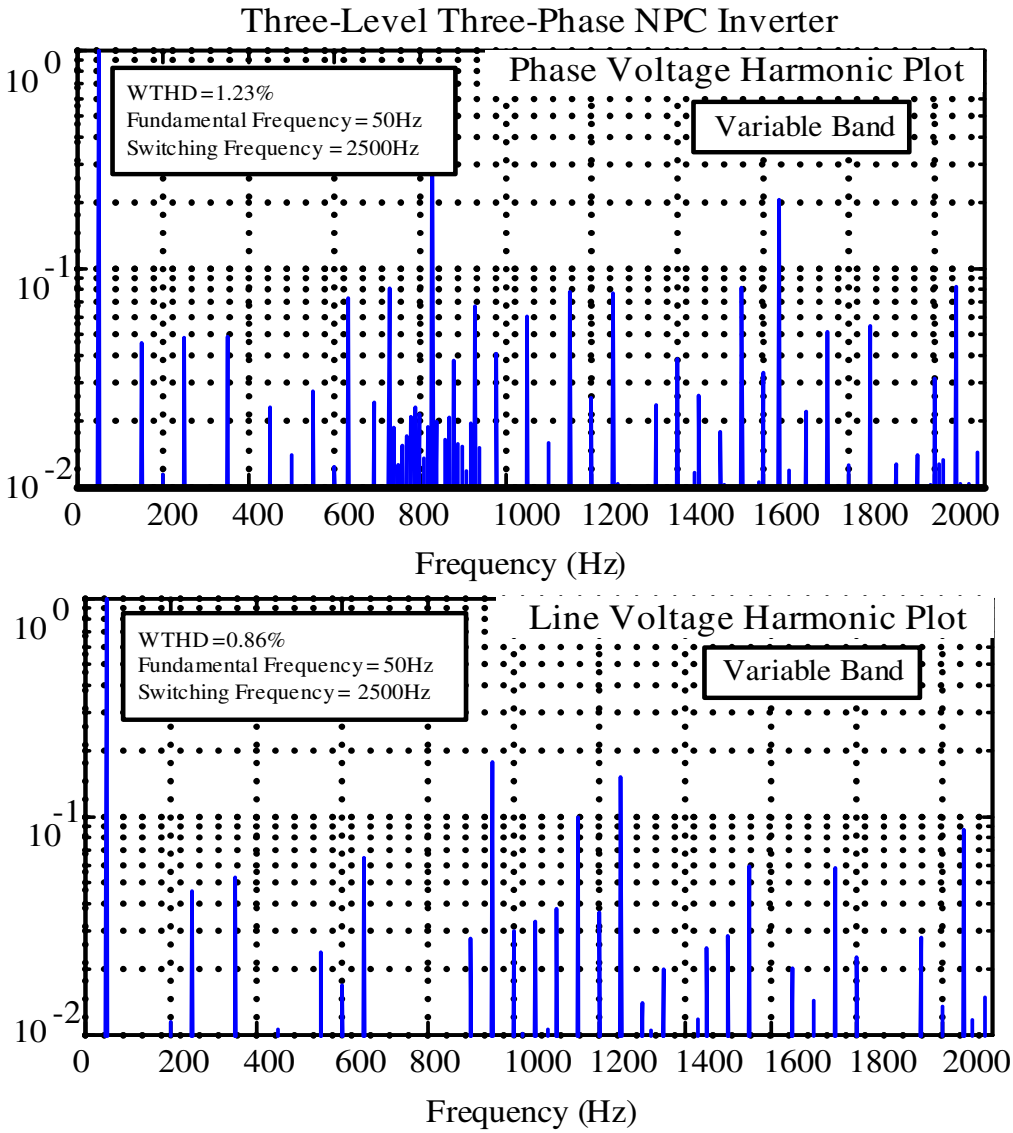


Figure 6.41: Harmonic performance of (a) phase voltage (b) line voltage
Pulse ratio = 16 , Modulation depth=0.9

6.7 Summary

This chapter has presented a new three-phase three-level hysteresis current regulator for a three-level three-phase multilevel inverters. The new constant frequency hysteresis current regulation approach has been extended to three-level three-phase NPC and FC inverters by compensating for the common mode interacting current using the switching information of the three-phase hysteresis switching signals and level selection signals, in a similar way as described in chapter 4.

The three-phase current error zero-crossings were then synchronized to a fixed reference clock to ensure that the VSI selects the three nearest space vectors and thus achieves a harmonic performance that is very close to the harmonically superior open-loop PD PWM.

Next, the operation of the controller was extended into the overmodulation region by clamping the three-level variable hysteresis band to a minimum value in a similar way as described in chapter 5. This ensures the controller switching stability and a smooth transition between the linear and non-linear regions.

It has also been shown that constant switching frequency operation will achieve natural balancing of the neutral point voltage of the NPC inverter. Knowing the fact that the NP voltage is common mode for a three-phase NPC inverter, an active NP balancing strategy was developed to calculate the NP current and subtract it from the common mode interacting current. This significantly improves the balancing response of the NP voltage in comparison to its natural balancing response.

Detailed simulation and experimental results under various operating conditions have been provided for both three-phase NPC and FC inverters.

Chapter 7

Description of Simulation and Experimental Systems

Implementation of the new two-level and three-level hysteresis current regulators has involved two key stages of implementation, simulation, and experiment. Throughout this thesis, simulation and experimental results have been provided at the end of each chapter, to confirm the performance of the proposed hysteresis current regulator over a wide range of operating conditions.

This chapter consists of two main sections. The first section describes the simulation systems that have been developed to examine the performance of the new hysteresis current regulators. Simulation systems are first developed to provide an initial confirmation of the research idea and the primary regulator performance. Next, the second order effects such as sampling and dead-time effects and any further optimizations are integrated into the simulation system to achieve a performance that closely matches the anticipated experimental conditions. Hence, a detailed examination of the research idea and controller performance can be undertaken before building the physical experimental system.

The second section describes the experimental systems that have been used for the final stage of investigations. This allows the operation of the proposed hysteresis current regulator algorithm and the simulations to be verified in real practical systems. This is an important final stage to explore second order effects which are well known to potentially limit the performance of the real experimental system. These effects can then be compensated to achieve as close a match as is possible to the ideal simulation performance. This section includes the description of the hardware platforms with further design, developments, and modifications of the existing analog/digital circuitry, together with the examination of the practical second order effects.

7.1 Simulation Systems

Simulation is an essential tool in the development of power electronic converters and their associated control and modulation. It is an essential step to validate theoretical developments and provide a time efficient and cost effective method to fully explore new concepts and examine the circuit operation before building the experimental prototypes. In this study, the Powersim (PSIM) simulation package is used - a time based circuit simulation software specifically designed for power electronics, motor control and dynamic system simulation.

This section discusses the simulation systems used in this study.

7.1.1 Simulation of Two-Level Single-Phase and Three-Phase VSIs

Figure 7.1(a)(b) shows the topologies of the two-level single-phase and the three-phase VSIs respectively. The single-phase VSI consists of two single-phase legs while the three-phase VSI consists of three phase legs, where each phase leg is a series combination of two N-channel MOSFETs with their anti-parallel diodes. These voltage source inverters are connected to two separate DC power supplies connected in series to form the DC bus with a mid-point (defined as the ground connection) as a reference to measure the analog and digital signals. Since the MOSFET is operating in its ideal condition, there is no operational difference in comparison to other power switches such as IGBTs. These switches turn ON when the gate signal is a logic ONE and they turn OFF when the gate signal is a logic ZERO.

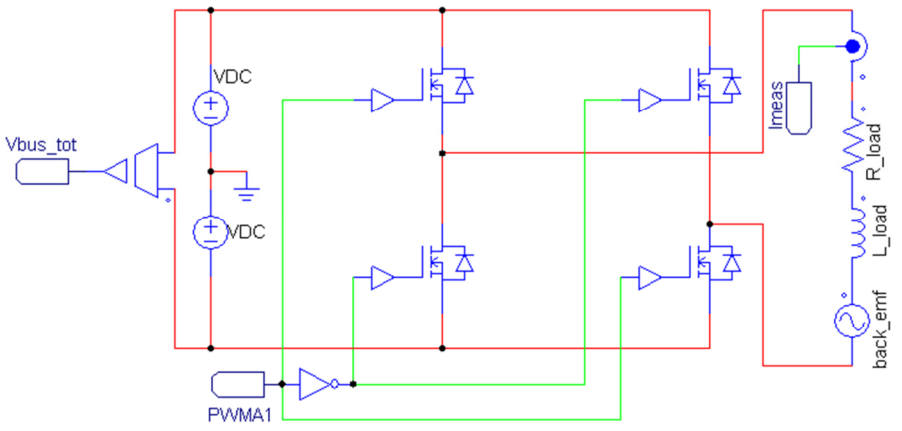
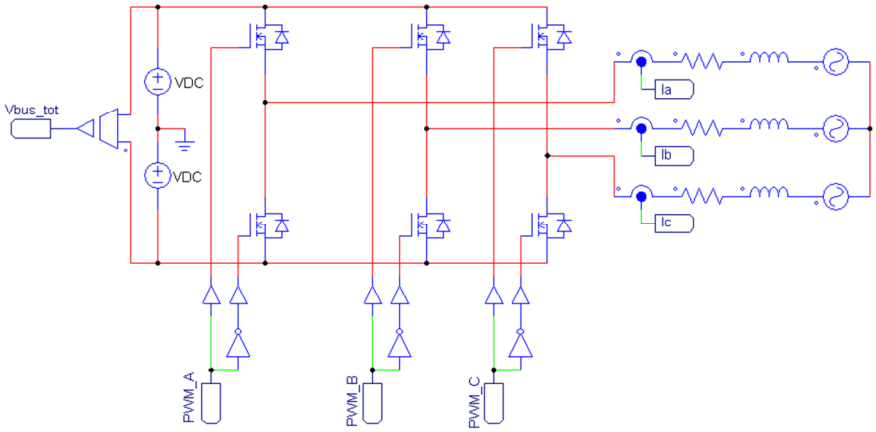


Figure 7.1: Simulation circuit diagram of (a) two-level single-phase VSI (H-bridge)



(b) Simulation circuit diagram of a two-level three-phase VSI

7.1.2 Simulation of Single-Phase Leg and Three-Phase NPC and FC VSIs

Figure 7.2 shows the topology of a three-level single-phase leg NPC, single-phase leg FC, three-phase NPC and three-phase FC inverter. The single-phase NPC and FC inverters consist of one phase leg each formed by stacking two two-level half bridges on top of each other. There are four power switches per phase leg (MOSFET or IGBT) each with their associated anti-parallel diode. One side of the output load is

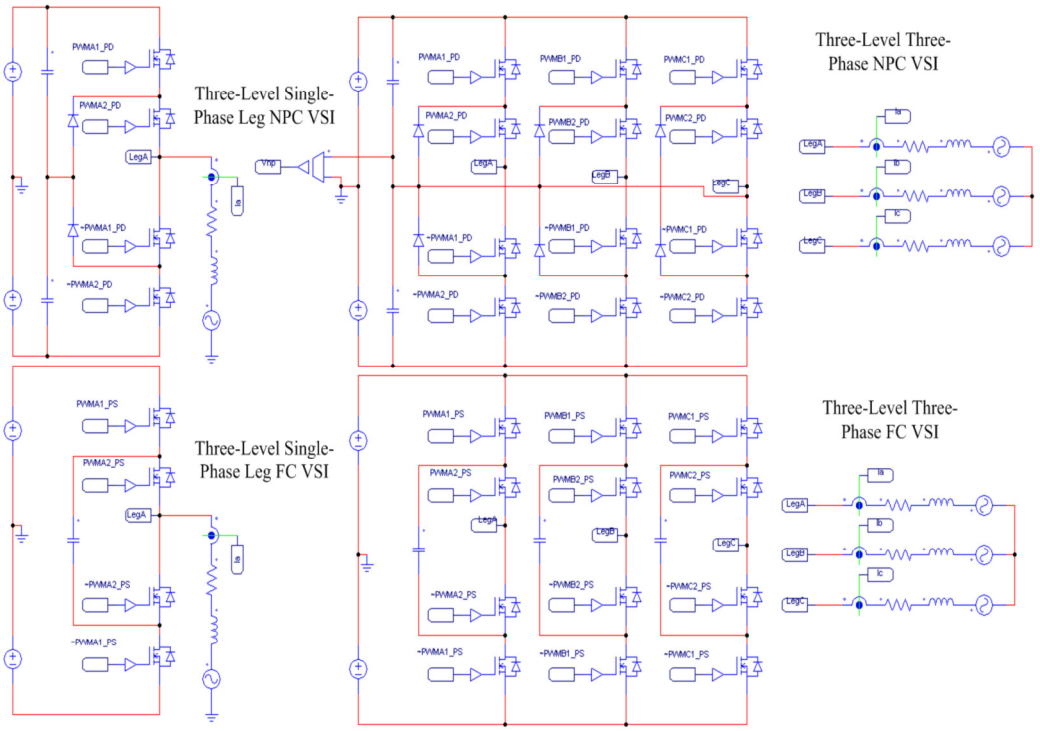


Figure 7.2: Simulation circuit diagram of a three-level single-phase leg NPC and FC inverters, three-level three phase NPC and FC inverters

connected to the output node of these inverters and the other side is connected to the mid-point of the DC supplies. The three-phase system consists of three-phase legs connected to an isolated three-phase RL load with a series back-emf.

The output currents of these VSIs are measured using a PSIM current sensor and its output is used for the hysteresis current control circuitry. For the NPC active balancing strategy, the neutral point voltage is measured using a differential voltage sensor and its output is fed back into the hysteresis current control circuitry.

7.1.3 Simulation of the Two-level Single-Phase Hysteresis Current Regulator

Figure 7.3 shows the simulation circuit diagram of the ideal new hysteresis current regulator applied to the two-level single-phase voltage source inverter (Figure 7.1(a)). The controller structure is divided into four units as follows:

- Average voltage and zero-crossing calculation unit
- Band offset calculation unit
- Variable band calculation unit
- Hysteresis comparison and switching process unit

The ***Average Voltage and Zero-Crossing Calculation Unit*** includes a DLL block which allows DSP C code to be written, compiled and linked to PSIM. The DLL C code implements the timer interrupt service routine (ISR) and the capture port ISR which will be discussed later in this chapter. The first input to the DLL block is the clock which emulates the DSP timer interrupt. It has a frequency of twice the VSI switching frequency. The second input to the DLL is the VSI gate switching signal which is used to extract the average inverter output voltage to calculate the variable hysteresis band using equation (3.19). The linear extrapolation from (3.21) is done within the DLL to get the nearest approximation of the extracted average voltage to the true average voltage.

Next, the DLL calculates the time information of the current error zero-crossing using (3.27) and (3.28) to calculate the band offset using the switching edge timing and the internal counter. The outputs of the DLL block are the normalized average inverter output voltage and the time difference between the current error zero-crossings and the emulated DSP fixed reference clock.

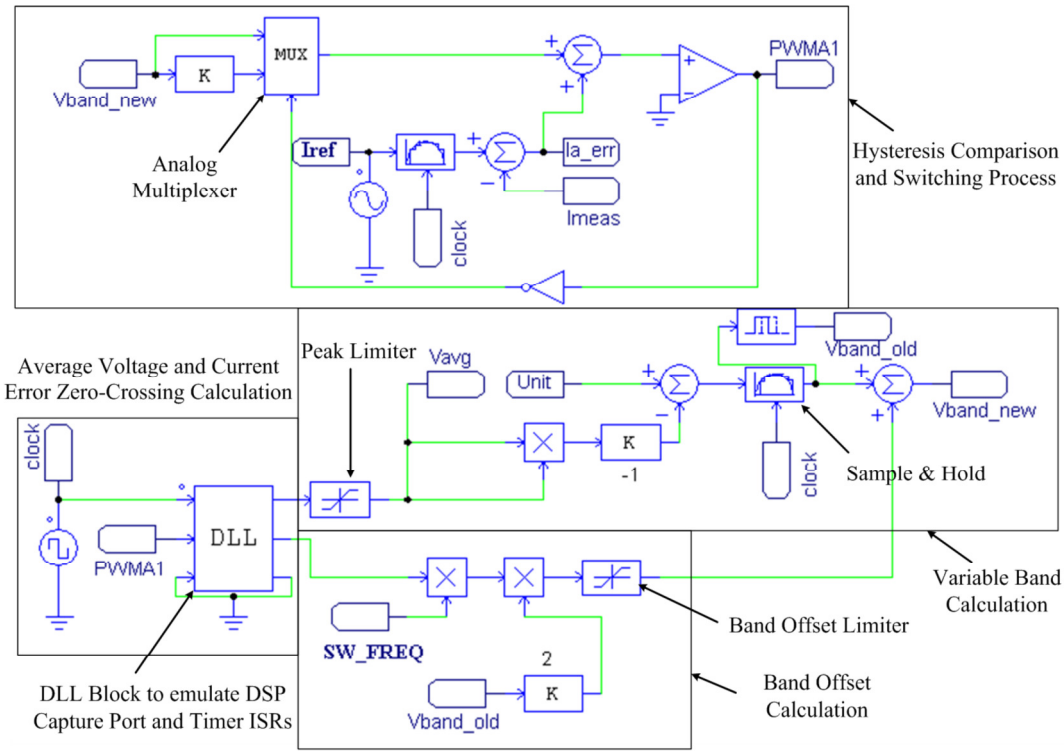


Figure 7.3: Simulation circuit diagram of the proposed two-level hysteresis current regulator for the single-phase VSI

The ***Band Offset Calculation Unit*** calculates the additional hysteresis band offset required to synchronize the current error zero-crossings to the DSP time interrupt, based on equation (3.25).

The ***Variable Band Calculation Unit*** calculates the variable hysteresis band from equation (3.14) using the output of the DLL block, which is the normalized average inverter output voltage. The calculated variable hysteresis band is then sampled using a sample and hold block to simulate the effect of sampling in the real practical platform. The output of the *sample and hold unit* is then delayed by one sample period using a *delay block* to generate the previously calculated variable hysteresis band for the band offset calculation unit. The new variable hysteresis band is then calculated based on equation (3.26).

The variable band calculation unit and the band offset calculation unit can be both implemented in the same DLL block that is used for the average inverter voltage and zero-crossing calculation.

The ***Hysteresis Comparison and Switching Process Unit*** is responsible for the comparison of the reference current and the actual load current to generate the current error. The reference current is sampled using a *sample and hold unit* to

replicate the sampled reference current generated by the DSP and the DAC in the experimental system. The positive and the negative hysteresis bands are selected from the switching state of the gate switching signal (Logic ONE or ZERO) using an analog multiplexer block. The selected hysteresis band and the current error are then summed together and fed into the comparator to generate the gate switching signal.

7.1.4 Simulation of the Two-level Three-Phase Hysteresis Current Regulator

Figure 7.4 shows the simulation circuit diagram of the two-level hysteresis current regulator controller for the three-phase voltage source inverter. The controller circuit schematic per phase leg has been simplified compared with Figure 7.3, since both the band offset calculation unit and variable band calculation unit have been implemented within the same DLL block in the *average voltage and current error zero-crossing calculation unit*. The third phase hysteresis regulator is also replaced with a open-loop sine-triangle PWM where its modulation command is calculated from phase A and B scaled average voltages.

The simulation circuit also shows the calculation of the common mode interacting current. To implement this, the scaled DC bus voltage is calculated using the phase B DLL block, based on the constant system parameters such as load inductance and total DC bus voltage. Next, the scaled bus voltage is inverted and is used by three separate multiplexers in order to replicate the three-phase scaled switched voltages. These voltages are then summed and integrated to calculate the common mode current using a simple summing-integrating circuit.

Also shown in Figure 7.4, there is another DLL block to calculate the third harmonic voltage term from equation (4.39) using the three-phase normalized average inverter output voltages. This voltage is then integrated and subtracted from the common mode interacting current to generate a modified interacting current, using equation (4.42).

Lastly, the hysteresis switching process for the three-phase VSI remains the same as for the single-phase VSI except that the common mode interacting current is subtracted from the total current error before feeding it into the hysteresis comparator.

Figure 7.5 summarizes the complete simulation setup for the two-level three-phase VSI.

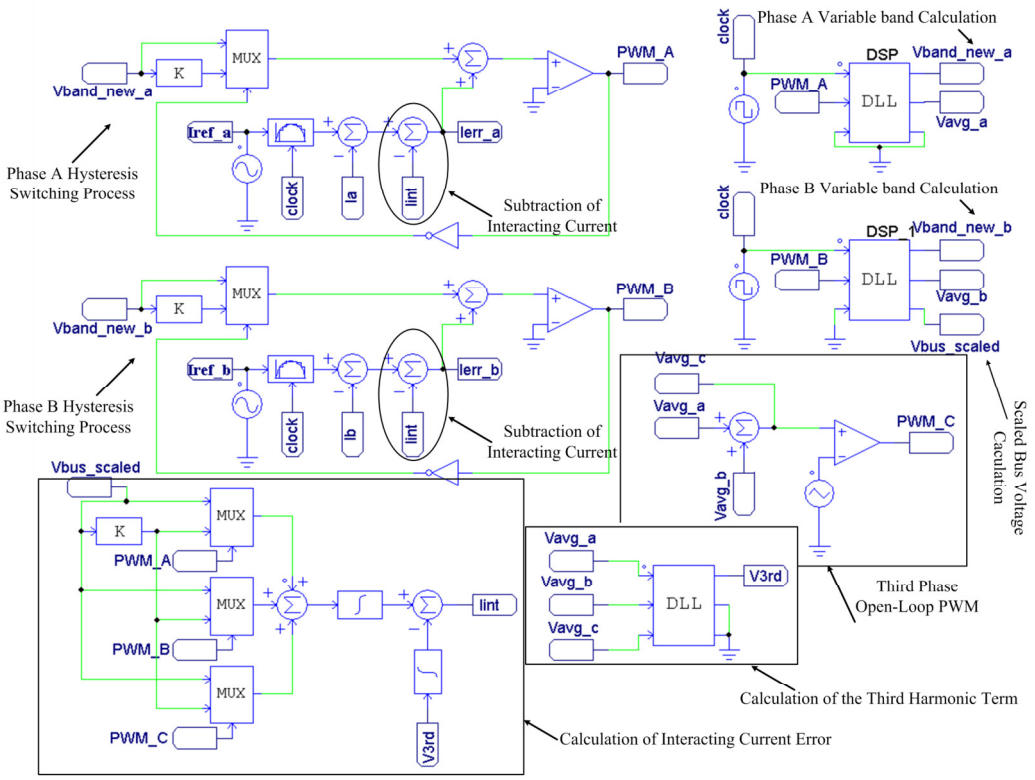


Figure 7.4: Simulation circuit diagram of the proposed two-level hysteresis current regulator for the three-phase VSI

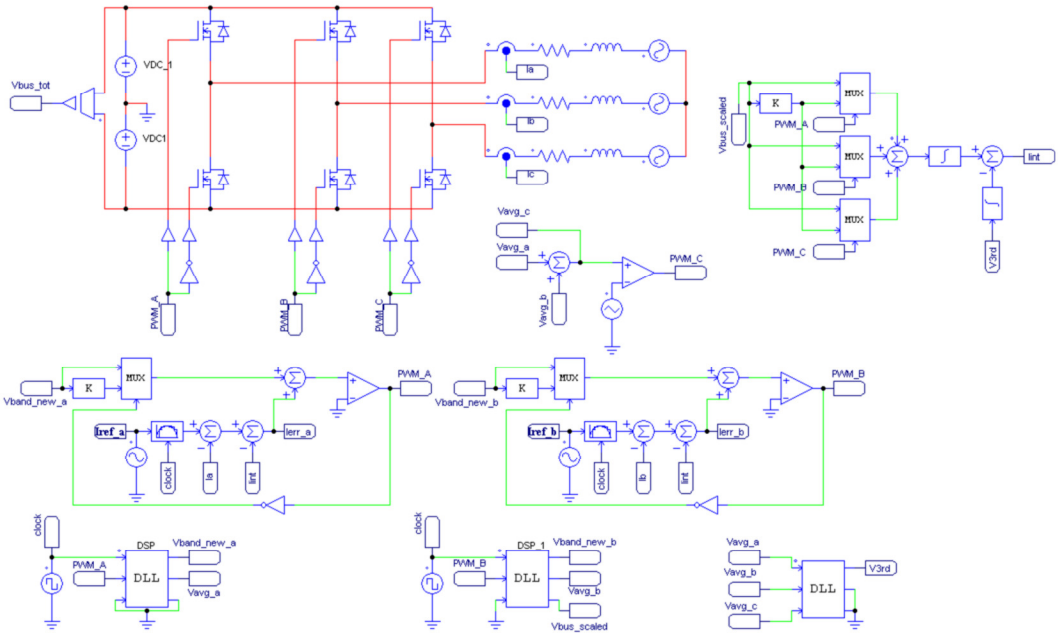


Figure 7.5: Complete simulation setup circuit diagram of the proposed two-level hysteresis current regulator for the three-phase VSI

7.1.5 Simulation of the Multilevel Hysteresis Current Regulator

Figure 7.6 shows the simulation circuit diagram of the new multilevel hysteresis current regulator for NPC and FC inverters. The controller structure is divided into four units as follows:

- Hysteresis comparison and switching process unit
- Logic decoder and FSM unit
- DSP unit to calculate the hysteresis bands, detect the point of output voltage level change and generate the multiplexer inputs
- Calculation of common mode current for three-phase multilevel unit

The ***Hysteresis Comparison and Switching Process Unit*** is responsible for comparing the per phase reference current and the variable hysteresis band using one hysteresis comparator per phase leg to generate the hysteresis switching signals. This is similar to the hysteresis comparison and switching process current regulation of the two-level three-phase VSI described in section 7.1.4.

The simulation schematic can also be used for the single-phase leg NPC or FC inverters, if the common mode interacting current is made zero by removing this

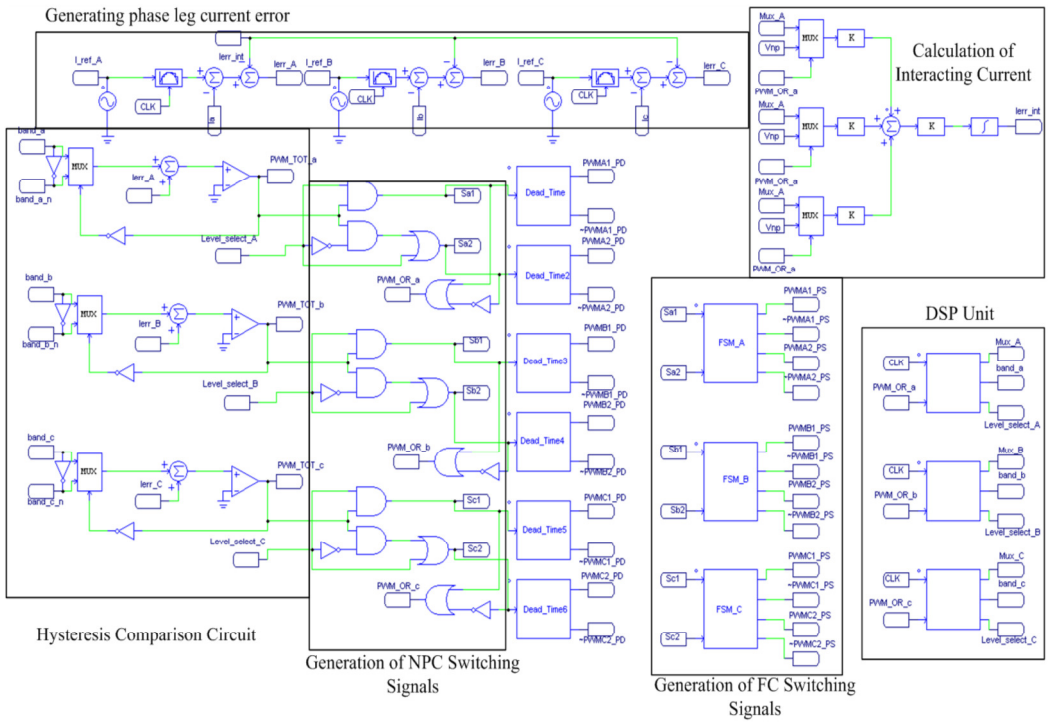


Figure 7.6: Simulation circuit diagram of the proposed three-level hysteresis current regulator for the three-phase three-level VSI

current from the summing junction or replacing it with a ZERO DC voltage source.

The ***Three-Phase Logic Decoder and FSM Unit*** generates the three-phase switching signals for the NPC and FC inverters. The logic decoder is built using PSIM logic elements such as AND, OR and NOT gates. The switching signals generated from the logic decoder are then fed into the dead-time generator to generate the three-phase switching signals for the NPC inverter. The three-phase switching signals for the modulation of the FC inverter are generated using three separate *C blocks*, each responsible to implement the FSM as described in section 5.4.2.2. The dead-time for these switching signals are also generated within the FSM unit. In the experimental system, the logic decoder, the FSM and the dead-time units are implemented within the CPLD as will be discussed later in this chapter.

The ***DSP Unit*** is implemented using three separate C blocks, each belonging to one phase leg, to emulate the operation of the actual DSP in the experimental system. It is responsible for the following tasks:

- Extracting the three-phase absolute normalized average inverter output voltages.
- Calculating the three-phase variable hysteresis bands and the synchronization of the phase leg current error zero-crossing to a fixed reference clock.
- Implementation of the average voltage zero-crossing detection algorithm and generating the three-phase output voltage level selection signals.
- Generating the three-phase inputs for the analog multiplexer.

The first input to the C blocks is clocked to emulate the DSP timer ISR, where its frequency is twice the target VSI switching frequency. The second input is the feedback OR gate switching signal, which extracts its fundamental component to calculate the three-level variable hysteresis band. The C block is also responsible to detect the average voltage zero-crossing and generate the phase leg level selection signal in the same way as discussed in section 5.3. The C block also outputs the multiplexer inputs (Mux_A, Mux_B, Mux_C) which are either $+V_{DC}$ (HIGH) or $-V_{DC}$ (LOW) for the analog multiplexer to replicate the three-level phase leg switched voltage.

The ***Interacting Current Calculation Unit*** shown in Figure 7.6 calculates the common mode interacting current for the three-level three-phase multilevel system. Each DSP C block outputs a MUX output (MUX_A, MUX_B, MUX_C) which is either a positive or negative bus voltage depending on the state of the phase leg level selection signal polarity of the output switched phase voltage. The inputs to the analog multiplexer are these MUX signals and the zero voltage (ground connection) which are selected based on the switching status of the corresponding phase leg feedback OR switching signal.

To implement the active balancing strategy for the NPC inverter, the zero volts can be replaced by the measured NP voltage in a similar way as described in section 6.4. In this way, the three-level three-switched phase voltages will be precisely generated similar to the actual three-phase voltages. These voltages are then summed and integrated to calculate the common mode current using equation (6.6).

Figure 7.7 summarizes the complete simulation setup used for the investigation of the new constant frequency hysteresis current regulation applied to three-level three-phase NPC and FC inverters.

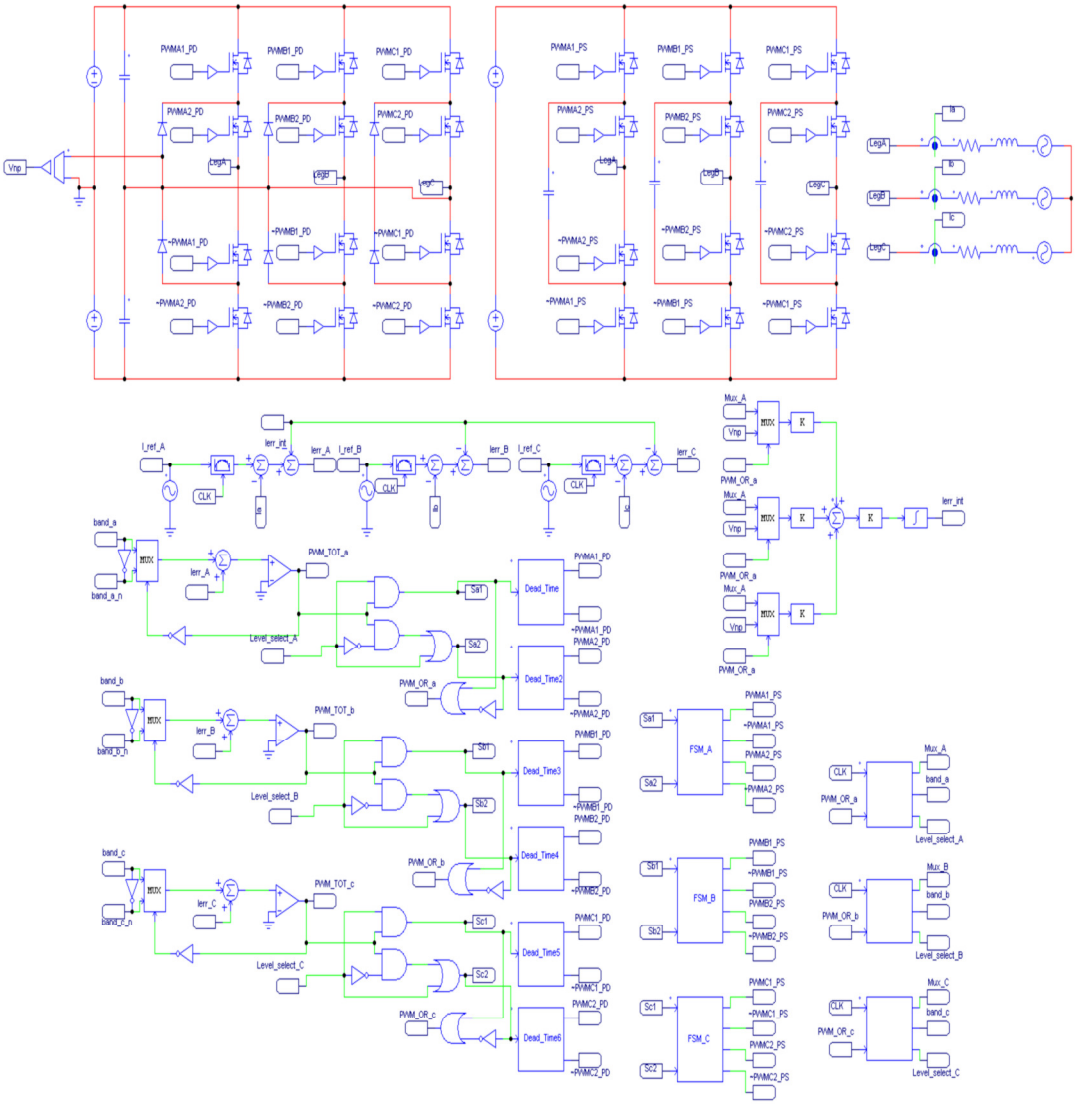


Figure 7.7: Complete simulation setup circuit diagram of the proposed two-level hysteresis current regulator for the three-phase VSI

7.2 Experimental Systems

The theoretical and simulation developments of the new hysteresis current regulation approach for two-level and multilevel inverters are required to be verified in a real experimental system. This is a necessary requirement for the final confirmation of the research ideas since second order effects often limit and degrade the performance of an inverter system and its associated control implementation. Building the real physical system identifies these effects and allows for any corrections and optimizations that are necessary to be made.

The implementation of the new hysteresis current regulator is implemented in two separate stages, being the power board stage and the controller board stage. The basic boards were provided by Creative Power Technologies and the Power Electronics Group at RMIT University. However, additional practical work such as system assembly, integration of the electronic components and building extra electronic circuitry were required to complete the experimental verification of the research work.

7.2.1 Overview of the Two-Level Experimental Systems

Figure 7.8 shows the complete block diagram of the final experimental setup used for the investigation of the new two-level single-phase and three-phase hysteresis current regulator. From this figure, the system consists of four main experimental blocks as follows:

- Power stage: two-level three-phase voltage source inverter.
- Controller cards: DSP and CPLD boards used to implement the hysteresis controller along with switching generations using analog circuitry.
- Load bank: three-phase inductors and resistors. A 10uF AC capacitor is connected in parallel across each phase resistor load to create an effective synchronized back-emf.
- Measurement instruments: differential voltage probes, current probes and digital CROs.

Figure 7.9 shows the photo of the complete experimental setup where each experimental block is marked accordingly. The controller cards consist of three

separate boards CPT-DA2810, CPT-MINI2810 and CPT-IIB which are commercially available from Creative Power Technology (CPT). Figure 7.10 shows how the CPT cards are stacked on top of each other to form the final controller platform shown in Figure 7.8. An isolated RS-232 serial port is used to connect to a personal computer to the overall converter structure, which acts as the Human Machine Interfaces (HMI). This interface allows the user to directly communicate with the experimental converter via a Hyper Terminal Program.

The CPT-DA2810 [120] is a low cost standardised DSP controller board designed to provide a fully flexible interface between the TMS320F2810 DSP processor and a user customised motherboard. This controller board is based on a Texas Instruments TMS320F2810 DSP, which is operating at a clock speed of 150MHz. The CPT-DA2810 provides the following functions:

- Analog to digital conversion (ADC)
- PWM outputs
- RS-232 for serial communication
- Capture port
- Execution of the C code to Implement the control algorithms

The CPT-MINI2810 [121] acts as an interface board between the CPT-IIB power stage board and the CPT-DA2810 DSP controller card. The functionality of the controller card which is used in this work includes:

- Buffering, level shifting and offsetting analog inputs from sensors to the correct voltage level for the hysteresis analog circuitry and ADC system.
- Analog circuitry for generating the current error and hysteresis switching process.
- Routing of PWM signals from the analog comparators via the CPLD.
- Eight 12-bit DAC outputs for reference currents and hysteresis bands.
- Communication with Mini Bus interface, which is responsible for controlling a number of peripherals on the CPT-IIB board.

The CPT-IIB [122] is a general converter controller support board, and it interfaces with the CPT-MINI2810 and CPT-DA2810 to provide an integrated solution for controlling the power electronic converter. In this work the following functions are used:

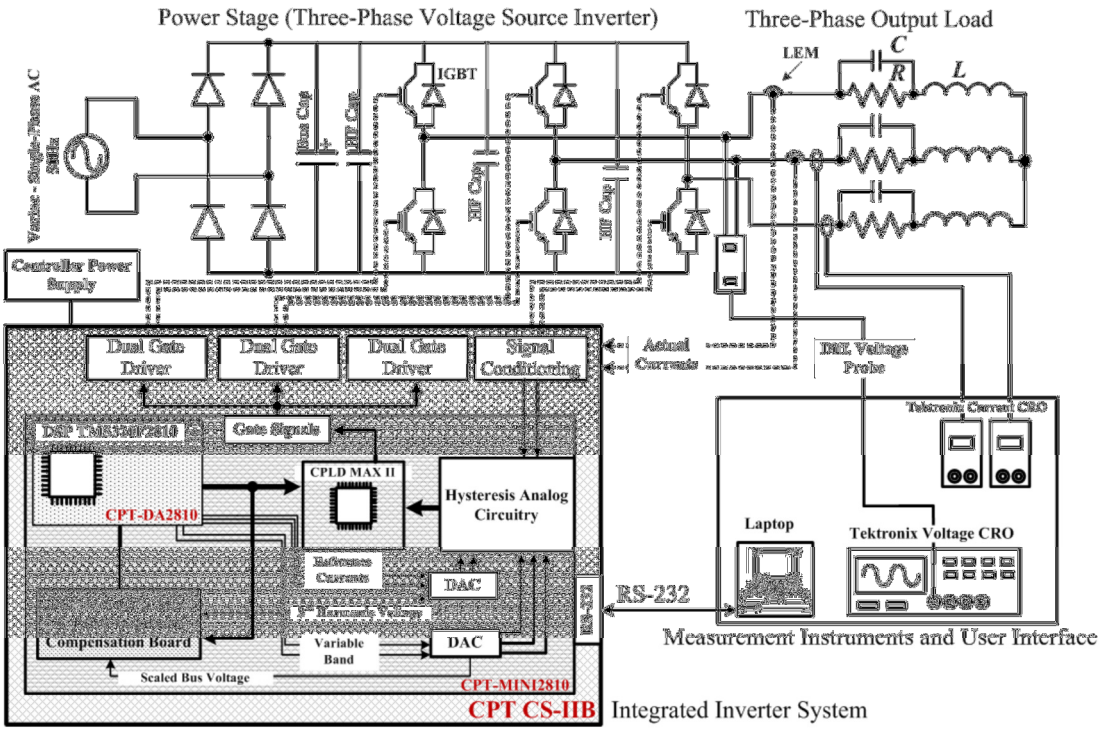


Figure 7.8: Functional and signal flow diagram of the experimental two-level VSI showing the power stage, controller boards, load banks and measuring instruments

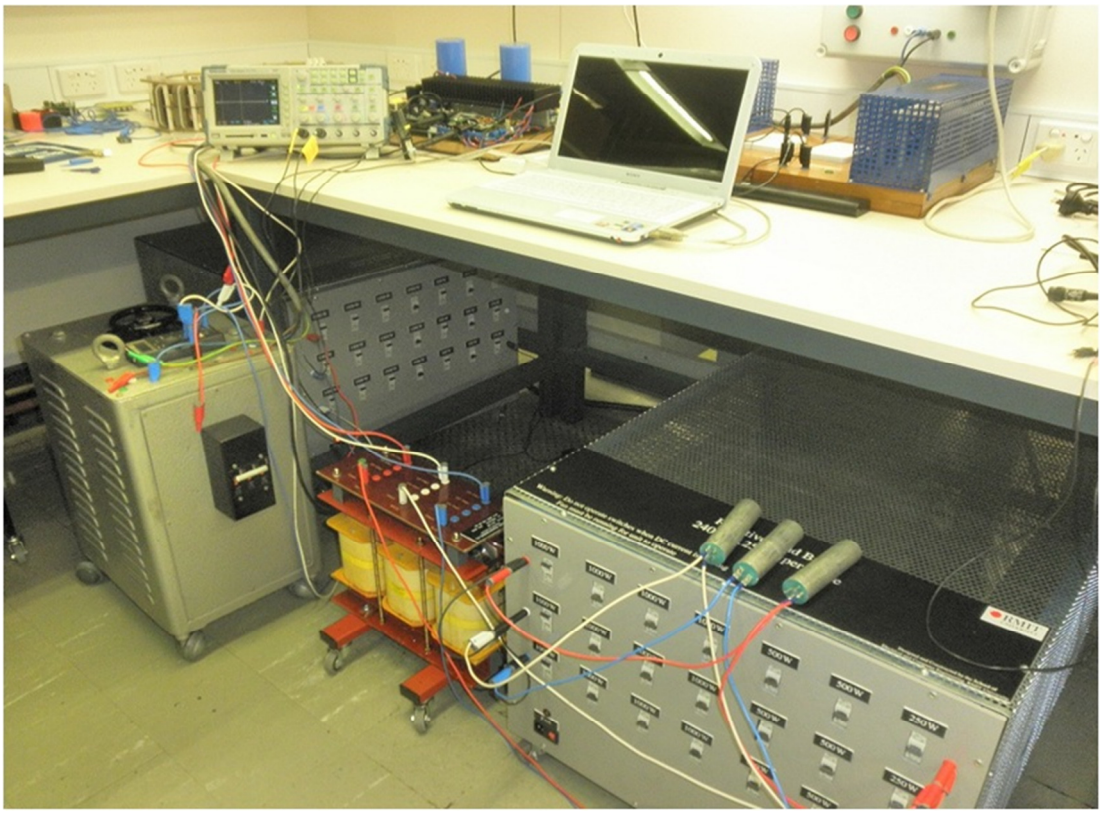


Figure 7.9: Experimental setup for the implementation of the single-phase and three-phase hysteresis regulator showing inverter system, input variac, load banks, voltage and current CRO

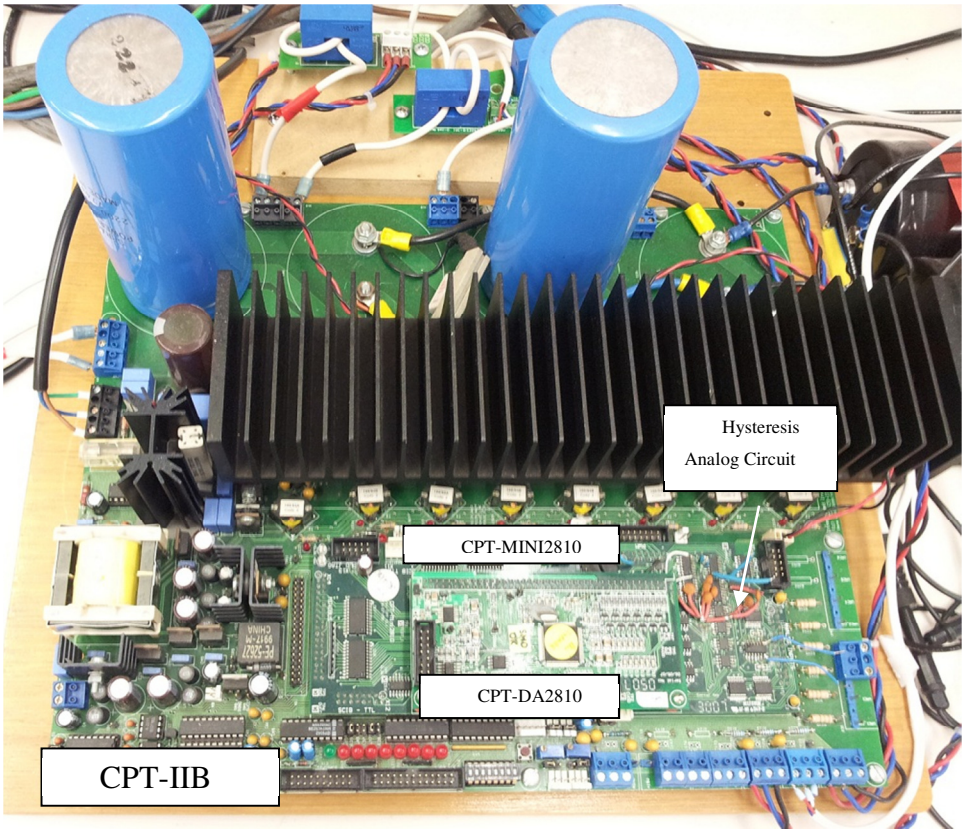


Figure 7.10: Photo of CPT controller cards stacked on top of each other

- Six isolated gate drivers interface circuit are used to modulate the IGBT switches.
- Buffered analog input circuitry with gain and offset circuits.
- Two current sensors, one for each converter phase leg.
- One DC voltage measurement for the DC bus voltage
- Protection circuitry for the over current and over voltage situation.
- On card switched mode power supply (SMPS) to provide rail voltage for the internal analog and digital electronics.
- On-card relay driver for driving the main input circuit breaker.
- Isolated RS-232 communication circuit.

7.2.2 Overview of the Multilevel Experimental Systems

Figure 7.11 shows the complete block diagram of the final experimental setup used for the investigation of the new three-level hysteresis current regulator. Figure 7.12 and Figure 7.13 shows the photo of the complete experimental setup for both NPC and FC systems respectively, where each experimental block is marked accordingly. From these figures, the system consists of four main experimental blocks as follows:

- Power stage: a three-level three-phase NPC and a three-level three-phase FC voltage source inverters.
- Three separate controller boards that are used to implement the hysteresis controller along with switching generation using analog circuitry, DSP and CPLD.
- Load bank: three-phase inductors, resistors and capacitors.
- Measurement instruments: differential voltage probes, current probes and digital CROs.

From this figure, each phase leg of the three-phase system uses separate controller boards configured as one master and two slave boards due to the limited number of available inputs, outputs and gate drivers on each board. Each of these controller boards consist of three separate modules - a CPT-DA2810, CPT-MINI2810 and CPT-GIIB [123], which are commercially available from Creative Power Technology (CPT). The functionality of the CPT-GIIB required for this investigation is similar to the CPT-IIB as described in 7.2.1. However, in contrast to the CPT-IIB, the power stage inverters (NPC and FC inverters) are not included on the power stage board. Instead, the GIIB board provides on-board headers to be used for the external modulation of separate power stages. The functionality of the CPT-DA2810 and CPT-MINI2810 remains similar to what has already been discussed in section 7.2.3. Three separate isolated RS-232 serial ports are used to connect the master-slave modules to a personal computer to act as the Human Machine Interfaces (HMI) for the real-time monitoring of the system status and measurements.

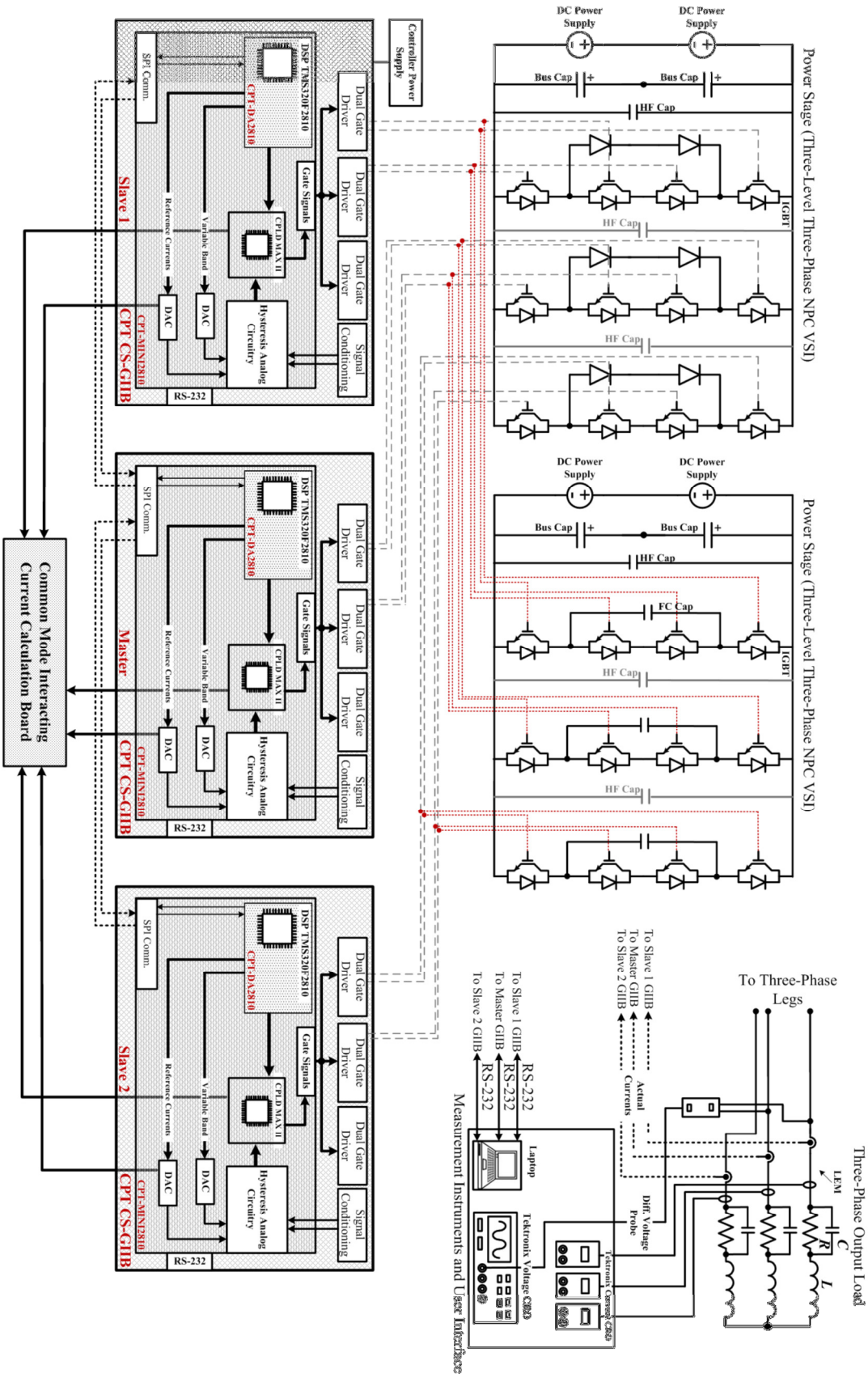


Figure 7.11: Functional and signal flow diagram of the experimental two-level VSI showing the power stage, controller boards, load banks and measuring instruments

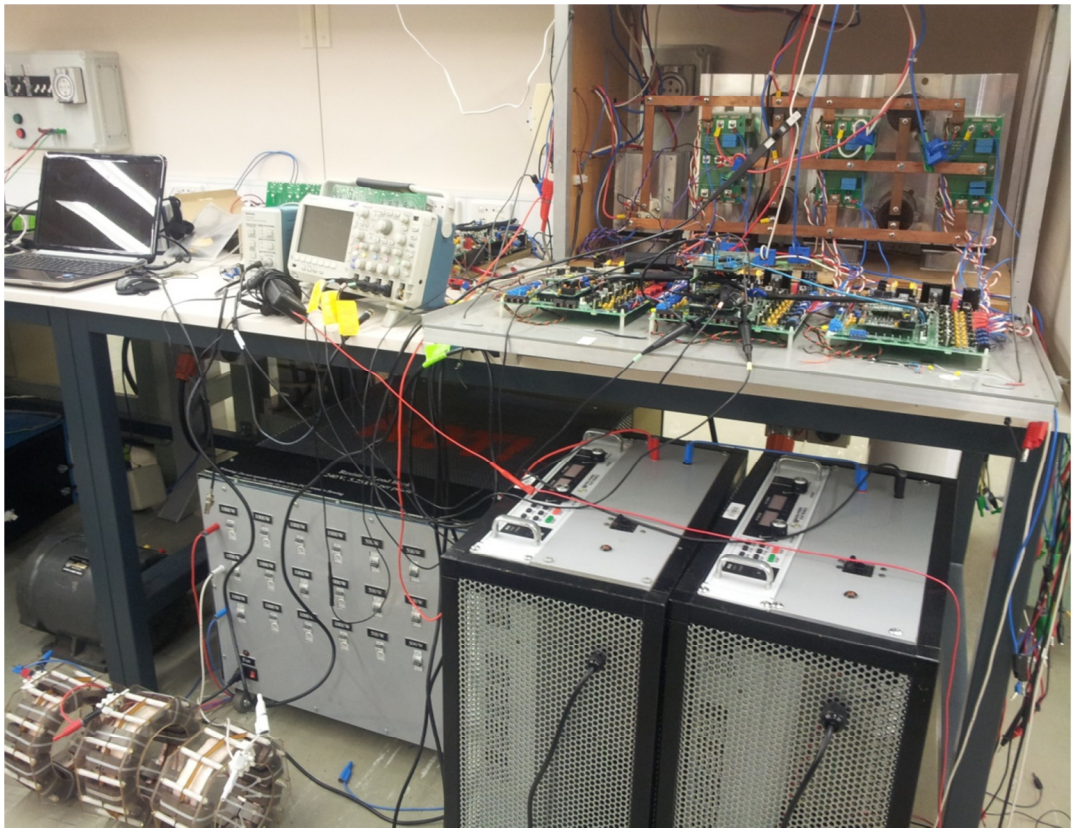


Figure 7.12: Experimental setup for the implementation of the three-phase NPC Inverter

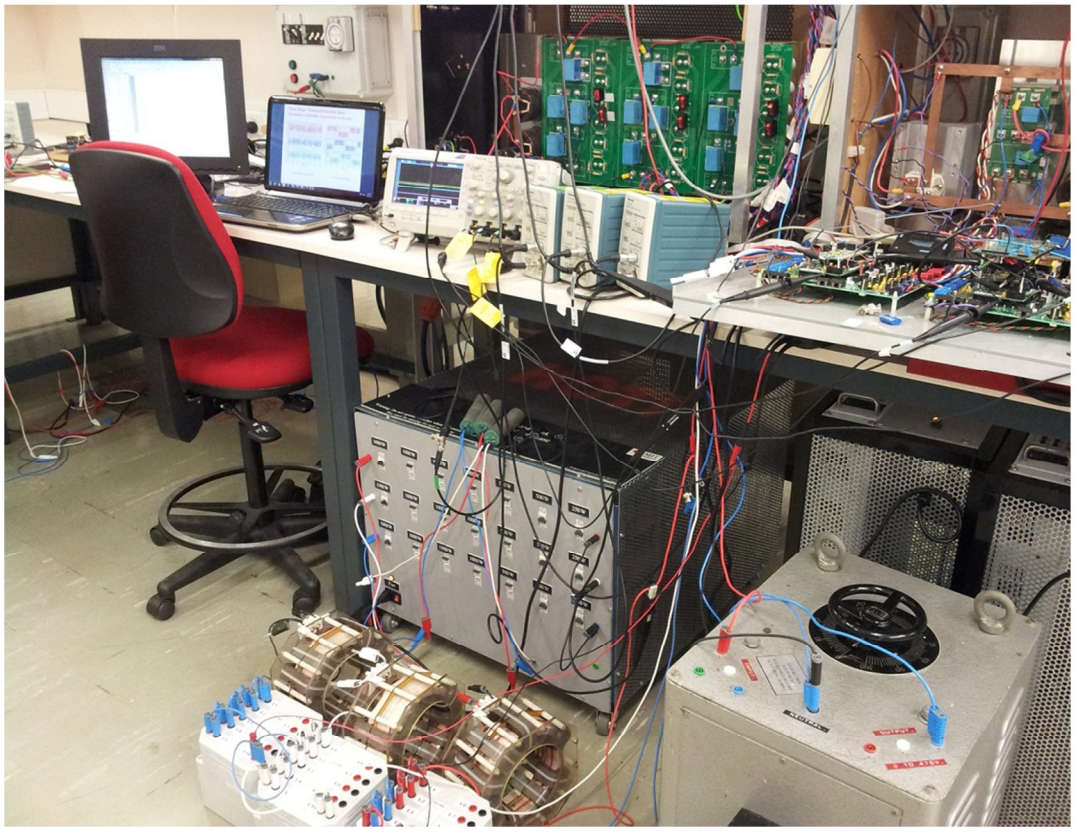


Figure 7.13: Experimental setup for the implementation of the three-phase FC Inverter

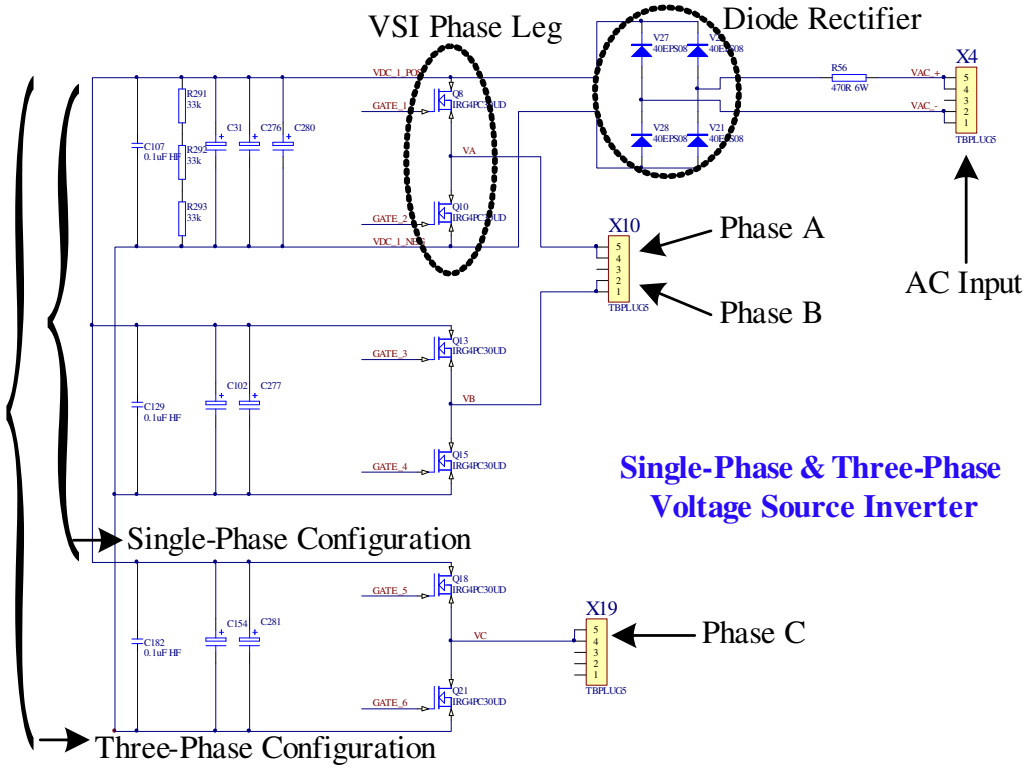


Figure 7.14: Circuit schematic diagram of the CS-IIB power stage showing the wiring configuration of single-phase and three-phase VSI

7.2.3 Experimental Power Stage for the Two-Level Three-Phase VSI

The CS-IIB is a flexible, low voltage, 4-phase leg integrated inverter system that includes an on-card FET/IGBT based power stage. It can be configured both as a single-phase and a three-phase VSI to be used with the plug-in DSP and CPLD controller cards to create a standalone voltage source inverter without additional circuitry. The power stage of the CS-IIB is capable of operation up to 15A at up to a 400V DC bus, and is designed for TO-247 FET/IGBT based switching devices. The power stage supply is supplied from a single-phase variac and full-bridge diode rectifier as shown in Figure 7.8. The circuit schematic diagram of the voltage source inverter available on CS-IIB platform is shown in Figure 7.14. Also, note that for the single-phase inverter, any of the two inverter phase legs can be used provided that the output load is accordingly connected to these phase legs using the on-board connectors.

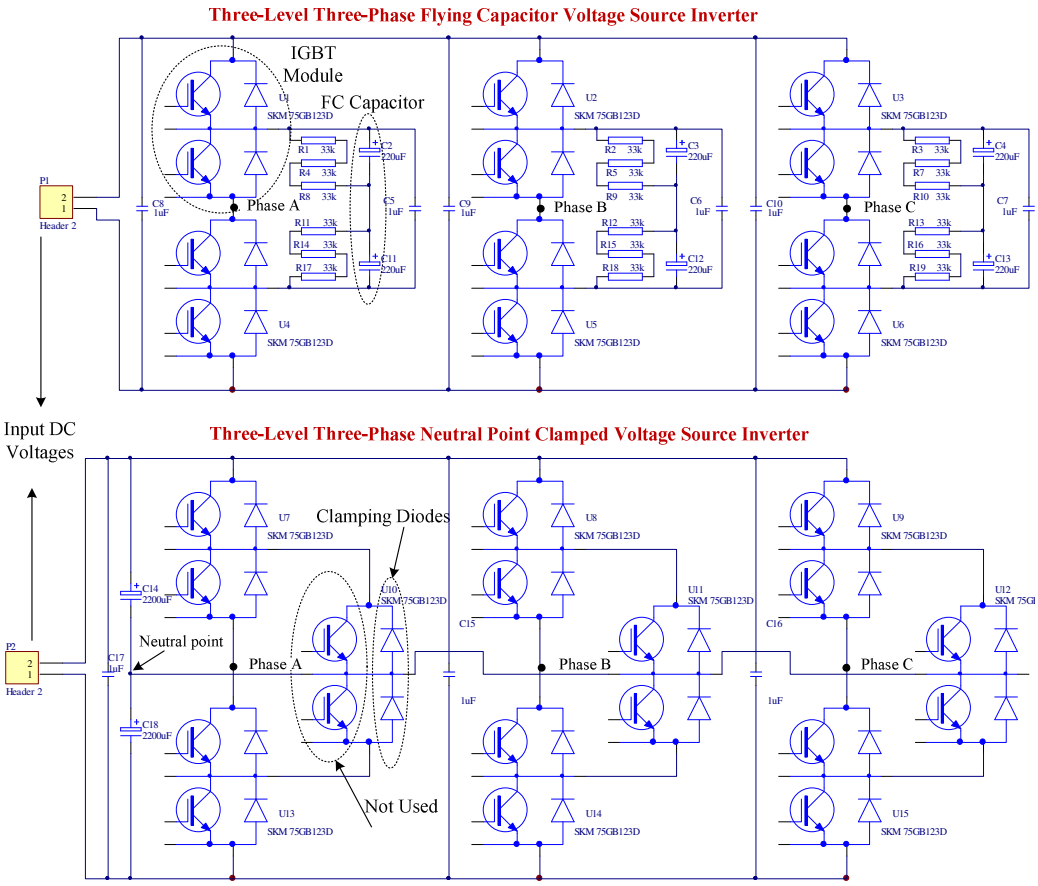


Figure 7.15: Circuit schematic diagram of three-level three-phase NPC and FC voltage source inverters

7.2.4 Experimental Power Stage for the Multilevel Three-Phase NPC and FC VSIs

Figure 7.15 shows the circuit diagram of thee-level three-phase flying capacitor (top) and three-level three-phase neutral point clamped inverters (bottom). These topologies are constructed using individual IGBT modules (SKM 75GB123D) where each module consists of two IGBTs and their associated anti-parallel diodes. For the NPC inverter, the clamping diodes are constructed using one IGPT module per phase leg. For any of these topologies, the three-phase output load is connected to the middle of each phase leg. These power stages are then externally connected to the CPT-GIIB controller board for the hysteresis modulation process.

The DC input voltages are connected through the external connector using two separate DC power trolleys. In this form a virtual zero voltage is created to measure the phase, line and neutral point voltages. For the NPC inverter, the bus capacitors are 4400uF to minimise the ripple on the neutral point voltage.

7.2.5 Experimental Implementation of the new Two-Level and Three-Level Hysteresis Regulators

The experimental system was implemented as a combination of analog and digital circuitry with a supporting TMS320F2810 DSP and a MAX II CPLD. For the two-level system, the hysteresis current regulator was implemented using three CPT cards (CPT-DA2810, CPT-MINI2810 and CBT-IIB) and one additional circuit board to calculate the interacting current board.

For the implementation of the three-phase multilevel system, each phase leg employs its own set of CPT cards (three CPT-DA2810 cards, three CPT-MINI2810 cards and three CBT-GIB cards). Modifications were made to the two-level interacting current calculation board to measure the common mode current for the three-level three-phase system. Also, a master-slave communication protocol was used for the synchronization and data transfer between these boards.

The rest of this section describes the details of the on-board analog and digital systems along with the software algorithm that has been used to implement the proposed two-level hysteresis controller.

7.2.5.1 Analog Processing Circuitry

The actual load currents are measured by the off-card Hall effect current transducers (LEM) whose output signals are conditioned and buffered using burden resistors and differential amplifiers before being used by the hysteresis switching circuitry. The analog circuitry used to implement the controller is available on CPT-MINI2810 and it consists of four main sections as follows:

1) Reference Current and Hysteresis Band Generation Using a DAC:

Figure 7.16 shows the analog circuitry used to generate the phase A reference current and the variable hysteresis band. For the hysteresis current regulator, the target reference currents $I_A^*(t)$ (DAC4 output) and the per phase positive and negative variable hysteresis bands $+I_{h,A}(t) - I_{h,A}(t)$ (BAND1 and BAND2 outputs) are generated using two quad 12-bit digital to analog converters (DAC). Each DAC has four outputs and they are conditioned to generate a bipolar $\pm 10V$ that is accessed via the DSP serial peripheral interface (SPI). The DSP SPI is responsible for transmitting the calculated quantized data from the DSP to the DAC by selecting

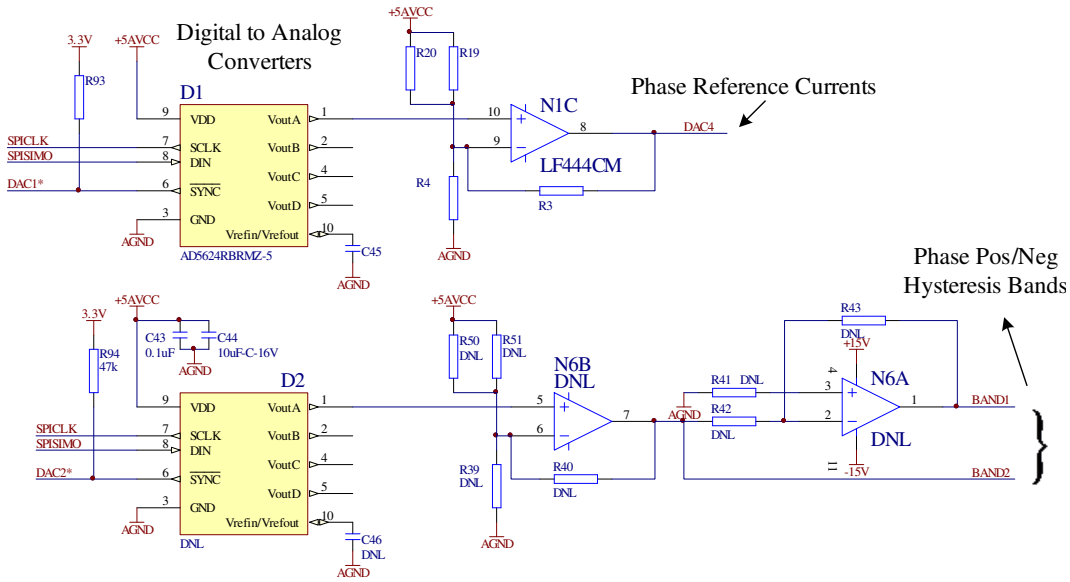


Figure 7.16: Circuit schematic diagram of the reference currents and hysteresis bands generation using DACs and op-amps

the appropriate DAC output channels. For the two-level system, the remaining outputs of the two DACs are used to generate the scaled bus voltage and the third harmonic term as will be discussed later in this chapter. For the three-level three-phase system, the remaining DAC output is used to generate the neutral point voltage of the NPC inverter for the active balancing strategy.

2) Current Error Generation Using Differential amplifiers:

Figure 7.17 shows the analog circuitry (phase A in this figure) which calculates the difference between the per phase reference current and the per phase measured load current to generate the per phase current error. For the three-phase hysteresis current regulator, a summing junction is added to the actual load current at the negative terminal of the differential op-amp to subtract the modified common mode interacting current $\gamma'(t)$ from the phase reference current to generate the non-interacting current errors $e_A(t)$ and $e_B(t)$. For the implementation of the hysteresis regulator in case of the two-level single-phase and three-level single-phase leg, only one of the differential amplifiers is used and the additional summing junction for the common mode current is not physically connected.

3) Hysteresis Switching Process Using Multiplexer and Comparator:

Figure 7.17 shows the per phase analog circuitry for the hysteresis switching process using a hysteresis comparator and analog multiplexer. The resultant per phase non-interacting currents from the output of the differential amplifiers and the hysteresis bands are fed into a summing junction through the series resistors at the positive input of the hysteresis comparator. In this way, the comparator switches its output when the input of the positive comparator terminal crosses zero. The comparator outputs (HB1, HB2) are then post-processed by CPLD based dead-time logic counters to generate the per phase final complementary phase leg switching commands. The analog multiplexer is responsible to select either the positive or negative hysteresis band depending on the switching status of the hysteresis comparators. The analog multiplexer has two channels where each channel has four differential inputs connected through to a common output. Note in this circuit, the inputs of each channel are connected together in such a way to form two differential inputs to a common output. Table 7.1 summarizes the operation of the analog multiplexer based on its inputs and outputs. From this table, depending on the logic status of the two-bits binary address A0 and A1 connected to the output of the hysteresis comparators (HB1, HB2), the positive or the negative hysteresis band per phase leg is selected to bring the summing junction voltage back toward zero.

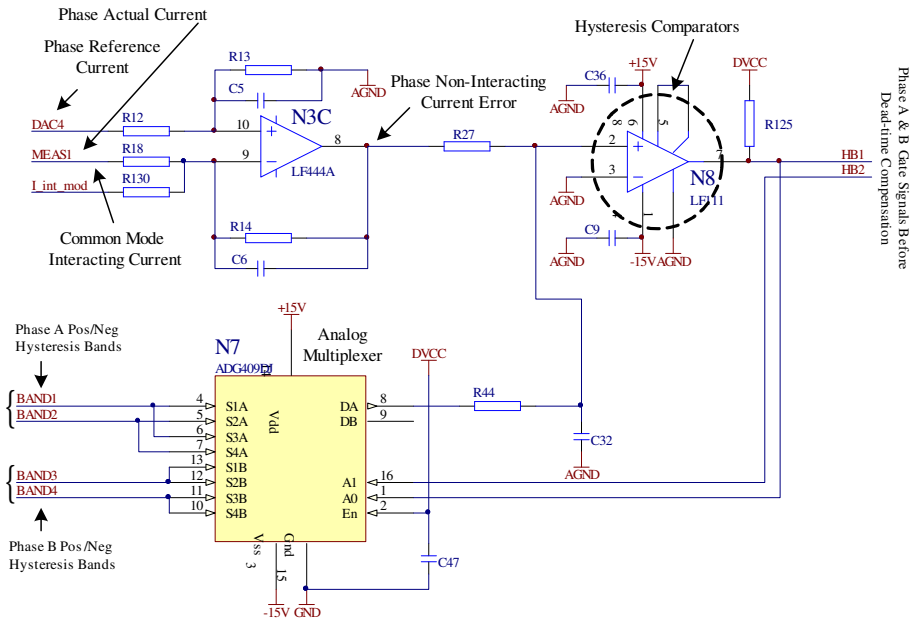


Figure 7.17: Circuit schematic diagram of the hysteresis switching process using hysteresis comparators and analog multiplexers.

Table 7.1: Operation of the analog multiplexers selecting the appropriate hysteresis band based on the comparator switching states.

Phase A Comparator	Phase B Comparator	Hysteresis Band
0	0	Band 1 & 3
0	1	Band 1 & 4
1	0	Band 2 & 3
1	1	Band 2 & 4

4) Generation of the Common Mode Interacting Currents:

Calculation of Common Mode Current for Two-level Three-Phase System

In order to generate the common mode interacting current, it is necessary to measure the three-phase voltages using voltage sensors. However as discussed in the chapter 4, a technique was developed to replicate the three-switched phase voltages using the three-phase switching signals. As shown in Figure 7.18, the per phase leg switching command (before applying dead-time) is fed into the analog multiplexer to switch between the positive and negative DC bus voltage, scaled to include the constant system parameters. The scaled DC bus is then generated using the DAC, for the analog multiplexer circuit. As a result, the three multiplexed outputs represent the instantaneous values of the actual phase leg switched voltages, which are then summed and integrated using an analogue summing-integrating op-amp circuit to generate the common mode interacting current. Note that this arrangement allows the DSP to tune the scaled bus voltage DAC output to match any particular load inductance, without requiring hardware circuit changes.

Figure 7.18 also shows the analog circuitry used to inject the third harmonic voltage in order to extend the modulation range. The DSP calculates the third harmonic offset V_{3rd} from the three-phase measured average inverter output voltages. The DSP then outputs this voltage using an intermediate DAC into the second stage integrating-differential amplifier. The output of the differential amplifier is then subtracted from the common mode interacting current to generate the modified common mode interacting current $\gamma'(t)$. This current is then fed back into the summing junction of the actual load current shown in Figure 7.17 to create the non-interacting current error for the final hysteresis switching process.

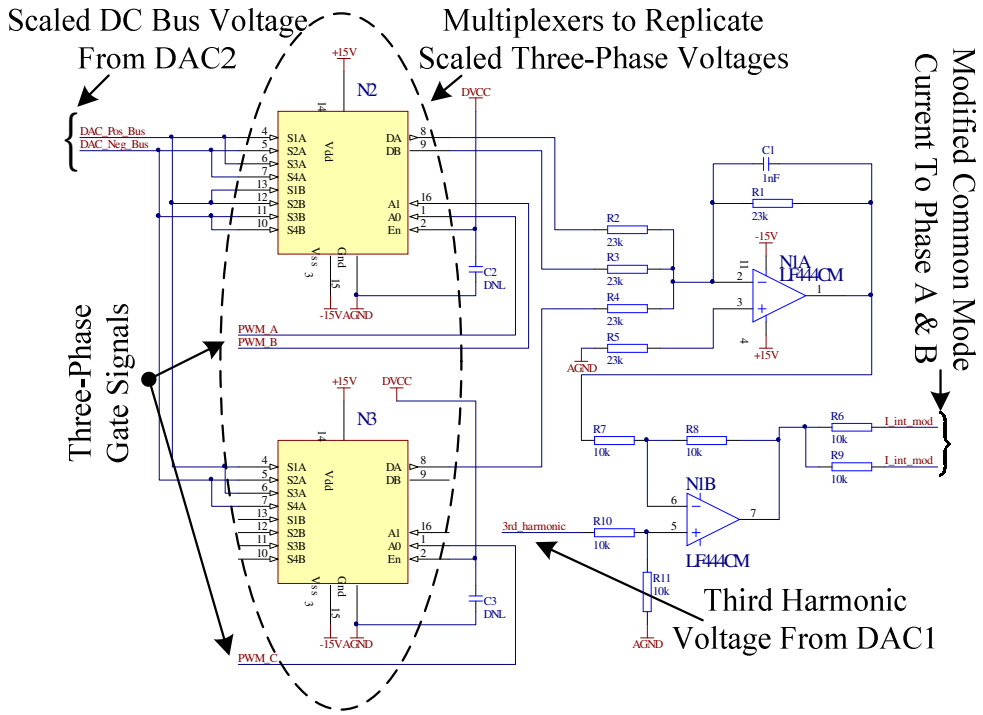


Figure 7.18: Circuit schematic diagram used to generate the two-level common mode interacting current from three-phase switching signals, analog multiplexers and op-amps.

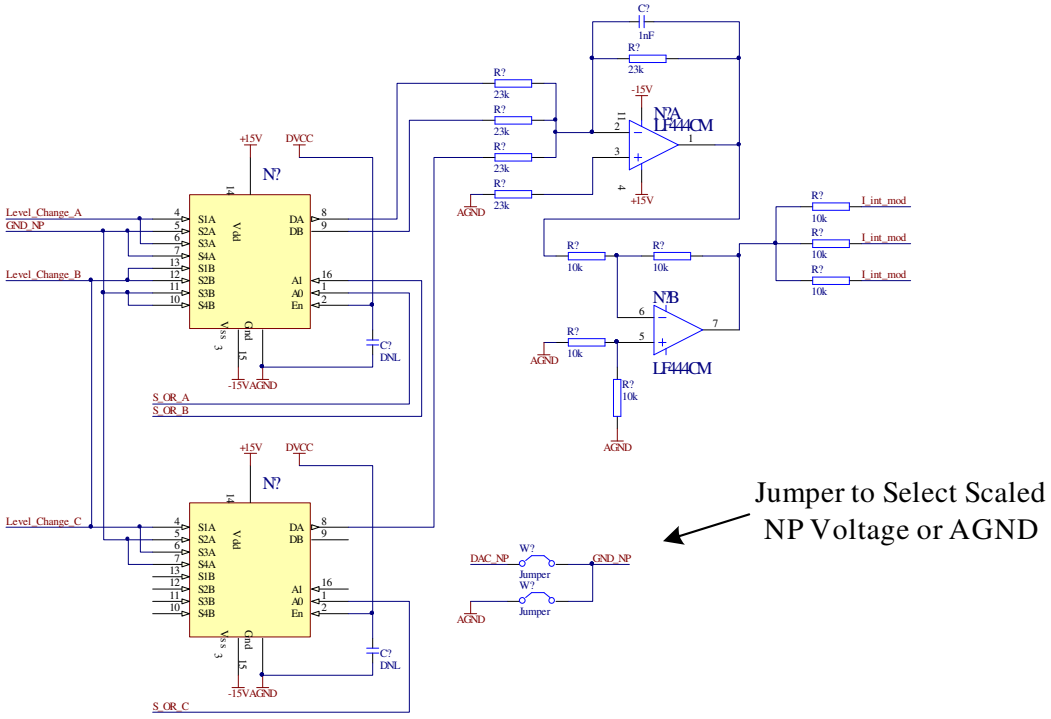


Figure 7.19: Circuit schematic diagram used to generate the three-level common mode interacting current from three-phase switching signals, analog multiplexers and op-amps.

Calculation of Common Mode Current for Three-level Three-Phase System

Figure 7.19 shows the modified circuit schematic in Figure 7.18 to calculate the common mode interacting current for a three-level three-phase inverter. This is done by replacing the negative scaled bus voltage with ground connection (GND_NP) to create the ZERO switching state of the three-level switched phase voltage. The positive scaled bus voltage is connected to three-phase level change signals (Level_Change_A, Level_Change_B, Level_Change_C) where their amplitude will be switched between scaled positive bus voltage and the negative scaled voltage when the corresponding zero-crossing of the per phase average voltage is detected. The output of the analog multiplexer is then equivalent to the scaled three-level three-switched phase voltages depending on the switching state of the phase modified hysteresis switching signal (S_OR_A, S_OR_B, S_OR_C). These voltages are then summed and integrated in the same way as calculating this current for the two-level three-phase inverter.

To implement the active balancing strategy for the three-phase NPC inverter, the ground connection of the analog multiplexer is now replaced with the scaled NP voltage as described in section 6.4. The NP voltage is calculated using ADC measurement of the split DC bus voltages with respect the neutral point voltage. The NP voltage is then scaled and is output to the additional DAC channel. From Figure 7.19, it can be seen that the selection between the scaled NP voltage and the ground connection is implemented using two additional jumpers. For the three-phase FC inverter, the jumper connections will always remain connected to the analog ground.

7.2.5.2 Digital Processing Circuitry

The digital processing circuitry is implemented using the TI DSP TMS320F2810 and the MAX II CPLD. Figure 7.20(a) shows the overview of the communication between the DSP, CPLD and the DACs for the two-level system. Figure 7.20(b) shows the overview of the communication between the DSP, CPLD and the DACs for the multilevel system. For the two-level system, the proposed hysteresis regulator is implemented within one controller card. For the multilevel system, each phase leg has its own controller card, which communicate with each other using a master/slave communication model. The detail functions of each of these processes will be discussed in this section.

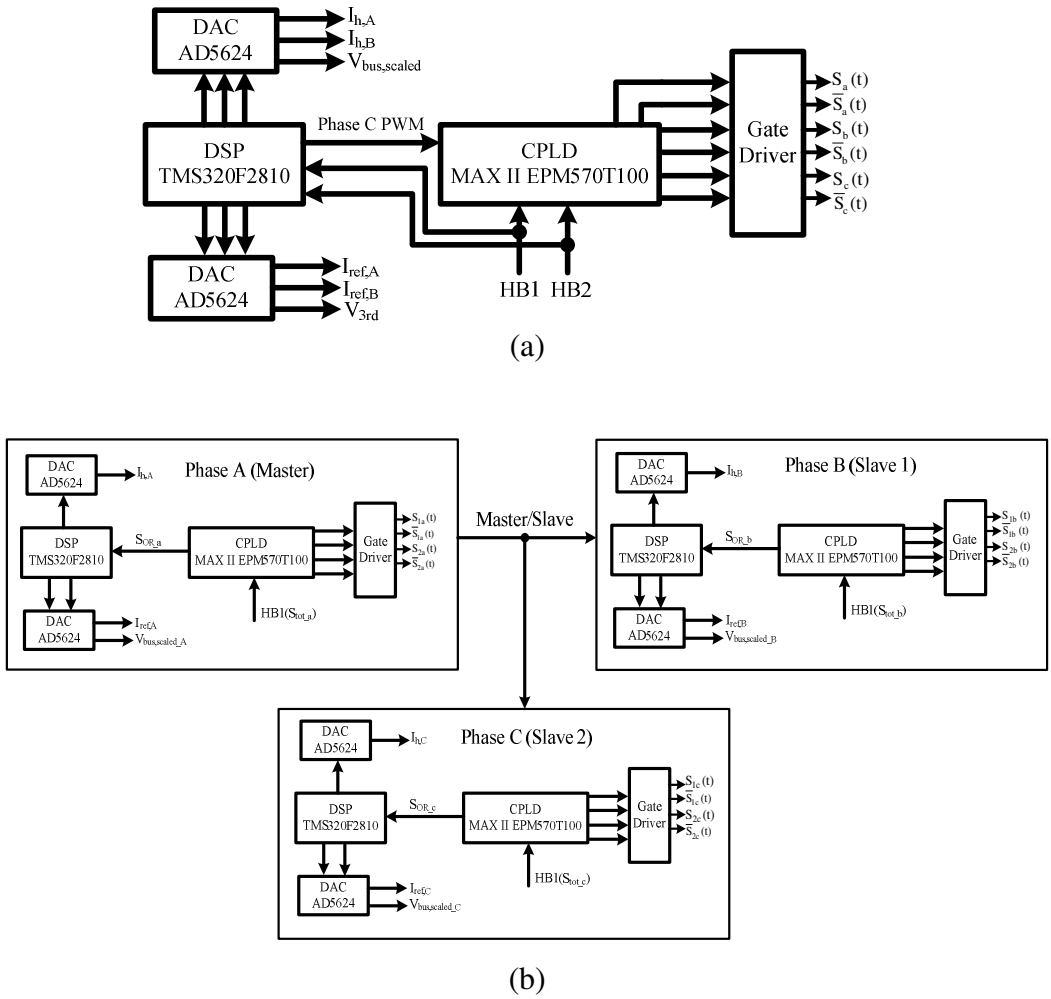


Figure 7.20: Overall view of inter communication between DSP, CPLD and the system peripheral (a) two-level system (b) multilevel system

1) MAX II EPM570T100 CPLD

The CPLD is located on the CPT-MINI2810 board that is accessed from the CPT-DA2810 DSP board using the SPI communication. The switching pulses that have been generated by the hysteresis comparison of the phase current error and the hysteresis bands (Phase A and Phase B) as well as the third-phase switching signal (generated from the DSP by open-loop PWM), are fed into the CPLD MAX II as shown in Figure 7.8. The CPLD introduces a programmable dead-time and creates the complimentary pair of switching signals to be used by the gate drivers for the pulse width modulation of the VSI's phase leg.

For the three-level inverters, the three-phase hysteresis switching signals that are generated from the phase leg hysteresis comparator, along with the level selection signal are fed into the CPLD as shown in Figure 7.11. These signals are then used by the logic decoder and the finite state machine to generate of the corresponding phase leg switching signals using the digital logic circuit arrangements shown in Figure 7.11.

2) DSP TMS320F2810

DSP Tasks for Two-Level Hysteresis Implementation

The DSP is located on the CPT-DA2810 board and is mainly responsible for the execution of the C program to implement the new hysteresis current regulators. The tasks that are performed by the DSP are divided into the high priority foreground and low priority background tasks.

The foreground tasks are managed using the two system interrupts - timer interrupt and the capture port ISR. The following tasks are handled within the timer ISR:

- Calculation and DAC writing of the reference currents.
- Calculation and DAC writing of the variable hysteresis bands.
- Calculation of the band offset to synchronize the current error zero-crossing to a fixed reference clock.
- Calculation and DAC writing of the third harmonic voltage.
- ADC measurement of DC bus voltage and DAC writing of scaled DC bus voltage to the analog multiplexers.
- Open-loop sine-triangle PWM of third phase leg.

The capture port interrupt is responsible for the following tasks:

- Calculation of the phase leg normalized average inverter voltage.
- Calculation of the current error zero-crossing time information.

Figure 7.21 shows the relative time occurrence of the timer ISR and the capture ISR along with the tasks that are executed within each of these ISRs. The timer interrupt has the highest priority and operates on a fixed frequency that is twice the target VSI switching frequency (5 kHz in case of $f_{sw} = 2.5kHz$). The capture port interrupt has a lower priority and its frequency varies depending on the rising or falling edges of the phase leg switching signal.

(a) Timer Interrupt Service Routine:

The sinusoidal reference currents are generated based on the target phase angle of each phase reference currents (0° for phase A, 120° for phase B and 240° for phase C). The DSP then calculates the index pointer for the existing Sine Table within the DSP ROM, to obtain the correct sinusoidal value for each phase. The Sine Table has 2048 data entries and linear interpolation is used to improve the accuracy of the sine table values. The value of the Sine Table entries are then scaled using the desired peak amplitude that is set by HMI via the RS-232 communication to be able to control the desired output current amplitude.

The DSP completes six DAC writes within one interrupt cycle as follows:

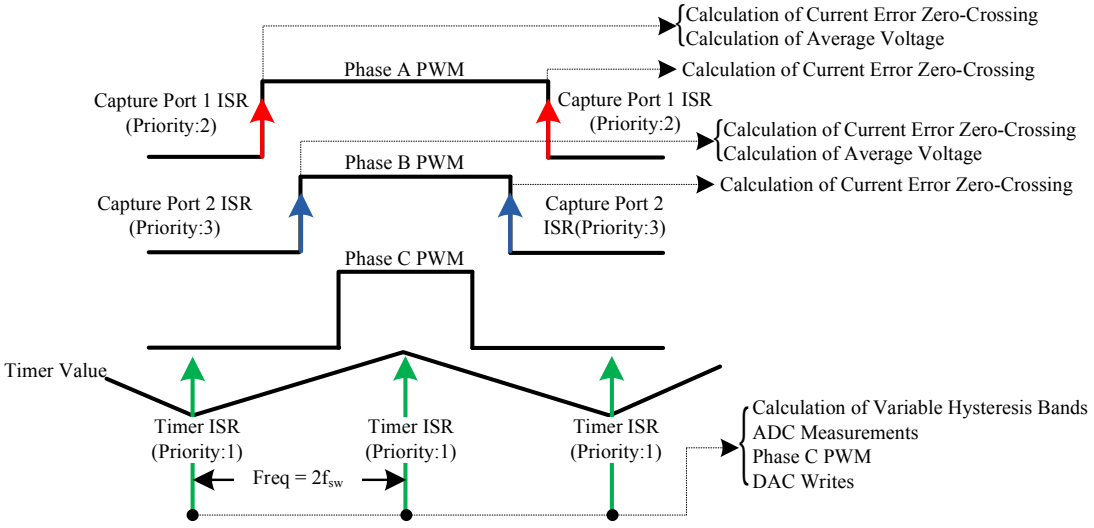


Figure 7.21: Occurrence of the timer ISR and the capture ISR and tasks completed within each of these ISRs.

- Two DAC writes for the reference currents.
- Two DAC writes for the variable hysteresis band.
- One DAC write for the third harmonic voltage.
- One DAC write for the scaled DC bus voltage.

The DSP communicates with DAC by clocking three bytes of data to the SPI interface to write the data into the DAC 24 bit shift register.

The calculation of the required band offset in order to synchronize the current error zero-crossing is done at the beginning of the timer ISR where the fixed reference clock is the timer ISR and the information of current error zero-crossing is provided by the previous capture ISR. Finally, the third phase modulation command is calculated from the phase A and B average voltage information and the values are stored into the COMPARE registers for the PWM peripheral in the DSP to generate the third phase switching signals.

(b) Capture Port Interrupt Service Routine:

In order to implement the new hysteresis current regulation approach, it is necessary to measure the switching edge time information (both for the rising and falling edges) of the phase leg gate signals. The capture interrupt service routine will be triggered when a switching edge is detected and it simply captures the timer counter value and store it into the capture FIFO register. These values are then used to calculate the gate signal fundamental component and the time information of the current error zero-crossings, which are then used by the timer ISR for the variable hysteresis band calculation.

(c) Background Tasks:

The background task runs in a continuous loop and it is executed whenever the system is not running an interrupt. These tasks include processing the user keyboard input, displaying the measured data on the screen and selecting between various peripherals on the controller board such as SPI peripherals.

DSP Tasks for Multilevel Hysteresis Implementation:

For the multilevel system, the DSP board within each controller card generates the corresponding per phase level selection signal, which is then used by the logic decoder within the CPLD. In addition, the DSP generates the scaled level selection

signals and the neutral point voltage from the ADC measurement of the split DC bus voltages and outputs them to the DAC for calculation of the common mode interacting current.

The rest of the DSP functionality remains the same as has been discussed for two-level system.

3) Master/Slave SPI Communication for Multilevel Hysteresis Current Regulation

In Section 7.2.2, it has been shown that to implement the three-phase multilevel hysteresis current regulator, three-separate CPT-GIIB controller cards have been used. Figure 7.22 shows the overview of the communication interface between the three controller cards. The model of communication that is used to transmit data between the controller cards is Master/Slave where phase A GIIB board is considered as the Master board and Phase B and C are considered as the two Slave boards. The communication protocol uses the DSP Serial Peripheral Interface (SPI) to transmit data between the controller cards that runs at a clock speed of 7.5MHz.

The SPI communication is responsible for the following tasks:

- To ensure synchronized operation between the three separate controller cards.
- To transfer system data across the controller boards for necessary calculations.

Synchronization of the three controller cards is an essential requirement for the

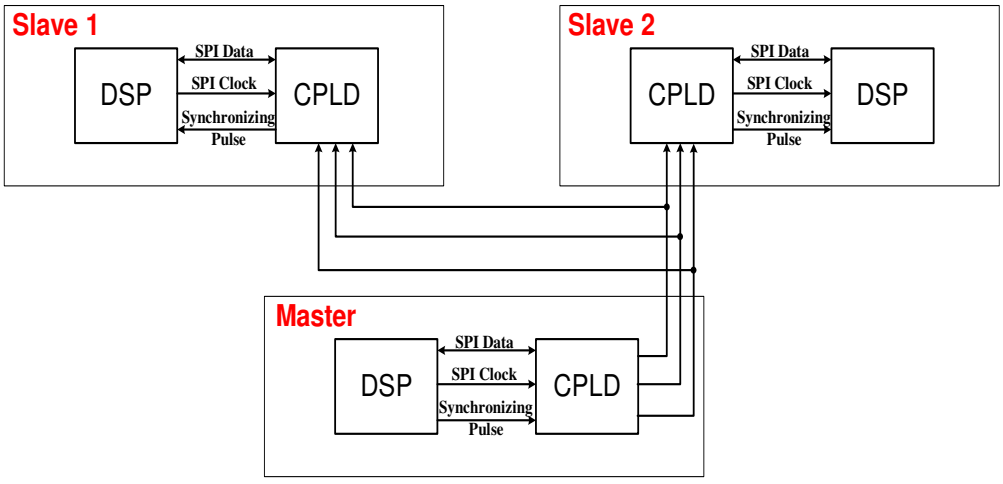


Figure 7.22: Master/Slave controller communication interface overview

following purposes:

- Generating the synchronized three-phase reference currents that are displaced by 120°.
- Synchronizing the three-phase current error zero-crossings to a fixed reference, which is same across the controller cards.

The synchronization is done by generating a short pulse within the timer ISR of the master board by toggling a digital I/O. This pin is then connected to an additional capture port on the slave boards. Once the rising edge of the pulse is detected, the slaves ISR start their interrupt and capture ISRs. If the slave carrier phase leads the master carrier, the period of the slave period is slightly reduced until it synchronizes again. The slaves always check this edge to ensure synchronized operation across all the three controller cards. If this edge is not detected, the slaves consider it as a fault situation and then report a fault state to the master board to stop the VSI operation.

In addition to the synchronization process, the real system parameters need to be transmitted between the controller cards. For this work, the master board is responsible for the following tasks by transmitting the essential data to the other two slave boards:

- Global enable/disable for the inverter system
- Transmitting the peak commanded reference current $I_{ref,max}$.
- Transmitting the maximum allowable hysteresis band $I_{h,max}$.
- Transmitting the scaled DC bus voltage
- Handling the fault state by commanding the slaves to disable their switching.

Each slave is also responsible for the following tasks:

- Calculating and DAC writing of its phase reference current.
- Calculating and DAC writing of its phase variable hysteresis band.
- DAC writing of its POS/NEG scaled DC bus voltage for calculation of the common mode interacting current.
- Managing local functions such as current sensors, report of fault to master board and switching of the corresponding phase leg.

The communication process in this way ensures to minimize the transmission and control delays between the controller cards, which can degrade the performance of the current regulator. The data transfer is initialized by sending the clock signal to the SPICLK pin of the master or slave boards. For the slave board, the data that is stored in the internal SPI buffer register of the master SPI, is shifted into the SPISIMO (Slave In Master Out) pin to be read by the timer ISR. For the master board, the data from the internal buffer register of the slave SPI is clocked into the SPISOMI (Slave Out Master In) pin to be read by the timer ISR. Figure 7.23 shows the experimental confirmation of the synchronization process across all three boards. Figure 7.24 shows experimental confirmation for the data transfer from master to the slaves where data is transferred once the rising edge of the SPI clock is detected

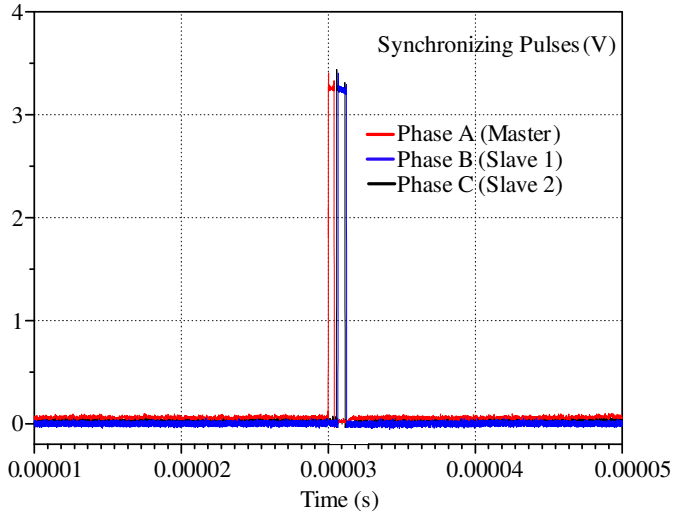


Figure 7.23: Synchronizing pulses for three controller boards. Master pulse (red trace) is received by slaves (blue and black traces)

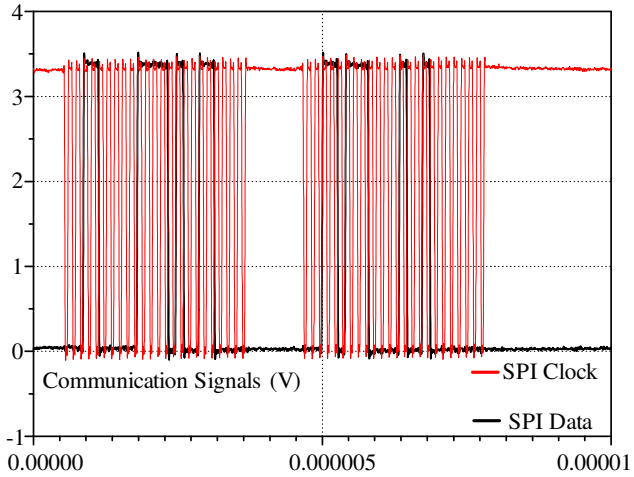


Figure 7.24: Communication signals. SPICLK (red trace) and SPI MOSI (black trace) transferring commanded peak reference current in two bytes from

7.3 Summary

This chapter has provided a detailed description of simulation and experimental systems that have been used develop the new two-level and three-level hysteresis current regulators.

The simulation system was developed to investigate the performance of the proposed hysteresis current regulation approach under various operating conditions for both two-level and three-level voltage source inverters. Initially, the simulation studies were performed with ideal system models to assist in the confirmation of the theoretical knowledge developed without being overburdened by physical implementation issues. Subsequently, the practical second order effects such as quantisation and sampling delays were introduced to generate useful insights necessary for fine-tuning the experimental modulator.

This chapter continued with detailed descriptions of the experimental systems used for the research investigations presented in this thesis. The two-level experimental systems were a combination of both analog and digital circuitry using a CPLD and a DSP platform to implement the proposed hysteresis current regulator. The same structure was extended to three-phase multilevel inverters where each phase leg uses its own controller board. The phase legs communicate with each other using the Master/Slave communication model that is responsible to synchronize the controller cards and transmit real system data across the three controller boards.

Chapter 8

Conclusions

Hysteresis current regulation of voltage source inverters has been the subject of substantial research in Power Electronics since late 1980's. The overall aim of any hysteresis current regulator is to achieve a harmonic performance that is similar to the high performance open-loop modulators while maintaining the advantages of a conventional HCC. Additionally, multilevel hysteresis current regulators must maintain balanced NP and FC voltages for the NPC and FC inverters respectively. Various hysteresis strategies have been proposed in the literature to achieve these aims. However, their performance is less robust and restricted due to the following major reasons:

- The switching frequency of the inverter varies according to the change in average inverter output voltage.
- Using the current error derivative and/or direct measurement of the current error zero-crossing to attempt to maintain a constant switching frequency.
- For three phase systems, the controller suffers from adverse interaction between phase legs that causes the current error to exceed the target limits.
- For multilevel inverters, the selection of the output voltage level is dependent on the multiple hysteresis band arrangement, current error derivative and/or zero crossings.

The work in this thesis offers a significant contribution to the field of hysteresis current regulation of voltage source inverters. The fundamentals of the proposed approach are based on a novel strategy to synchronously measure the average inverter output voltage of a VSI. This allows the development of an integrated approach for constant frequency closed-loop hysteresis current regulation of voltage source inverters.

This chapter explains the main findings of this research work.

8.1 Summary of Research and Conclusions

8.1.1 Measuring the normalised average inverter output voltage

A novel approach is presented that uses the phase leg gate signals to synchronously measure the normalised average output voltage of the VSI, without requiring any direct voltage measurements. A linear extrapolation technique has also been applied to the measured average voltages to compensate for the effect of sampling delay.

This technique has the following particular advantages:

1. Avoids both the susceptibility of derivative action on the measurement process and/or any requirement for analog or digital filtering.
2. Avoids errors caused by bandwidth limits of a conventional analog or digital low pass filter are avoided.
3. Analogue voltage measurement circuits are not required to measure the phase leg switched voltage since the normalised average inverter output voltage can be directly calculated from the gate switching signals.

Once the average voltage of the inverter is measured, the development of an integrated hysteresis current regulation approach is presented that achieves a performance similar to the harmonically superior open-loop modulators.

8.1.2 Hysteresis Current Regulation of Two-Level Single-Phase VSI

A new variable band hysteresis current regulator has been developed for a two-level single phase leg voltage source inverter with the following advantages:

- The performance of the variable hysteresis band is essentially independent of the output load/filter parameters and the DC link voltage. While these variables do affect the maximum magnitude of the variable hysteresis band $I_{h.max}$, they do not affect the algorithm's ability to maintain a constant switching frequency. Furthermore, errors in their estimation can be readily compensated to achieve a specific target frequency by tuning the value of $I_{h.max}$ during normal operation.
- The variable hysteresis band calculation is independent of the current error derivative time information of the current error zero-crossing.

- The variable hysteresis band directly incorporates the effect of the normalised average inverter output voltage.
- The inverter switching frequency remains constant throughout the reference target fundamental cycle.

Current error zero-crossings are then synchronized to a reference clock as an additional feedforward step to further improve the frequency regulation of the HCC, minimising the output current ripple, and achieving with this combined strategy a harmonic performance that is very close to asymmetrical regular sampled PWM. The new approach also calculates the zero-crossing time information from the switching time events to avoid the need for additional zero-crossing measurement circuitry. Finally, the synchronization technique is improved by compensating for the effect of dead-time in the inverter switching process.

8.1.3 Hysteresis Current Regulation of Two-Level Three-Phase VSI

The new two-level single-phase leg HCC was then extended to a three-phase VSI by compensating for the common mode current, calculated from the switching information of the three-phase gate signals to avoid requiring direct measurement of the three-switched phase voltages. The phases A and phase B current error zero-crossings were then synchronised to a fixed reference clock. This ensures that the VSI selects the “three nearest space vectors” within each switching cycle. The resultant strategy achieves a harmonic performance that is very close to the open-loop CSVM. Next, the third phase leg is controlled using an open-loop sine-triangle PWM by replacing the third phase HCC with a fixed frequency direct modulator. This reduces the implementation complexity of the three-phase HCC while improving the line-to-line harmonic performance.

The modulation depth of the VSI was extended by injecting a third harmonic component to fully utilize the available DC bus voltage. The third harmonic voltage was calculated using the information of the three-phase average inverter output voltages. Finally, the operation of the proposed hysteresis current regulator was extended into the overmodulation region by clamping the variable hysteresis band while maintaining excellent switching stability.

8.1.4 Hysteresis Current Regulation of Single-Phase Leg Multilevel VSI

The fourth contribution developed a new constant frequency multilevel hysteresis current control approach for three-level VSIs such as a single-phase H-bridge, and for a single phase leg NPC and FC inverter. The proposed controller synchronously measures the average inverter output voltages of a three-level switched voltage and varies the hysteresis bands with an additional band offset to maintain a constant switching frequency. The controller detects the impending polarity change of a three-level switched voltage from the zero-crossing information of the average voltage, without requiring multiple hysteresis bands and/or time information of the current error zero-crossing. This means only one hysteresis comparator is required per phase leg while still eliminating any DC steady-state tracking error. Additionally, the controller ensures robust and reliable operation during the transient and overmodulation region by avoiding the interference that can be caused by multiple hysteresis bands.

Finally, logic circuitry was developed to decode the hysteresis switching signals to generate gate signals that are similar to open-loop modulators. For the FC, a finite state machine uses redundant inverter zero states to maintain a balanced FC voltage without requiring additional voltage sensors.

The resultant switching performance has a harmonic spectrum that is similar to asymmetrical double-edge regular sampled PWM for a three-level single-phase H-bridge and level shifted PWM for the NPC phase leg and phase shifted PWM for the FC phase leg.

8.1.5 Hysteresis Current Regulation of Three-Level Three-Phase VSIs

This fifth contribution extended the variable band hysteresis concept of this thesis to achieve hysteresis current regulation of a three-phase three-level VSI such as a NPC and FC inverter. The proposed controller synchronously measures the three-phase average inverter output voltages and varies the hysteresis bands with an additional band offset for each phase leg to maintain a constant switching frequency. The common mode current was calculated and compensated using the hysteresis switching signals and the level change signals to reconstruct the three-phase three-level switched phase voltages.

The resultant switching performance achieving with this strategy has a performance that is very close to the harmonically superior open-loop PD PWM.

Finally, a novel active strategy was developed to maintain balanced neutral point voltage of the NPC inverter, by injecting the common mode current into the interacting current calculation.

8.1.6 Experimental Confirmation

The operation of the proposed hysteresis current regulator has been verified in a real experimental situation and it has been compared against its simulated performance. This was an important step to verify the robustness and reliability of the new approach against various system parameters and nonidealities such as current sensor mismatch and variation of the circuit parameter such as load inductors and sampling effects.

8.2 Suggestion for Future Work

8.2.1 Discontinuous Modulation of the Two-Level Three-Phase VSI

Another area of research is to implement discontinuous modulation using the hysteresis current regulator presented in this thesis. Discontinuous PWM [11][15] is known to have harmonic advantages in the linear high modulation depth and overmodulation region while reducing the switching losses of the inverter. For conventional sine-triangle PWM, depending on the type of discontinuous PWM, this is done by injecting a common mode offset calculated from the three-phase modulation commands into the reference commands. Since the proposed hysteresis current regulation approach measures these average voltages, the required common mode offsets can be calculated accordingly. These voltages can be then compensated into the common mode interacting current in a similar way as is done for the third harmonic offset compensation discussed in chapter 4.

8.2.2 Optimisation of the Space Vector Sequence for Three-Phase Multilevel Inverters

Another area of further research is optimization of the space vector switching sequences for the three-phase multilevel inverters using the new hysteresis current regulation approach. For the two-level three-phase inverter, this is done by

calculating the third-harmonic common mode offset from the three-phase measured average voltage and compensating in a similar way as was done for the common mode interacting current. From open-loop modulation theory [11], this ensures to achieve the optimized switching sequences similar to the two-level open-loop centered space vector PWM. A similar approach can be used for three-phase multilevel inverters since the regulator measures the three-phase average voltages. However, these voltages need to be further reformulated to convert them from the unipolar form to the bipolar shapes. Now from [15], the common mode voltage can be easily determined and subtracted from the common mode interacting. In this way, the phase disposition switching sequences should be equivalent to the multilevel SVM to achieve the optimum arrangement of the switching sequences.

8.2.3 Hysteresis Current Regulation of Five-Level Three-Phase VSI

The concept of the new constant frequency hysteresis current regulation using the average inverter output voltage has been verified for two-level and three-level inverters. However, there is scope to extend the new approach to at least five-level inverters. The existing five level hysteresis current regulators such as MB and MO or SV based HCC require a minimum of four hysteresis bands per phase leg to detect the correct level of the output voltage level. The new hysteresis current regulator uses only one hysteresis current regulator, because the point of voltage level change is detected from the prediction of the average inverter output voltage zero-crossing. To extend this approach to a five level inverter there are four regions of the average inverter output voltage that need to be identified. These include the average voltage zero-crossing, the two points of 50% modulation depth for the positive and the negative average voltage slope during positive average voltage, and the two points of 50% modulation depth for the positive and the negative average voltage slope during the negative average voltage. These boundary regions along with the necessary combinational digital logic can be used to decode the output of the single hysteresis comparator to generate the appropriate switching signals. Sequential logic may also be necessary to utilize the redundant states depending on the inverter topology. Once the level of the output switched voltage is detected, the previously strategy used for the compensation of the common current from the gate signals, can be employed to extend the new approach to three-phase five-level inverters. Additionally, a five-level variable hysteresis band can be developed using mathematical analysis in a

similar way presented in chapter 3 and chapter 5 with additional band fine tuning to ensure synchronization of the current error zero-crossing. In this way, for the three-phase five-level inverter, the line to line voltage harmonic performance should be similar to a five-level open-loop PD PWM.

8.2.4 Digital Implementation

Another research area is the complete digital implementation of the new hysteresis current regulation approach. In this work, part of the regulator has been implemented using analog circuitry. However, analog implementation of the hysteresis regulator makes it more susceptible to the noise and variation caused by component tolerance while the implementation cost and analog complexity is high. Digital hysteresis current regulation requires a relatively high sampling frequency to get adequate sampling points of the current error for the hysteresis switching process and the proper calculation of the common mode interacting current. But with recent developments in the area of high performance FPGAs, it should be quite feasible to implement the new hysteresis regulator in a complete digital form.

8.2.5 Three-Level Three-Phase Self-Synchronising HCC

The author of this thesis has recently presented a novel three-level self-synchronising HCC[1] for a grid-connected three-level single phase H-bridge. The grid voltage was determined from the average inverter voltage and estimation of the load inductance without requiring extra voltage sensors. A reference current was then generated using the inbuilt DSP SINE lookup table to synchronise to the grid voltage at the desired power factor. Synchronization and operation in this way ensures that the reference current is not affected either by the grid voltage distortion or operation in the over-modulation region. A potential further research area is to extend the proposed strategy to the grid connected three-level three-phase inverters such as NPC and FC VSIs for high power PV applications.

[1]Davoodnezhad, R.; Holmes, D. G.; McGrath, B. P.; "A Three-Level Self-Synchronizing Hysteresis Current Regulator with Constant Switching Frequency" *Energy Conversion Conference (ECCEAsia DownUnder 2013).*

8.3 Closure

Developing hysteresis current regulators has been a topic of substantial research over several decades. The overall aim of any hysteresis current regulator is to achieve a harmonic performance that is comparable to high performance open-loop modulators while maintaining the advantages of the conventional hysteresis current controller. Unfortunately, most of the existing strategies in the literature do not meet this aim and their extension to three-phase multilevel systems requires a complex physical implementation.

This thesis has presented an integrated approach for constant switching frequency hysteresis current regulation of voltage source inverters based on the synchronous measurement of the average inverter output voltage from the switching time information of the switched phase voltage. This approach is independent of the current error derivative and it does not require direct measurement of the current error zero-crossings. For multilevel inverters, the regulator uses only one hysteresis comparator per phase leg to eliminate the DC tracking error. The regulator has been extended to three-phase inverters by compensating for the common mode interacting current, calculated from the hysteresis comparator switching signals. The regulator performance was similar to the high performance two-level and three-level open-loop modulators while retaining all the advantages of conventional HCC.

Appendix A

CPLD Program Code

This appendix contains the CPLD code for the implementation of new hysteresis current regulation approach for two-level and three-level voltage source inverters. The basic firmware of the VHDL code is provided by the Creative Power Technology. The firmware has been modified to include the code for the generation of the PWM signals and their routing depending on the VSI topology. Also, a separate SPI communication algorithm was developed in CPLD to implement a Master/Slave communication protocol using the existing digital I/Os.

The CPLD also generates a programmable dead band between the complementary pair of switching signal and reroutes them to the gate drivers for the final switching process. For three-level inverters, the CPLD implements the combinational digital logic and the finite state machine for the generation of PWM signals generated from the single hysteresis comparator.

This section contains the VHDL code including PWM routing, PWM dead-time generation, SPI communication protocol, special register arrangement for the DSP/CPLD communication, digital I/O assignments and chip selects. Each part of the CPLD code identifies its functionality with necessary comments.

A. 1 Overall Structure of the CPLD VHDL Code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--use IEEE.STD_LOGIC_ARITH.ALL; -- only either one can be used
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity spi_to_bus_v5 is

port(

-- CPLD control signals
-----------------------
CLKOUT          : IN STD_LOGIC;   --Clock into CPLD
RESETn          : IN STD_LOGIC;   --DSP Reset
MCLK            : OUT STD_LOGIC;  --Minibus Clock from CPLD
TEST_1          : INOUT STD_LOGIC;--for debugging
```

```
   -- SPI interface
   ----------------
   SPICLK          : IN STD_LOGIC;   --SPI clock from the 2810
   CPLD_CSn  : IN STD_LOGIC;         --chip select from the 2810
   SPISIMO         : IN STD_LOGIC;   --data from the 2810
   SPISOMI         : OUT STD_LOGIC;  --data to the 2810

   -- Minibus interface
   --------------------
   MINIBUSn  : OUT STD_LOGIC;                    --minibus enable
   RDn       : OUT STD_LOGIC;                    --minibus read
   WRn       : OUT STD_LOGIC;                    --minibus write
   CSn       : OUT STD_LOGIC_VECTOR(2 downto 0);--minibus chip select
   MAx       : OUT STD_LOGIC_VECTOR(2 downto 0);--minibus addresses
   Dx        : INOUT STD_LOGIC_VECTOR(7 downto 0);--minibus bus data

   -- PWM Signals
   --------------
   -- Event Manager A (EVA)
   PWMx      : IN STD_LOGIC_VECTOR (5 downto 0);
   --complementary PWM outputs from DSP
   T1PWM     : IN STD_LOGIC;
   --individual PWM output #7A from DSP
   T2PWM     : IN STD_LOGIC;
   --individual PWM output #8A from DSP
   PDPINTAn  : IN STD_LOGIC;
   --PDPINTA* from external input, Added from SG 25/11/09
   PWMAx     : OUT STD_LOGIC_VECTOR (7 downto 0) := "00000000";

   -- Event Manager B (EVB)
   T3PWM     : IN STD_LOGIC;   -- individual PWM output #7B from DSP
   T4PWM     : IN STD_LOGIC;   -- individual PWM output #8B from DSP
   PDPINTBn  : IN STD_LOGIC;
   --PDPINTB* from external input, Added from SG 25/11/09
   PWMB7     : OUT STD_LOGIC:= '0';  -- PWM output #7B to buffer
   PWMB8     : OUT STD_LOGIC:= '0';  -- PWM output #8B to buffer

   -- Interrupt inputs
   -------------------
   XINT1A          : IN STD_LOGIC;         -- XINT1A from X10
   XINT1B          : IN STD_LOGIC;         -- XINT1B from minibus
   XINT1           : OUT STD_LOGIC;        -- XINT1 to dsp

   -- Digital I/O bits
   -------------------
   DIGINx          : IN STD_LOGIC_VECTOR (3 downto 0);
   -- (16,17,18,19) DIGIN5/6/7/8  from X6
   INDEXx          : IN STD_LOGIC_VECTOR (1 downto 0);
   -- (61,62) INDEXB and INDEXA   from X10
   CAPx            : OUT STD_LOGIC_VECTOR (5 downto 0);
   -- (82,81,78,97,96,95) capture port inputs to DSP
   EN_TTLn         : OUT STD_LOGIC;  -- SCIB X11/X12 header select
   H_ENABn         : OUT STD_LOGIC;  -- enable X4/X5 hysteresis PWMAx
inputs

   INx             : OUT STD_LOGIC_VECTOR (6 downto 0);
   -- (75,74,73,70,69,71,72) IN1 IN2 IN3 INA INB INC IND
   --  analog hysteresis select bits

   GPIOx           : INOUT STD_LOGIC_VECTOR (1 downto 0);
   -- (54,55) GPIOD1, GPIOF7, generic digital IO (outputs)
```

```
  -- Special signal
  -----------------
  SPECIAL         : OUT STD_LOGIC;
  SPI_switch_en   : OUT STD_LOGIC;
  end spi_to_bus_v5;


  architecture behaviour of spi_to_bus_v5 is

  -- automatically set during compilation by update_date.tcl
  ----------------------------------------------------------
  constant DATE_VECTOR_C : std_logic_vector(15 downto
0):="0001100010000101";

  -- signals defined here
  -----------------------

  -- SPI data read from 2810 DSP
  ------------------------------
  -- [8 bits command][8 bits write data][8 bits read data (don't
care)]
  signal SPISIMO_reg          : STD_LOGIC_VECTOR (23 downto 0);
  -- pointer to present SPI register bit
  signal spi_count            : STD_LOGIC_VECTOR (4 downto 0);

  -- Internal CPLD registers
  --------------------------
  signal SCIBMODE_reg   : STD_LOGIC_VECTOR (7 downto 0)
  := "00000000";
  signal CAPQEP_reg     : STD_LOGIC
  :='0';
  signal INTSEL_reg     : STD_LOGIC_VECTOR (7 downto 0)
  := "00000000";
  signal EVACOMCON_reg  : STD_LOGIC_VECTOR (7 downto 0)
  := "00000000";
  signal EVACONDB_reg   : STD_LOGIC_VECTOR (7 downto 0)
  := "00000000";
  signal EVBCOMCON_reg  : STD_LOGIC_VECTOR (7 downto 0)
  := "00000000";
  signal EVBCONDB_reg   : STD_LOGIC_VECTOR (7 downto 0)
  := "00000000";
  signal ANLGSW_reg     : STD_LOGIC_VECTOR (7 downto 0)
  := "00000000";
  signal GPIOx_reg      : STD_LOGIC_VECTOR (7 downto 0)
  := "11110000";

  -- Internal CPLD registers
  --------------------------
  signal PWMAx_reg      : STD_LOGIC_VECTOR (7 downto 0)
  := "00000000";
  signal PWMB7_reg      : STD_LOGIC;
  signal PWMB8_reg      : STD_LOGIC;

  signal PWMA7_DB_reg   : STD_LOGIC;
  signal PWMA8_DB_reg   : STD_LOGIC;
  signal PWMB7_DB_reg   : STD_LOGIC;
  signal PWMB8_DB_reg   : STD_LOGIC;

  signal T1PWM_count    : UNSIGNED (8 downto 0);
  signal T3PWM_count    : UNSIGNED (8 downto 0);
  signal T1PWM_sync     : STD_LOGIC;
```

```
signal T3PWM_sync     : STD_LOGIC;

signal SPECIAL_reg    : STD_LOGIC_VECTOR (7 downto 0)
:= "00000000";

--Hysteresis CPLD registers
signal HSPWMDB_reg    : STD_LOGIC_VECTOR (7 downto 0)
:= "00000000";
signal HSPWM_count    : UNSIGNED (8 downto 0);
signal PWM1_sync      : STD_LOGIC;
signal PWM2_sync      : STD_LOGIC;
signal PWM3_sync      : STD_LOGIC;
signal PWM4_sync      : STD_LOGIC;
signal HSPWMA1_DB_reg : STD_LOGIC;
signal HSPWMA2_DB_reg : STD_LOGIC;
signal HSPWMA3_DB_reg : STD_LOGIC;
signal HSPWMA4_DB_reg : STD_LOGIC;
signal HSPWMA5_DB_reg : STD_LOGIC;
signal HSPWMA6_DB_reg : STD_LOGIC;
signal HSPWMA7_DB_reg : STD_LOGIC;
signal HSPWMA8_DB_reg : STD_LOGIC;
-- logic begins here
--------------------
begin

-- asynchronous declarations of combination logic
-------------------------------------------------

-- special function signal
------------------------
SPECIAL <= '1' when SPECIAL_reg /= "00000000"
         else '0';
SPI_switch_en <= SCIBMODE_reg(7);

-- general i/o signals
----------------------
TEST_1    <= GPIOx_reg(2) when GPIOx_reg(6) = '1'
             else 'Z';
GPIOx(1)  <= GPIOx_reg(1) when GPIOx_reg(5) = '1'
             else 'Z';
GPIOx(0)  <= GPIOx_reg(0) when GPIOx_reg(4) = '1'
             else 'Z';
MCLK                <= CLKOUT;

-- hardware source selection signals
------------------------------------
EN_TTLn   <= SCIBMODE_reg(0);
H_ENABn   <= not ANLGSW_reg(7);

-- Hysteresis Source/Band Selection
-----------------------------------
INx(6)          <= ANLGSW_reg(4);
INx(5)          <= ANLGSW_reg(5);
INx(4)          <= ANLGSW_reg(6);
INx(3)          <= ANLGSW_reg(0);
INx(2)          <= ANLGSW_reg(1);
INx(1)          <= ANLGSW_reg(2);
INx(0)          <= ANLGSW_reg(3);

-- XINT1 pass-thru muxed source
-------------------------------
```

```
XINT1 <= XINT1B when INTSEL_reg(0) = '0' and INTSEL_reg(4) = '1'
    else XINT1A when INTSEL_reg(0) = '1'
    else  '0';

-- muxed capture/index signals
------------------------------
CAPx(0) <= DIGINx(0) when CAPQEP_reg = '1'
         else  DIGINx(2);
CAPx(1) <= DIGINx(1) when CAPQEP_reg = '1'
         else  DIGINx(3);
CAPx(2) <= INDEXx(0) when CAPQEP_reg = '1'
         else  INDEXx(1);
CAPx(3) <= DIGINx(2) when CAPQEP_reg = '1'
         else  DIGINx(0);
CAPx(4) <= DIGINx(3) when CAPQEP_reg = '1'
         else  DIGINx(1);
CAPx(5) <= INDEXx(1) when CAPQEP_reg = '1'
         else  INDEXx(0);

-- minibus read/write/control signals
-------------------------------------
MAx <= SPISIMO_reg(19 downto 17);

WRn <= SPISIMO_reg(16)
when (spi_count > "00000") and (spi_count <= "00101")
else '1';

RDn <= not SPISIMO_reg(16)
when (spi_count > "01001") and (spi_count <= "01101")
else '1';

MINIBUSn <= (SPISIMO_reg(23) and SPISIMO_reg(22))
when (spi_count >= "00000") and (spi_count <= "01110")
else '1';

CSn(0) <= (SPISIMO_reg(23) or SPISIMO_reg(22))
when (spi_count >= "00000") and (spi_count <= "01110")
else '1';

CSn(1) <= not (not SPISIMO_reg(23) and SPISIMO_reg(22))
when (spi_count >= "00000") and (spi_count <= "01110")
else '1';

CSn(2) <= not (SPISIMO_reg(23) and not SPISIMO_reg(22))
when (spi_count >= "00000") and (spi_count <= "01110")
else '1';

Dx <= SPISIMO_reg(15 downto 8)
when (SPISIMO_reg(16) = '0')
and (spi_count >= "00000") and (spi_count <= "01110")
else "ZZZZZZZZ";

-- spi data in/out process
--------------------------
spi: process(RESETn,CPLD_CSn,SPICLK,PDPINTAn,PDPINTBn)
variable SPISOMI_reg        : STD_LOGIC_VECTOR (7 downto 0);
-- SPI data read to 2810 DSP
-- SPISOMI_reg is a variable to avoid possible race condition
    because
-- (a) Additional Peripherals and Registers section writes to
    SPISOMI_reg at spi_count = 00111
```

```
-- (b) code responsible for puting data on SPISOMI also starts at
    spi_count = 00111
--
-- Solution: make SPISOMI a variable and have part (a) placed
    before (b)
-- to ensure that the right data is sent out.

    begin

-- Reset all internal data registers on RESET input (RESETn = '0')
-------------------------------------------------------------------
if(RESETn = '0') then

-- HighZ data readback
SPISOMI <= 'Z';

-- Reset SPI counter (also resets all minibus data/command
    signals)
spi_count <= "10111";       -- set to 23

-- Reset all internal command registers (also resets PWM outputs)
SPISIMO_reg                 <= (others => '0');
SCIBMODE_reg          <= "00000000";
CAPQEP_reg                  <= '0';
INTSEL_reg                  <= "00000000";
EVACOMCON_reg         <= "00000000";
EVACONDB_reg          <= "00000000";
EVBCOMCON_reg         <= "00000000";
EVBCONDB_reg    <= "00000000";
ANLGSW_reg                  <= "00000000";
GPIOx_reg                   <= "11110000";
SPECIAL_reg                 <= "00000000";

-- Reset all internal command registers Related to Hysteresis
    Modifications
HSPWMDB_reg           <= "00000000";
-- CPLD not selected  (CPLD_CSn = '1')
---------------------------------------
elsif(CPLD_CSn = '1') then
-- HighZ data readback
SPISOMI <= 'Z';
-- Reset SPI counter (also resets all minibus data/command
    signals)
spi_count <= "10111";       -- set to 23

-- CPLD selected, no reset active (CPLD_CSn = '0')
-------------------------------------------------
else
-- rising edge SPI processing - SPISIMO data from DSP is stable
-----------------------------
if(rising_edge(SPICLK)) then

-- read in the value, MSB first
    SPISIMO_reg(conv_integer(spi_count)) <= SPISIMO;
-- decrement counter (starts at 23)
    spi_count <= spi_count - 1;

-- end SPI rising clock case
end if;
```

```
-- falling edge SPI processing - SPISOMI data to DSP can be
    changed
------------------------------
if(falling_edge(SPICLK)) then

-- Read MINIBUS data when spi_count is (10)
    if(spi_count = "01010")
          then  SPISOMI_reg := Dx;
    end if;


-- Read/Write into Additional Peripherals and Registers
-----------------------------------------------------
-- executes when spi_count is (7) - BYTE0, BYTE1 received, and
    SCS1,SCI0 both HI
    if(spi_count = "00111"
    and SPISIMO_reg(23) = '1'and SPISIMO_reg(22) = '1')
    then
-- check the address of the register to access
          case(SPISIMO_reg(21 downto 17)) is

-- Register: SCIBMODE (Memory Address: 0xC2)
-------------------------------------------
when "00001" =>
    if (SPISIMO_reg(16) = '0') then
          SCIBMODE_reg <= SPISIMO_reg(15 downto 8);  -- write reg
    else
          SPISOMI_reg := SCIBMODE_reg; -- read reg
    end if;

-- Register: CAPQEP         (Memory Address: 0xC4)
-------------------------------------------
when "00010" =>
    if (SPISIMO_reg(16) = '0') then
          CAPQEP_reg <= SPISIMO_reg(8);  -- write reg
    else
          SPISOMI_reg := "0000000" & CAPQEP_reg; -- read reg
    end if;

-- Register: INTSEL         (Memory Address: 0xC6)
-------------------------------------------
when "00011" =>
    if (SPISIMO_reg(16) = '0') then
          INTSEL_reg <= SPISIMO_reg(15 downto 8);  -- write reg
    else
          SPISOMI_reg := INTSEL_reg; -- read reg
    end if;

-- Register: EVACOMCON            (Memory Address: 0xD0)
-----------------------------------------------
when "01000" =>
    if (SPISIMO_reg(16) = '0') then
          EVACOMCON_reg <= SPISIMO_reg(15 downto 8);-- write reg
    else
          SPISOMI_reg := EVACOMCON_reg; -- read reg
    end if;

-- Register: EVACONDB       (Memory Address: 0xD2)
-------------------------------------------
when "01001" =>
    if (SPISIMO_reg(16) = '0') then
```

```
            EVACONDB_reg <= SPISIMO_reg(15 downto 8);  -- write reg
        else
            SPISOMI_reg := EVACONDB_reg; -- read reg
        end if;

    -- Register: EVBCOMCON           (Memory Address: 0xD4)
    ------------------------------------------------
    when "01010" =>
        if (SPISIMO_reg(16) = '0') then
            EVBCOMCON_reg <= SPISIMO_reg(15 downto 8);-- write reg
        else
            SPISOMI_reg := EVBCOMCON_reg; -- read reg
        end if;

    -- Register: EVBCONDB       (Memory Address: 0xD6)
    ------------------------------------------------
    when "01011" =>
        if (SPISIMO_reg(16) = '0') then
            EVBCONDB_reg <= SPISIMO_reg(15 downto 8);  -- write reg
        else
            SPISOMI_reg := EVBCONDB_reg; -- read reg
        end if;

    -- Register: ANLGSW         (Memory Address: 0xD8)
    ------------------------------------------------
    when "01100" =>
        if (SPISIMO_reg(16) = '0') then
            ANLGSW_reg <= SPISIMO_reg(15 downto 8);  -- write reg
    -- MSB [IN3  IN2  IN1  IND  INC  INB  INA] LSB
        else
            SPISOMI_reg := ANLGSW_reg; -- read reg
        end if;

    -- Register: GPIO           (Memory Address: 0xDA)
    ------------------------------------------------
    when "01101" =>
        if (SPISIMO_reg(16) = '0') then  -- write reg
            GPIOx_reg <= SPISIMO_reg(15 downto 8);
        else
            SPISOMI_reg := "0" & GPIOx_reg(6 downto 4)
            & "0" & TEST_1 & GPIOx; -- read reg
        end if;

    -- Register: SPECIAL        (Memory Address: 0xF0)
    ------------------------------------------------
    when "11000" =>
        if (SPISIMO_reg(16) = '0') then  -- write reg
            SPECIAL_reg <= SPISIMO_reg(15 downto 8);
        else
            SPISOMI_reg := SPECIAL_reg; -- read reg
        end if;

    -- Register: DATELO         (Memory Address: 0xFA)
    ------------------------------------------------
    when "11101" =>
        if (SPISIMO_reg(16) = '1') then
            SPISOMI_reg := DATE_VECTOR_C(7 downto 0); -- read reg
        end if;

    -- Register: DATEHI         (Memory Address: 0xFC)
    ------------------------------------------------
```

```
  when "11110" =>
      if (SPISIMO_reg(16) = '1') then
            SPISOMI_reg := DATE_VECTOR_C(15 downto 8); -- read reg
      end if;

  -- Register: VER            (Memory Address: 0xFE)
  -----------------------------------------
  when "11111" =>
      if (SPISIMO_reg(16) = '1') then
            SPISOMI_reg := X"21"; -- read reg
      end if;

  -- Register: HSPWMDB  (Memory Address: 0xDC)
  -----------------------------------------
  when "01110" =>
      if (SPISIMO_reg(16) = '0') then
            HSPWMDB_reg <= SPISIMO_reg(15 downto 8);  -- write reg
      else
            SPISOMI_reg := HSPWMDB_reg; -- read reg
      end if;
  -- Register: HSPWM_STATUS   (Memory Address: 0xDE)
  -----------------------------------------
  when "01111" =>
      if (SPISIMO_reg(16) = '1') then
            SPISOMI_reg := PWMAx_reg; -- read reg
      end if;

  -- default case (should never happen - just return zero data)
  -----------------------------------------------------------
  when others =>
      SPISOMI_reg := "00000000";

  -- end Additional Peripherals and Registers case
      end case;
      end if;

  -- Send SPI data back to the DSP
  --------------------------------
  -- if we're down to 7, time to put data out
      if(spi_count <= "00111") then
            SPISOMI <= SPISOMI_reg(conv_integer(spi_count));
                                                                    --
send out the value, MSB first
      end if;

  -- end SPI falling clock case
  end if;

  -- end reset/chip select case
  end if;

  -- Reset PWMA enable on PDPINTAn input (PDPINTAn = '0')
  -------------------------------------------------------
  if(PDPINTAn = '0') then
  EVACOMCON_reg(0) <= '0';
  end if;

  -- Reset PWMB enable on PDPINTBn input (PDPINTBn = '0')
  -------------------------------------------------------
  if(PDPINTBn = '0') then
  EVBCOMCON_reg(0) <= '0';
```

```
  end if;

  -- end spi process
  end process spi;
  -- PWMAx lockout protection on PDPINTA*
  ---------------------------------------
  PWMAx     <= PWMAx_reg when EVACOMCON_reg(0) = '1'
  else "00000000";

  -- PWMB7/PWMB8 lockout protection on PDPINTB* --------------------
-------------------------------------------------
PWMB7 <= '0' when EVBCOMCON_reg(7) = '1' and EVBCOMCON_reg(0) = '0'
  else PWMB7_reg;
PWMB8 <= '0' when EVBCOMCON_reg(7) = '1' and EVBCOMCON_reg(0) = '0'
  else PWMB8_reg;
  -- PWMAx routing
  ----------------
  PWMAx_proc:
process(EVACOMCON_reg,EVACONDB_reg,PWMx,T1PWM,T2PWM,PWMA7_DB_reg,PWM
A8_DB_reg,
  HSPWMA8_DB_reg,HSPWMA7_DB_reg,HSPWMA6_DB_reg,HSPWMA5_DB_reg,HSPWMA
4_DB_reg,HSPWMA3_DB_reg,
  HSPWMA2_DB_reg,HSPWMA1_DB_reg,SPECIAL_reg)
  begin
  -- NOTE: this process MUST only write to PWMAx_reg
  -- Asynchronous process / Combinational logic

  -- Passes T1PWM, T2PWM or complementary T1PWM through to PWMA7-8
  ------------------------------------------------------------------
  -- PWM7A / PWM8A pass through  : (EVACOMCON_reg(7) = '0')
  -- Hysteresis PWM Routing
  if(EVACOMCON_reg(7) = '0') then
  PWMAx_reg(7) <= T2PWM;
  PWMAx_reg(6) <= T1PWM;
  -- HB4-->PWMA7  , ~HB4-->PWMA8
  elsif(EVACOMCON_reg(7) = '1' and SPECIAL_reg = "00001001") then
  PWMAx_reg(7) <=   HSPWMA8_DB_reg;
  PWMAx_reg(6) <=   HSPWMA7_DB_reg;
  -- HB4-->PWMA7  , ~HB4-->PWMA8
  elsif(EVACOMCON_reg(7) = '1' and SPECIAL_reg = "00001010") then
  PWMAx_reg(7) <=   HSPWMA8_DB_reg;
  PWMAx_reg(6) <=   HSPWMA7_DB_reg;
  -- Complementary PWMAx[6]/PWMAx[7] based on T1PWM  :
(EVACOMCON_reg(7) = '1')
  else
  -- if EDBT is LO, then disable deadband generator
  if (EVACONDB_reg(7) = '0') then
      PWMAx_reg(7) <= not T1PWM;
      PWMAx_reg(6) <= T1PWM;
  else
  -- enable deadband generator
      PWMAx_reg(7) <= PWMA8_DB_reg;
      PWMAx_reg(6) <= PWMA7_DB_reg;
  end if;

  end if; -- EVACOMCON_reg(7)

  -- Passes PWM1-6 through to PWMA1-6
  ----------------------------------

  -- PWMA6 sourced from PWM6 Input (pass through)
```

```
if(EVACOMCON_reg(6) = '0') then
PWMAx_reg(5) <= PWMx(5);
-- ~HB3-->PWMA6
elsif(EVACOMCON_reg(6) = '1' and SPECIAL_reg = "00001001") then
PWMAx_reg(5) <= HSPWMA6_DB_reg;
-- ~HB3-->PWMA6
elsif(EVACOMCON_reg(6) = '1' and SPECIAL_reg = "00001010") then
PWMAx_reg(5) <= HSPWMA6_DB_reg;
else
-- PWMA6 sourced from internal computation
PWMAx_reg(5) <= PWMx(5);
end if;

-- PWMA5 sourced from PWM5 Input (pass through)
if(EVACOMCON_reg(5) = '0') then
PWMAx_reg(4) <= PWMx(4);
-- HB3-->PWMA5
elsif(EVACOMCON_reg(5) = '1' and SPECIAL_reg = "00001001") then
PWMAx_reg(4) <= HSPWMA5_DB_reg;
-- HB3-->PWMA5
elsif(EVACOMCON_reg(5) = '1' and SPECIAL_reg = "00001010") then
PWMAx_reg(4) <= HSPWMA5_DB_reg;
else
-- PWMA5 sourced from internal computation
PWMAx_reg(4) <= PWMx(4);
end if;

-- PWMA4 sourced from PWM4 Input (pass through)
if(EVACOMCON_reg(4) = '0') then
PWMAx_reg(3) <= PWMx(3);
-- ~HB2-->PWMA4
elsif(EVACOMCON_reg(3) = '1' and SPECIAL_reg = "00001001") then
PWMAx_reg(3) <= HSPWMA4_DB_reg;
-- ~HB2-->PWMA3
elsif(EVACOMCON_reg(3) = '1' and SPECIAL_reg = "00001010") then
PWMAx_reg(3) <= HSPWMA1_DB_reg;
else
-- PWMA4 sourced from internal computation
PWMAx_reg(3) <= PWMx(3);
end if;

-- PWMA3 sourced from PWM3 Input (pass through)
if(EVACOMCON_reg(3) = '0') then
PWMAx_reg(2) <= PWMx(2);
-- HB2-->PWMA3
elsif(EVACOMCON_reg(4) = '1' and SPECIAL_reg = "00001001") then
PWMAx_reg(2) <= HSPWMA3_DB_reg;
-- modification HB2-->PWMA4
elsif(EVACOMCON_reg(4) = '1' and SPECIAL_reg = "00001010") then
PWMAx_reg(2) <= HSPWMA2_DB_reg;

else
-- PWMA3 sourced from internal computation
PWMAx_reg(2) <= PWMx(2);
end if;

-- PWMA2 sourced from PWM2 Input (pass through)
if(EVACOMCON_reg(2) = '0') then
PWMAx_reg(1) <= PWMx(1);
-- ~HB1-->PWMA2
elsif(EVACOMCON_reg(2) = '1' and SPECIAL_reg = "00001001") then
```

```
   PWMAx_reg(1) <= HSPWMA2_DB_reg;
   -- ~HB1-->PWMA2
   elsif(EVACOMCON_reg(2) = '1' and SPECIAL_reg = "00001010") then
   PWMAx_reg(1) <= HSPWMA2_DB_reg;
   else
   -- PWMA2 sourced from internal computation
   PWMAx_reg(1) <= PWMx(1);
   end if;

   -- PWMA1 sourced from PWM1 Input (pass through)
   if(EVACOMCON_reg(1) = '0') then
   PWMAx_reg(0) <= PWMx(0);
   -- HB1-->PWMA1
   elsif(EVACOMCON_reg(1) = '1' and SPECIAL_reg = "00001001") then
   PWMAx_reg(0) <= HSPWMA1_DB_reg;
   -- HB1-->PWMA1
   elsif(EVACOMCON_reg(1) = '1' and SPECIAL_reg = "00001010") then
   PWMAx_reg(0) <= HSPWMA1_DB_reg;

   else
   -- PWMA1 sourced from internal computation
   PWMAx_reg(0) <= PWMx(0);
   end if;

   end process PWMAx_proc;

   -- PWMBx routing
   ----------------
   PWMBx_proc:
process(EVBCOMCON_reg,EVBCONDB_reg,T3PWM,T4PWM,PWMB7_DB_reg,PWMB8_DB
_reg)
   begin
   -- NOTE: this code MUST only write to PWMB7_reg and PWMB8_reg
   -- Asynchronous process / Combinational logic

   -- Passes T3PWM, T4PWM or complementary T3PWM through to PWMB7-8
   ------------------------------------------------------------------
   -- PWM7B / PWM8B pass through  : (EVBCOMCON_reg(7) = '0')
   if(EVBCOMCON_reg(7) = '0') then
   PWMB7_reg <= T3PWM;
   PWMB8_reg <= T4PWM;

   -- Complementary PWMB7/PWMB8 based on T3PWM  : (EVBCOMCON_reg(7) =
'1')
   else
   -- if EDBT is LO, then disable deadband generator
   if (EVBCONDB_reg(7) = '0') then
       PWMB7_reg <= T3PWM;
       PWMB8_reg <= not T3PWM;
   else
   -- enable deadband generator
       PWMB7_reg <= PWMB7_DB_reg;
       PWMB8_reg <= PWMB8_DB_reg;
   end if;

   end if; -- EVBCOMCON_reg(7)

   end process PWMBx_proc;

   -- Deadband section
```

```
  ------------------------------------------------------------------
-----------------

  -- T1PWM & T3PWM input synchronizer
  DB_sync : process(CLKOUT)
  variable T1_FF : STD_LOGIC;
  variable T3_FF : STD_LOGIC;
  begin
  if(rising_edge(CLKOUT)) then
  T1PWM_sync <= T1_FF;
  T3PWM_sync <= T3_FF;
  T1_FF := T1PWM;
  T3_FF := T3PWM;
  end if;
  end process DB_sync;



  -- T1PWM Deadband
  -----------------
  DB_T1PWM_proc: process(CLKOUT)
  variable counter_en   : STD_LOGIC;
  variable prev_T1PWM   : STD_LOGIC;
  variable count  : UNSIGNED (8 downto 0);
  begin

  if(rising_edge(CLKOUT)) then

  -- if we see a transition, start the clock and enable deadband
  if(prev_T1PWM /= T1PWM_sync) then
      counter_en := '1';
      count := T1PWM_count;
  end if;

  -- if we've reached 0, stop counting and disable deadband
  if(count = "000000000") then
      counter_en := '0';
  -- else decrement counter by one
  else
      count := count - 1;
  end if;

  -- remember T3PWM for next time
  prev_T1PWM := T1PWM_sync;
  -- and output delayed PWM outputs
  PWMA7_DB_reg    <= T1PWM_sync and not counter_en;
  PWMA8_DB_reg    <= not T1PWM_sync and not counter_en;
  end if; -- if rising_edge(CLKOUT)

  end process DB_T1PWM_proc;

  -- T3PWM Deadband
  -----------------
  DB_T3PWM_proc: process(CLKOUT)
  variable counter_en   : STD_LOGIC;
  variable prev_T3PWM   : STD_LOGIC;
  variable count  : UNSIGNED (8 downto 0);
  begin

  if(rising_edge(CLKOUT)) then
```

```
   -- if we see a transition, start the clock and enable deadband
   if(prev_T3PWM /= T3PWM_sync) then
       counter_en := '1';
       count := T3PWM_count;
   end if;

   -- if we've reached 0, stop counting and disable deadband
   if(count = "000000000") then
       counter_en := '0';
   -- else decrement counter by one
   else
       count := count - 1;
   end if;

   -- remember T3PWM for next time
   prev_T3PWM := T3PWM_sync;
   -- and output delayed PWM outputs
   PWMB7_DB_reg    <= T3PWM_sync and not counter_en;
   PWMB8_DB_reg    <= not T3PWM_sync and not counter_en;
   end if; -- if rising_edge(CLKOUT)

   end process DB_T3PWM_proc;

   DB_T1PWM_period : process(EVACONDB_reg)
   variable multiplier : UNSIGNED (2 downto 0);
   begin
   case EVACONDB_reg(2 downto 0) is
   when "110" => multiplier := "101"; -- /32 divisor
   when "111" => multiplier := "101";      -- /32 divisor
   when others => multiplier := UNSIGNED(EVACONDB_reg(2 downto 0));
       -- /actual divisor
   end case;

   T1PWM_count <= ("00000" & UNSIGNED(EVACONDB_reg(6 downto 3))) sll
TO_INTEGER(multiplier);
   end process DB_T1PWM_period;

   DB_T3PWM_period : process(EVBCONDB_reg)
   variable multiplier : UNSIGNED (2 downto 0);
   begin
   case EVBCONDB_reg(2 downto 0) is
   when "110" => multiplier := "101"; -- /32 divisor
   when "111" => multiplier := "101";      -- /32 divisor
   when others => multiplier := UNSIGNED(EVBCONDB_reg(2 downto 0));
       -- /actual divisor
   end case;

   T3PWM_count <= ("00000" & UNSIGNED(EVBCONDB_reg(6 downto 3))) sll
TO_INTEGER(multiplier);
   end process DB_T3PWM_period;

   -- HSPWM Deadband for PWM1
   -----------------
   DB_HSPWM1_proc: process(CLKOUT)
   variable counter_en   : STD_LOGIC;
   variable prev_PWM1    : STD_LOGIC;
   variable count  : UNSIGNED (8 downto 0);
   begin

   if(rising_edge(CLKOUT)) then
```

```
-- if we see a transition, start the clock and enable deadband
if(prev_PWM1 /= PWM1_sync) then
    counter_en := '1';
    count := HSPWM_count;
end if;

-- if we've reached 0, stop counting and disable deadband
if(count = "000000000") then
    counter_en := '0';
-- else decrement counter by one
else
    count := count - 1;
end if;

-- remember T3PWM for next time
prev_PWM1 := PWM1_sync;
-- and output delayed PWM outputs
HSPWMA1_DB_reg  <= PWM1_sync and not counter_en;
HSPWMA2_DB_reg  <= not PWM1_sync and not counter_en;
end if; -- if rising_edge(CLKOUT)

end process DB_HSPWM1_proc;

-- HSPWM Deadband for PWM2
-----------------
DB_HSPWM2_proc: process(CLKOUT)
variable counter_en   : STD_LOGIC;
variable prev_PWM2    : STD_LOGIC;
variable count  : UNSIGNED (8 downto 0);
begin

if(rising_edge(CLKOUT)) then

-- if we see a transition, start the clock and enable deadband
if(prev_PWM2 /= PWM2_sync) then
    counter_en := '1';
    count := HSPWM_count;
end if;

-- if we've reached 0, stop counting and disable deadband
if(count = "000000000") then
    counter_en := '0';
-- else decrement counter by one
else
    count := count - 1;
end if;

-- remember T3PWM for next time
prev_PWM2 := PWM2_sync;
-- and output delayed PWM outputs
HSPWMA3_DB_reg  <= PWM2_sync and not counter_en;
HSPWMA4_DB_reg  <= not PWM2_sync and not counter_en;
end if; -- if rising_edge(CLKOUT)

end process DB_HSPWM2_proc;

-- HSPWM Deadband for PWM3
-----------------
DB_HSPWM3_proc: process(CLKOUT)
variable counter_en   : STD_LOGIC;
variable prev_PWM3    : STD_LOGIC;
```

```
variable count  : UNSIGNED (8 downto 0);
begin

if(rising_edge(CLKOUT)) then

-- if we see a transition, start the clock and enable deadband
if(prev_PWM3 /= PWM3_sync) then
    counter_en := '1';
    count := HSPWM_count;
end if;

-- if we've reached 0, stop counting and disable deadband
if(count = "000000000") then
    counter_en := '0';
-- else decrement counter by one
else
    count := count - 1;
end if;

-- remember T3PWM for next time
prev_PWM3 := PWM3_sync;
-- and output delayed PWM outputs
HSPWMA5_DB_reg  <= PWM3_sync and not counter_en;
HSPWMA6_DB_reg  <= not PWM3_sync and not counter_en;
end if; -- if rising_edge(CLKOUT)

end process DB_HSPWM3_proc;

-- HSPWM Deadband for PWM4
-----------------
DB_HSPWM4_proc: process(CLKOUT)
variable counter_en   : STD_LOGIC;
variable prev_PWM4    : STD_LOGIC;
variable count  : UNSIGNED (8 downto 0);
begin

if(rising_edge(CLKOUT)) then

-- if we see a transition, start the clock and enable deadband
if(prev_PWM4 /= PWM4_sync) then
    counter_en := '1';
    count := HSPWM_count;
end if;

-- if we've reached 0, stop counting and disable deadband
if(count = "000000000") then
    counter_en := '0';
-- else decrement counter by one
else
    count := count - 1;
end if;

-- remember T3PWM for next time
prev_PWM4 := PWM4_sync;
-- and output delayed PWM outputs
HSPWMA7_DB_reg  <= PWM4_sync and not counter_en;
HSPWMA8_DB_reg  <= not PWM4_sync and not counter_en;
end if; -- if rising_edge(CLKOUT)

end process DB_HSPWM4_proc;
```

```
  DB_HSPWM_period : process(HSPWMDB_reg)
  variable multiplier : UNSIGNED (2 downto 0);
  begin
  case HSPWMDB_reg(2 downto 0) is
  when "110" => multiplier := "101"; -- /32 divisor
  when "111" => multiplier := "101";      -- /32 divisor
  when others => multiplier := UNSIGNED(HSPWMDB_reg(2 downto 0)); --
/actual divisor
  end case;

  HSPWM_count <= ("00000" & UNSIGNED(HSPWMDB_reg(6 downto 3))) sll
TO_INTEGER(multiplier);

  end process DB_HSPWM_period;

  -- Hysteresis PWM Synchronizer
  HSDB_sync : process(CLKOUT)
  variable PWM1_FF : STD_LOGIC;
  variable PWM2_FF : STD_LOGIC;
  variable PWM3_FF : STD_LOGIC;
  variable PWM4_FF : STD_LOGIC;
  begin
  if(rising_edge(CLKOUT)) then

  PWM1_sync <= PWM1_FF;
  PWM1_FF := PWMx(0);

  PWM2_sync <= PWM2_FF;
  PWM2_FF := PWMx(1);

  PWM3_sync <= PWM3_FF;
  PWM3_FF := PWMx(2);

  PWM4_sync <= PWM4_FF;
  PWM4_FF := PWMx(3);

  end if;
  end process HSDB_sync;
  end architecture behaviour
```

A. 2 VHDL Code for the FC Finite State Machine

```
  library IEEE;
  use IEEE.STD_LOGIC_1164.ALL;

  entity Three_Level_FSM is

      port(
              Sa1                 : IN STD_LOGIC  ;
              Sa2                 : IN STD_LOGIC  ;
              CLKOUT              : IN STD_LOGIC;

              PWMA1               : OUT STD_LOGIC;
              PWMA2               : OUT STD_LOGIC

      );

  end Three_Level_FSM;
```

```
  architecture behaviour of Three_Level_FSM is
  begin

      process(CLKOUT)
      variable state_current: STD_LOGIC_VECTOR (2 downto 0) :=
"000";
      begin
      if (rising_edge(CLKOUT)) then
            if(state_current = "000") then
            if(Sa1 = '1' and Sa2 = '1') then
                PWMA1 <= '1' ;
                PWMA2 <= '1' ;
                state_current := "001" ;
            elsif(Sa1 = '0' and Sa2 = '0') then
                PWMA1 <= '0' ;
                PWMA2 <= '0' ;
                state_current := "100" ;
            else
                PWMA1 <= '0' ;
                PWMA2 <= '1' ;
                state_current := "000" ;
            end if ;


        elsif(state_current = "001") then
            if(Sa1 = '0' and Sa2 = '1') then
                PWMA1 <= '1' ;
                PWMA2 <= '0' ;
                state_current := "010" ;
            else
                PWMA1 <= '1' ;
                PWMA2 <= '1' ;
                state_current := "001" ;
            end if ;


        elsif(state_current = "010") then
            if(Sa1 = '1' and Sa2 = '1') then
                PWMA1 <= '1' ;
                PWMA2 <= '1' ;
                state_current := "011" ;
            elsif(Sa1 = '0' and Sa2 = '0') then
                PWMA1 <= '0' ;
                PWMA2 <= '0' ;
                state_current := "110" ;
            else
                PWMA1 <= '1' ;
                PWMA2 <= '0' ;
                state_current := "010" ;
            end if ;


        elsif(state_current = "011") then
            if(Sa1 = '0' and Sa2 = '1') then
                PWMA1 <= '0' ;
                PWMA2 <= '1' ;
                state_current := "000" ;
            else
                PWMA1 <= '1' ;
                PWMA2 <= '1' ;
                state_current := "011" ;
```

```
            end if ;


        elsif(state_current = "100") then
            if(Sa1 = '0' and Sa2 = '1') then
                PWMA1 <= '1' ;
                PWMA2 <= '0' ;
                state_current := "101" ;
            else
                PWMA1 <= '0' ;
                PWMA2 <= '0' ;
                state_current := "100" ;
            end if ;


        elsif(state_current = "101") then
            if(Sa1 = '0' and Sa2 = '0') then
                PWMA1 <= '0' ;
                PWMA2 <= '0' ;
                state_current := "110" ;
            elsif(Sa1 = '1' and Sa2 = '1') then
                PWMA1 <= '1' ;
                PWMA2 <= '1' ;
                state_current := "011" ;
            else
                PWMA1 <= '1' ;
                PWMA2 <= '0' ;
                state_current := "101" ;
            end if ;


        elsif(state_current = "110") then
            if(Sa1 = '0' and Sa2 = '1') then
                PWMA1 <= '0' ;
                PWMA2 <= '1' ;
                state_current := "111" ;
            else
                PWMA1 <= '0' ;
                PWMA2 <= '0' ;
                state_current := "110" ;
            end if ;
          elsif(state_current = "111") then
            if(Sa1 = '0' and Sa2 = '0') then
                PWMA1 <= '0' ;
                PWMA2 <= '0' ;
                state_current := "100" ;
            elsif(Sa1 = '1' and Sa2 = '1') then
                PWMA1 <= '1' ;
                PWMA2 <= '1' ;
                state_current := "001" ;
            else
                PWMA1 <= '0' ;
                PWMA2 <= '1' ;
                state_current := "111" ;
            end if ;
        end if ;
    end if ;
    end process;
end architecture behaviour;
```

A. 3 VHDL Code for SPI Signal Routing

```
  library IEEE;
  use IEEE.STD_LOGIC_1164.ALL;

  entity SPI_switch3 is

      port(
                  en, oe, S   : IN BIT;

                  p_CSn_out          : OUT STD_LOGIC;
                  p_SCLK_out  : OUT STD_LOGIC;
                  p_SIMO_out  : OUT STD_LOGIC;
                  p_SOMI_in          : IN STD_LOGIC;

                  s_CSn_in           :     IN    STD_LOGIC;
                  s_SCLK_in          :     IN    STD_LOGIC;
                  s_SIMO_in          :     IN    STD_LOGIC;
                  s_SOMI_out  :     OUT   STD_LOGIC;

                  s_CSn_out          :     OUT   STD_LOGIC;
                  s_SCLK_out  :     OUT   STD_LOGIC;
                  s_SIMO_out  :     OUT   STD_LOGIC;
                  s_SOMI_in          :     IN    STD_LOGIC;

                  CSn          :     INOUT STD_LOGIC;
                  SCLK  :     INOUT STD_LOGIC;
                  SIMO  :     INOUT STD_LOGIC;
                  SOMI  :     INOUT STD_LOGIC
      );

  end SPI_switch3;


  architecture behaviour of SPI_switch3 is
  begin

      process(en,S,oe,
                              p_SOMI_in,
                              s_CSn_in, s_SCLK_in, s_SIMO_in,
s_SOMI_in,
                              CSn, SCLK, SIMO, SOMI)
      begin

  -- default state where the  physical pins are
  -- connected to the primary SPI entity which is a slave
            p_CSn_out          <= CSn;
            p_SCLK_out  <= SCLK;
            p_SIMO_out  <= SIMO;
            --p_SOMI_in

            --s_CSn_in
            --s_SCLK_in
            --s_SIMO_in
            s_SOMI_out  <= SOMI;

            s_CSn_out          <= '1';
            s_SCLK_out  <= SCLK;
            s_SIMO_out  <= SIMO;
            --s_SOMI_in
```

```
            CSn         <= 'Z';
            SCLK  <= 'Z';
            SIMO  <= 'Z';
            SOMI  <= p_SOMI_in;

   -- if enabled and secondary = '1'
            if (en = '1' and S = '1') then
   -- if secondary entity is in slave mode
                  if (oe = '0') then
                        p_CSn_out   <= '1';
                        s_CSn_out   <= CSn;

                        SOMI  <= s_SOMI_in;

   -- if secondary entity is in master mode
                  else
                        p_CSn_out   <= '1';
                        s_CSn_out   <= '1';

                        CSn         <= s_CSn_in;
                        SCLK  <= s_SCLK_in;
                        SIMO  <= s_SIMO_in;
                        SOMI  <= 'Z';
                  end if;
            end if; -- en = '1' and s = '1'
      end process;

   end architecture behaviour;
```

Appendix B

DSP Program Code

The basic firmware to control the experimental platforms includes the CS-GIIB, CS-IIB, CPT-DA2810 and CPT-MINI2810 was provided by Creative Power Technology. This firmware is written as multiple header and C files for the initialization of the I/O's, inbuilt sine look up table and serial interface to provide appropriate HMI interface and a state machine to control the overall switching process of VSI together with the over-current, over voltage and over temperature of the inverter.

The foreground C codes including the timer ISR and the capture port ISR have been substantially modified to implement the proposed hysteresis current regulator for two-level and multilevel inverters.

For to two-level VSI, the C code allows the platform to implement the conventional HCC, new single phase and the three-phase HCC via the keyboard using HMI. Detailed functionality of DSP C code has already been discussed in section 7.2.5.2.

For multilevel inverter the C code allow the platform to operate either as a master or slave depending on local configuration of DIP switches. The primary difference between the two-level and multilevel C codes are the calculation of the variable hysteresis bands and also the addition of the C code to detect the point of inverter output voltage level change. Detailed functionality of DSP C code has already been discussed in section 7.2.5.2.

In this appendix only the background codes and the foreground interrupt codes are listed since listing them all would not add to further understanding of the experimental software.

B. 1 Background C code for Two-level Variable Band HCC

```
/*****************************************************************
main.c
*****************************************************************/
// compiler standard include files
#include <stdlib.h>
#include <stdio.h>

// processor standard include files
#include <DSP281x_Device.h>
#include <DSP281x_Examples.h>


// board standard include files
//#include <lib_da2810.h>
#include <lib_mini2810.h>
#include <lib_cpld.h>
#include <iib.h>

// common project include files
//#define AD5624
#define DAC_SHIFT 4
#include <dac_ad56.h>

// common project include files

// local include files
#include "main.h"
#include "conio.h"
#include "vsi.h"
#include "curreg.h"
#include "cas.h"




/*===============================================================
__Definitions()
================================================================ */
#define LCD_CTRL                  (ADD_MINICS2_BASE+MINIBUS_MA1)
#define TPB2                      BIT2
#define SET_TPB2()                GpioDataRegs.GPBSET.all = TPB2
#define CLEAR_TPB2()        GpioDataRegs.GPBCLEAR.all = TPB2

/*===============================================================
__Typedefs()
================================================================ */

/// Time related flag type
/** This structure holds flags used in background timing. */
typedef struct
{
    Uint16
          usec100:1,    ///<1/10 millisecond
          msec:1,           ///< millisecond flag
          msec10:1,   ///< 10ms flag
```

```
            sec0_1:1,   ///< tenth of a second flag
            sec:1,            ///< second flag
            sec5:1;
   } type_time_flag;


   /*=================================================================
   __Variables()
   ================================================================= */

   #ifndef BUILD_RAM
   // These are defined by the linker (see F2812.cmd)
   extern Uint16 RamfuncsLoadStart;
   extern Uint16 RamfuncsLoadEnd;
   extern Uint16 RamfuncsRunStart;
   #endif

   // Background variables
   Uint16
       quit = 0;                     ///< exit flag

   /// timing variable
   type_time_flag
       time =
       {
             0,0,0,0 // flags
       };

   Uint32
       idle_count = 0,                     ///< count of idle time in
the background
       idle_count_old = 0,                 ///< previous count of
idle time
       idle_diff = 0;                      ///< change in idle time
btwn low speed tasks

   char
       str[40];                      // string for displays

   extern signed int
       ZX_time;


   extern int16
       loop_back_character;

   //testing variable for CPLD

   extern int16 master_slave_mode;

   extern int16 sync_method;
   extern int16 NO_SPI_CHAR;

   int16 Unit_number = 0;

   extern Uint16 slave_delay;
   extern int16 Vdc ;
   long k = 0;
   /*
=====================================================================
   __Serial_input_variable()
```

```
  ================================================================*/
  int mod_depth_serial =0;    //In 2810 modulation depth go from 0
to 1000 (0-100%)
  int step_mod_depth_serial = 20;
  int final_mod_depth_serial = 0;
  int mod_depth_max = MOD_DEPTH_MAX;

  double mod_f_freq_serial = INIT_FF; //Fund. modulation freq. in Hz
  double step_f_freq_serial = 0.5;
  double mod_f_freq_max = 100.0;

  int sw_freq_serial                    = SW_FREQ;
  int step_sw_freq_serial               = 50;
  int real_sw_freq_serial               =  SW_FREQ;
  int step_real_sw_freq_serial          = 50;
  int real_bus_voltage_serial           =  850 ;
  int real_fixed_band_serial            =  1015;
  int real_fixed_band_serial_B          =  1015;
  int real_fixed_band_serial_C          =  1015;
  int real_step_fixed_band_serial       = 1 ;
  int real_V3rd_serial                  = 0 ;
  int real_V3rd_offset_serial           = 0 ;
  int average_offset_serial             = 0 ;


  double
      real_Kp_serial = KP_INIT,
      real_Tint_serial = TINT_INIT,
      real_Hband_serial = BMAX;

  long
      step_at_phase_serial = 0;

  int
      step_enable_flag = FALSE,
      step_direction = 1;

  Uint16 spi_fail_count_cpld;
  int16 trash_spi_cpld;

  extern int16
      carrier_offset, carrier_allowed_error;

  extern int16
      ctrl_latch;
  extern int16
         Flag_max_finder;
  extern int16    MDepth_max_main ;
  //extern int16
  //  fault ;
  int16 stepping = 0 ;
  int display_help_counter = 0 ;
  int screen_flag = 0 ;
  int help_flag = 0 ;
  extern int16 I_ref_A_flag;
  extern I_ref_A;
  extern int16 I_ref_B_flag;
  extern I_ref_B;
  extern int16 I_ref_C_flag;
  extern I_ref_C;
  extern reference_flag ;
```

```
  extern int16 refMode ;
  /*===================================================================
  __Local_Function_Prototypes()
  ==================================================================== */

  /* 1 second interrupt for display */
  interrupt void isr_cpu_timer0(void);

  /// display operating info
  void com_display(void);

  /* process keyboard input */
  void com_keyboard(void);
  void display_help(void) ;
  void EnablePowerRelay(void);
  void init_dac_mini(void);
  void SetHystBand(double);
  void SetBusVolts(double);
  void CPLD_FAST_WRITE(char,char);
  void select_operating_mode(void);
  Uint16 CPLD_FAST_READ(char);
  /*===================================================================
  __Grab_Variables()
  ==================================================================== */

  #ifdef GRAB_INCLUDE
  //#pragma DATA_SECTION(grab_array, "bss_grab")
  int16
      grab_mode = GRAB_STOPPED,
      grab_index;
  int32
      grab_array[GRAB_LENGTH][GRAB_WIDTH];
  #endif


  /*===================================================================
      /* Main */
  /*
==================================================================== */
  /* Idle time benchmark:
  \li Ram based program with only bios interrupt and an empty main
loop gives an
  idle_diff of 4.69M (4,685,900)
  \li 23/03/09 V1.02 1.23M with no modbus running
  */
  void main(void)
  {
      static int i = 0;
      Uint32 while_counter=0;

  // Disable CPU interrupts
      DINT;
  // Initialise DSP for PCB
      lib_mini2810_init(150/*MHz*/,37500/*kHz*/,150000/*kHz*/,LIB_EV
AENCLK

      |LIB_EVBENCLK|LIB_ADCENCLK|LIB_SCIAENCLK|LIB_SCIBENCLK|LIB_MCB
SPENCLK);
```

```
      InitGpio();
      spi_init(MODE_CPLD);
      //SpiaRegs.SPICCR.bit.SPILBK = 1;                    //Set
SPI on loop back for testing
      cpld_reg_init();
      //giib_init();
      iib_init();
  // 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();

  #ifndef BUILD_RAM
  // 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

      InitAdc();
      InitCpuTimers();

      // Initialise COM port
      bios_init(9600L);

  // Configure CPU-Timer 0 to interrupt every tenth of a second:
  // 150MHz CPU Freq, 1ms Period (in uSeconds)
      ConfigCpuTimer(&CpuTimer0, 150.0/*MHz*/, 1000.0/*us*/);
      StartCpuTimer0();

  // Interrupts that are used in this example are re-mapped to
  // ISR functions found within this file.
      EALLOW;  // This is needed to write to EALLOW protected
register
      PieVectTable.TINT0 = &isr_cpu_timer0;
      EDIS;    // This is needed to disable write to EALLOW
protected registers

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

      //put_str("\n\nCPT-DA2810 Demo VSI\n\n");
```

```
      //Setting up off board SPI mode: WYK 20090729
      spi_set_mode(MODE_CPLD);                  //Use mode setting
for CPLD for SPI to initialize SPI setting

      //EnableMOSFET2();

      #ifdef GRAB_INCLUDE
      GrabInit();
  #endif
      vsi_init();



  // initialise the chip selects on the board and initialise the spi
      DISABLE_CPLD();   // Disable the CPLD interface
      SET_SPI_MASTER(); // Set the system to Master mode
      CLEAR_OC_SPI_EN();// Turn off external SPI access
      DISABLE_DAC1();
      DISABLE_DAC2();

      spi_init(MODE_CPLD);
      init_dac_mini();

      set_compensation_filter();


      put_str("\n\n\tCS-IIB V1.0 --> Three Phase Hysteresis
Controller Board\n");
      while(while_counter < 250000) while_counter++;
      put_str("\n\n\tReza Davoodnezhad\n");
      while(while_counter < 250000) while_counter++;
      put_str("\n\n\tRMIT University - Power & Energy Group\n\n");
      while(while_counter < 250000) while_counter++;

      select_operating_mode();
      while(1)
      {

      com_keyboard(); // process keypresses
          if (time.msec != 0) // millisecond events
            {
                  com_keyboard(); // process keypresses
                  time.msec = 0;
                  vsi_state_machine();
                  //com_keyboard(); // process keypresses
            }
            else if (time.msec10 != 0) // ten millisecond events
            {
                  time.msec10 = 0;
                  com_keyboard(); // process keypresses
            }
            else if (time.sec0_1 != 0) // tenth of second events
            {
                  time.sec0_1 = 0;
                  if(help_flag)display_help() ;
                  if(GrabShowTrigger() && i < GRAB_LENGTH){
                        GrabDisplay(i);
                        i++;
                  }
                  else if(GrabShowTrigger() && i == GRAB_LENGTH){
                        GrabStop();
```

```
                        i = 0;
                  }
            }
            else if (time.sec != 0) // one second events
            {
                  time.sec = 0;
                  idle_diff = idle_count - idle_count_old;
                  idle_count_old = idle_count;
                  if(screen_flag == 0) com_display();
            }
            else if (time.sec5 != 0){//five second events
                  time.sec5 = 0;

                  if(step_enable_flag == TRUE){
                        if(step_direction == 1){
                              step_direction = 0;
                        }
                        else{
                              step_direction = 1;

                        }
                  }
            }
            else // low priority events
            {
                  idle_count++;
            }
      } /* end while quit == 0 */

//  DISABLE_PWM();
      EvaRegs.T1CON.bit.TENABLE = 0;
      EvaRegs.ACTRA.all = 0x0000;
      DINT;
} /* end main */


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

/* * * Display operating information out COM1.* * * * * * * * */

\author A.McIver
\par History:
\li 22/06/05 AM - initial creation5

\param[in] mode Select whether to start a new display option
*/
void com_display(void)
{
      Uint16
            status;
      if(GrabShowTrigger()){
      }
      else{
            putu(Unit_number);
            put_str(" ");
            status = vsi_get_status();
            if (status & VSI_FAULT)
            {
            //    put_char('F');
```

```
            //    putxx(vsi_get_faults());
            }
            else
            {
            //    putxx(status);
            }
            display_ref_mode();
      //    put_str(" DM:");
      //    putu(vsi_get_mod()/100);
      //    put_str("% ");
      //    put_str(" , ");

            put_str(" Iref:");
            putdbl((double)vsi_get_mod()*20.00/32768.00,2);
            put_str(" , ");

      //    put_str("M:");
      //    putu(MDepth_max_main);
      //    put_str("% ");
      //    put_str(" , ");

            put_str(" Vb:");
            putdbl(((double)ADC_VDC_SC)*((double)Vdc),0);
            put_str("V");
            put_str(" , ");

            put_str("FF:");
            putdbl(vsi_get_freq(),2);
            put_str(" , ");

      //    put_str("FQ:");
      //    putdbl(real_sw_freq_serial,2);
      //    put_str(" , ");

            put_str("Ba:");
            putdbl(real_fixed_band_serial,2);
            put_str(" , ");

            put_str("Bb:");
            putdbl(real_fixed_band_serial_B,2);
            put_str(" , ");

            put_str("Bc:");
            putdbl(real_fixed_band_serial_C,2);
            put_str(" , ");

            put_str("IE:");
            putdbl(real_bus_voltage_serial,2);
            put_str(" , ");

            put_str("3G");
            putdbl(real_V3rd_serial,2);
            put_str(" , ");

            put_str("\r");

      }
  } /* end com_display */
```

```
   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
   /* void com_keyboard
   Parameters: none
   Returns: nothing
   Description: Process characters from COM0.
   Notes:
   History:
       22/06/05 AM - initial creation
   \li 27/11/07 PM - added in testing of the digital I/O
   */
   void select_operating_mode(void)
   {

       char c;
       Uint16 valid = 0;
       Uint32 while_counter = 0;

       // Clear anything on buffer
       while(Kbhit())    c = get_char();

       put_str("\n\nSelect Operating Mode\n");
       while(while_counter < 250000) while_counter++;
       put_str("1. Three Phase Open Loop\n");
       while(while_counter > 1) while_counter--;
       put_str("2. TPH_HCC_VB_3CNT (Change Wiring)\n");
       while(while_counter < 250000) while_counter++;
       put_str("3. Single Phase Hysteresis\n");
       while(while_counter > 1) while_counter--;
       put_str("4. TPH_HCC_FB_2CNT\n");
       while(while_counter > 1) while_counter--;
       put_str("5. TPH_HCC_VB_2CNT\n");
       while(while_counter > 1) while_counter--;
       put_str("6. Third_Harmonic\n");

       while(valid!=1)
       {
             if (Kbhit())
             {
                   c = get_char();
                   switch(c)
                   {
                         case '1': valid = 1 ;
                         CPLD_FAST_WRITE(ADD_CAPQEP,0x00);
                         CPLD_FAST_WRITE(ADD_HSPWMDB,0xFA);
                         CPLD_FAST_WRITE(ADD_SPECIAL,0x00);
                         CPLD_FAST_WRITE(ADD_ANLGSW,0x00);
                         CPLD_FAST_WRITE(ADD_EVACOMCON,0x81);
                         CPLD_FAST_WRITE(ADD_EVACONDB,0xFA);
                         set_RefMode(THREE_PHASE_OL);
                         vsi_disable();
                         EALLOW ;
                         GpioMuxRegs.GPADIR.bit.GPIOA0 = 1 ;
                         GpioMuxRegs.GPADIR.bit.GPIOA1 = 1 ;
                         GpioMuxRegs.GPADIR.bit.GPIOA2 = 1 ;
                         GpioMuxRegs.GPADIR.bit.GPIOA3 = 1 ;
                         GpioMuxRegs.GPADIR.bit.GPIOA4 = 1 ;
                         GpioMuxRegs.GPADIR.bit.GPIOA5 = 1 ;
                         GpioMuxRegs.GPADIR.bit.GPIOA6 = 1 ;
                         GpioMuxRegs.GPADIR.bit.GPIOA7 = 1 ;
```

```
//Enabling PWM pins
GpioMuxRegs.GPAMUX.bit.PWM1_GPIOA0 = 1;
GpioMuxRegs.GPAMUX.bit.PWM2_GPIOA1 = 1;
GpioMuxRegs.GPAMUX.bit.PWM3_GPIOA2 = 1;
GpioMuxRegs.GPAMUX.bit.PWM4_GPIOA3 = 1;
GpioMuxRegs.GPAMUX.bit.PWM5_GPIOA4 = 1;
GpioMuxRegs.GPAMUX.bit.PWM6_GPIOA5 = 1;
GpioMuxRegs.GPAMUX.bit.T1PWM_GPIOA6= 1;
EDIS;
break;

case '2': valid = 1 ;
CPLD_FAST_WRITE(ADD_CAPQEP,0x00);
CPLD_FAST_WRITE(ADD_HSPWMDB,0xFA);
CPLD_FAST_WRITE(ADD_SPECIAL,0x09);
CPLD_FAST_WRITE(ADD_ANLGSW,0x80);
CPLD_FAST_WRITE(ADD_EVACOMCON,0xFF);
set_RefMode(TPH_HCC_VB_3CNT);
vsi_disable();
step_enable_flag = FALSE;
EALLOW ;
GpioMuxRegs.GPADIR.bit.GPIOA0 = 0 ;
GpioMuxRegs.GPADIR.bit.GPIOA1 = 0 ;
GpioMuxRegs.GPADIR.bit.GPIOA2 = 0 ;
GpioMuxRegs.GPADIR.bit.GPIOA3 = 0 ;
GpioMuxRegs.GPADIR.bit.GPIOA4 = 1 ;
GpioMuxRegs.GPADIR.bit.GPIOA5 = 1 ;
GpioMuxRegs.GPADIR.bit.GPIOA6 = 0 ;
GpioMuxRegs.GPADIR.bit.GPIOA7 = 0 ;
GpioMuxRegs.GPAMUX.bit.PWM1_GPIOA0 = 0;
GpioMuxRegs.GPAMUX.bit.PWM2_GPIOA1 = 0;
GpioMuxRegs.GPAMUX.bit.PWM3_GPIOA2 = 0;
GpioMuxRegs.GPAMUX.bit.PWM4_GPIOA3 = 1;
GpioMuxRegs.GPAMUX.bit.PWM5_GPIOA4 = 1;
GpioMuxRegs.GPAMUX.bit.PWM6_GPIOA5 = 0;
GpioMuxRegs.GPAMUX.bit.T1PWM_GPIOA6= 0;
EDIS;
break;

case '3': valid = 1 ;
CPLD_FAST_WRITE(ADD_CAPQEP,0x00);
CPLD_FAST_WRITE(ADD_HSPWMDB,0xFA);
CPLD_FAST_WRITE(ADD_SPECIAL,0x0A);
CPLD_FAST_WRITE(ADD_ANLGSW,0x90);
CPLD_FAST_WRITE(ADD_EVACOMCON,0xFF);
set_RefMode(SINGLE_PHASE_HS);
vsi_disable();
step_enable_flag = FALSE;break;

case '4': valid = 1;

CPLD_FAST_WRITE(ADD_CAPQEP,0x00);
CPLD_FAST_WRITE(ADD_HSPWMDB,0xFA);
CPLD_FAST_WRITE(ADD_SPECIAL,0x09);
CPLD_FAST_WRITE(ADD_ANLGSW,0x80);
CPLD_FAST_WRITE(ADD_EVACOMCON,0x9F);
set_RefMode(TPH_HCC_FB_2CNT);
vsi_disable();
step_enable_flag = FALSE;
EALLOW ;
GpioMuxRegs.GPADIR.bit.GPIOA0 = 0 ;
```

```
GpioMuxRegs.GPADIR.bit.GPIOA1 = 0 ;
GpioMuxRegs.GPADIR.bit.GPIOA2 = 0 ;
GpioMuxRegs.GPADIR.bit.GPIOA3 = 0 ;
GpioMuxRegs.GPADIR.bit.GPIOA4 = 1 ;
GpioMuxRegs.GPADIR.bit.GPIOA5 = 1 ;
GpioMuxRegs.GPADIR.bit.GPIOA6 = 0 ;
GpioMuxRegs.GPADIR.bit.GPIOA7 = 0 ;
GpioMuxRegs.GPAMUX.bit.PWM1_GPIOA0 = 0;
GpioMuxRegs.GPAMUX.bit.PWM2_GPIOA1 = 0;
GpioMuxRegs.GPAMUX.bit.PWM3_GPIOA2 = 0;
GpioMuxRegs.GPAMUX.bit.PWM4_GPIOA3 = 0;
GpioMuxRegs.GPAMUX.bit.PWM5_GPIOA4 = 1;
GpioMuxRegs.GPAMUX.bit.PWM6_GPIOA5 = 1;
GpioMuxRegs.GPAMUX.bit.T1PWM_GPIOA6= 0;
EDIS;
break;
case '5': valid = 1;
CPLD_FAST_WRITE(ADD_CAPQEP,0x00);
CPLD_FAST_WRITE(ADD_HSPWMDB,0xFA);
CPLD_FAST_WRITE(ADD_SPECIAL,0x09);
CPLD_FAST_WRITE(ADD_ANLGSW,0x80);
CPLD_FAST_WRITE(ADD_EVACOMCON,0x9F);
set_RefMode(TPH_HCC_VB_2CNT);
vsi_disable();
step_enable_flag = FALSE;
EALLOW ;
GpioMuxRegs.GPADIR.bit.GPIOA0 = 0 ;
GpioMuxRegs.GPADIR.bit.GPIOA1 = 0 ;
GpioMuxRegs.GPADIR.bit.GPIOA2 = 0 ;
GpioMuxRegs.GPADIR.bit.GPIOA3 = 0 ;
GpioMuxRegs.GPADIR.bit.GPIOA4 = 1 ;
GpioMuxRegs.GPADIR.bit.GPIOA5 = 1 ;
GpioMuxRegs.GPADIR.bit.GPIOA6 = 0 ;
GpioMuxRegs.GPADIR.bit.GPIOA7 = 0 ;
GpioMuxRegs.GPAMUX.bit.PWM1_GPIOA0 = 0;
GpioMuxRegs.GPAMUX.bit.PWM2_GPIOA1 = 0;
GpioMuxRegs.GPAMUX.bit.PWM3_GPIOA2 = 0;
GpioMuxRegs.GPAMUX.bit.PWM4_GPIOA3 = 0;
GpioMuxRegs.GPAMUX.bit.PWM5_GPIOA4 = 1;
GpioMuxRegs.GPAMUX.bit.PWM6_GPIOA5 = 1;
GpioMuxRegs.GPAMUX.bit.T1PWM_GPIOA6= 0;
EDIS;
break;

case '6': valid = 1;

CPLD_FAST_WRITE(ADD_CAPQEP,0x00);
CPLD_FAST_WRITE(ADD_HSPWMDB,0xFA);
CPLD_FAST_WRITE(ADD_SPECIAL,0x09);
CPLD_FAST_WRITE(ADD_ANLGSW,0x80);
CPLD_FAST_WRITE(ADD_EVACOMCON,0x9F);
set_RefMode(TRD_HARM);
vsi_disable();
step_enable_flag = FALSE;
EALLOW ;
GpioMuxRegs.GPADIR.bit.GPIOA0 = 0 ;
GpioMuxRegs.GPADIR.bit.GPIOA1 = 0 ;
GpioMuxRegs.GPADIR.bit.GPIOA2 = 0 ;
GpioMuxRegs.GPADIR.bit.GPIOA3 = 0 ;
GpioMuxRegs.GPADIR.bit.GPIOA4 = 1 ;
GpioMuxRegs.GPADIR.bit.GPIOA5 = 1 ;
```

```
                        GpioMuxRegs.GPADIR.bit.GPIOA6 = 0 ;
                        GpioMuxRegs.GPADIR.bit.GPIOA7 = 0 ;
                        GpioMuxRegs.GPAMUX.bit.PWM1_GPIOA0 = 0;
                        GpioMuxRegs.GPAMUX.bit.PWM2_GPIOA1 = 0;
                        GpioMuxRegs.GPAMUX.bit.PWM3_GPIOA2 = 0;
                        GpioMuxRegs.GPAMUX.bit.PWM4_GPIOA3 = 0;
                        GpioMuxRegs.GPAMUX.bit.PWM5_GPIOA4 = 1;
                        GpioMuxRegs.GPAMUX.bit.PWM6_GPIOA5 = 1;
                        GpioMuxRegs.GPAMUX.bit.T1PWM_GPIOA6= 0;
                        EDIS;
                        break;
                  }
            }
      }
}
/**
Displays help menu

\author R.Davoodnezhad

\par History:
\li 05/05/11 RD - Creation

*/
void display_help(void)
{
    switch(display_help_counter)
    {
          /* case 0 never happens */
          case 1: put_str("\n\tHelp Menu\n");
          break;
          case 2: put_str("\t Press 'e' to enable VSI\n");
          break;
          case 3: put_str("\t Press 'd' to disable VSI\n");
          break;
          case 4: put_str("\t Press 'o' to change operation
          mode\n") ;
          break;
          case 5: put_str("\t Press 'f' to set a 50Hz
          fundamental\n");
          break;
          case 6: put_str("\t Press 'i/I' to increase/decrease
          reference current\n");
          break;
          case 7: put_str("\t Press 's/S' to increase/decrease
          Phase A hysteresis band\n");
          break;
          case 8: put_str("\t Press 'z/Z' to increase/decrease
          Phase B hysteresis band\n");
          break;
          case 10:put_str("\t Press 'a/A' to increase/decrease
          Phase C hysteresis band\n\n");screen_flag = 0;
          break;
          default: break;
    }
    if (display_help_counter<11) display_help_counter++;

}

void com_keyboard(void)
{
```

```
    char c;

//  put_str("KEY");
    if (Kbhit())
    {
          c = get_char();
          switch (c)
          {
                case '!':
                      spi_init(MODE_CPLD);
                      put_str("SPECIAL: ");
                      putxx(cpld_read(ADD_SPECIAL));
                      put_str("\n");
                break ;
                case '@':
                      put_str("EVACOMCON: ");
                      putxx(cpld_read(ADD_EVACOMCON));
                      put_str("\n");
                break;
                case '#' :
                      put_str("HSPWMDB: ");
                      putxx(cpld_read(ADD_HSPWMDB));
                      put_str("\n");
                break;
                case '$' :
                      put_str("ANLGSW: ");
                      putxx(cpld_read(ADD_ANLGSW));
                      put_str("\n");
                break;

                case '%' :
                      put_str("HSPWM_STATUS: ");
                      putxx(cpld_read(ADD_HSPWM_STATUS));
                      put_str("\n");
                break;

                case 'e':
                      vsi_enable();
                      ctrl_latch |= CTRL_POWER_RELAY;
                      CPLD_FAST_WRITE(LCD_CTRL,ctrl_latch);
                      EnablePowerRelay();
                      put_str("\nENABLED\n\n");
                      put_str("\n");
                      EnableMOSFET2();
                break;

                case 'd':
                      vsi_disable();
                      ctrl_latch &= ~CTRL_POWER_RELAY;
                      CPLD_FAST_WRITE(LCD_CTRL, ctrl_latch);
                      DisablePowerRelay();
                      put_str("\nDISABLED\n\n");
                      put_str("\n");
                      DisableMOSFET2();
                break;
                //    Modulation depth set up
                case 'i':
                      Flag_max_finder = 1 ;
```

```
                        if((mod_depth_serial+step_mod_depth_serial)
                        < mod_depth_max){ mod_depth_serial
                        +=step_mod_depth_serial; }
                        else{ mod_depth_serial = mod_depth_max;   }
                        vsi_set_mod(mod_depth_serial);
                  break;
                  case 'I':
                        Flag_max_finder = 1 ;
                        if((mod_depth_serial-step_mod_depth_serial)
                        > 0){ mod_depth_serial -
                        =step_mod_depth_serial; }
                        else{ mod_depth_serial = 0;   }
                        vsi_set_mod(mod_depth_serial);
                  break;

                  // Set modulation step
                  case 'u':

                  if((final_mod_depth_serial+step_mod_depth_serial)
                  < mod_depth_max){ final_mod_depth_serial
                  +=step_mod_depth_serial;      }
                  else{ final_mod_depth_serial = mod_depth_max;   }
                  break;
                  case 'U':
                  if((final_mod_depth_serial-step_mod_depth_serial)
                  > 0){ final_mod_depth_serial-
                  =step_mod_depth_serial; }
                  else{ final_mod_depth_serial = 0;   }
                  break;

                  //Fundamental frequency of modulation signal
                  case 'f':
                        if((mod_f_freq_serial+step_f_freq_serial) <
                        mod_f_freq_max){ mod_f_freq_serial
                        +=step_f_freq_serial;}
                        else{ mod_f_freq_serial = mod_f_freq_max; }
                        vsi_set_freq(mod_f_freq_serial);
                        vsi_set_freq(50.00);
                  break;
                  case 'F':
                        if((mod_f_freq_serial-step_f_freq_serial)
                        >0){ mod_f_freq_serial -
                        =step_f_freq_serial;}
                        else{ mod_f_freq_serial = 0;}
                        vsi_set_freq(mod_f_freq_serial);
                  break;

                  case 'b':
                        if (real_bus_voltage_serial <
                        MAX_FIXED_BAND)
real_bus_voltage_serial+=real_step_fixed_band_serial;
SetBusVolts(real_bus_voltage_serial);
                  break;

                  case 'B':
                        if (real_bus_voltage_serial >
MIN_FIXED_BAND) real_bus_voltage_serial-
=real_step_fixed_band_serial; SetBusVolts(real_bus_voltage_serial);
                  break;

                  case 's':
```

```
                        if (real_sw_freq_serial < MAX_SWT_FREQ)
real_sw_freq_serial+=step_real_sw_freq_serial;
SetHystBand(real_sw_freq_serial);
                        if (real_fixed_band_serial < MAX_FIXED_BAND)
real_fixed_band_serial+=real_step_fixed_band_serial;
SetHystBand(real_fixed_band_serial);
                  break;

                  case 'S':
                        if (real_sw_freq_serial > MIN_SWT_FREQ)
real_sw_freq_serial-=step_real_sw_freq_serial;
SetHystBand(real_sw_freq_serial);
                        if (real_fixed_band_serial > MIN_FIXED_BAND)
real_fixed_band_serial-=real_step_fixed_band_serial;
SetHystBand(real_fixed_band_serial);
                  break;
                  case 'z':
                        if (real_sw_freq_serial < MAX_SWT_FREQ)
real_sw_freq_serial+=step_real_sw_freq_serial;
SetHystBand(real_sw_freq_serial);
                        if (real_fixed_band_serial_B <
MAX_FIXED_BAND)
real_fixed_band_serial_B+=real_step_fixed_band_serial;
SetHystBand(real_fixed_band_serial_B);
                        if (real_fixed_band_serial_B <
MAX_FIXED_BAND) real_fixed_band_serial_B+=1L;
SetHystBand(real_fixed_band_serial_B);
                  break;

                  case 'Z':
                        if (real_sw_freq_serial > MIN_SWT_FREQ)
real_sw_freq_serial-=step_real_sw_freq_serial;
SetHystBand(real_sw_freq_serial);
                        if (real_fixed_band_serial_B >
MIN_FIXED_BAND) real_fixed_band_serial_B-
=real_step_fixed_band_serial; SetHystBand(real_fixed_band_serial_B);
                        if (real_fixed_band_serial_B > -5000)
real_fixed_band_serial_B-=1; SetHystBand(real_fixed_band_serial_B);
                  break;
                  case 'a':
                        if (real_sw_freq_serial < MAX_SWT_FREQ)
real_sw_freq_serial+=step_real_sw_freq_serial;
SetHystBand(real_sw_freq_serial);
                        if (real_fixed_band_serial_C <
MAX_FIXED_BAND) real_fixed_band_serial_C+=1;
SetHystBand(real_fixed_band_serial_C);
                  break;

                  case 'A':
                        if (real_sw_freq_serial > MIN_SWT_FREQ)
real_sw_freq_serial-=step_real_sw_freq_serial;
SetHystBand(real_sw_freq_serial);
                        if (real_fixed_band_serial_C >
MIN_FIXED_BAND) real_fixed_band_serial_C-=1;
SetHystBand(real_fixed_band_serial_C);
                  break;

                  case 'b': if (real_Hband_serial < MAX_HYST_BAND-
0.01) real_Hband_serial+=0.01; SetHystBand(real_Hband_serial);
                  break;
```

```
                case 'B': if (real_Hband_serial >
MIN_HYST_BAND+0.01) real_Hband_serial-=0.01;
SetHystBand(real_Hband_serial); break;
                case 'h':
                            display_help_counter = 0 ;
                            screen_flag = 1 ;
                            help_flag = 1 ;
                            break;

                case '+':
                            stepping = 1 ;
                break;
                case '-':
                            stepping = 0 ;
                break;
                case 'o':
                            select_operating_mode() ;
                break;

  #ifdef GRAB_INCLUDE
                case 'g': /* grab interrupt data */
                      GrabClear();
                      GrabStart();
                      GrabRun();
                break;
                case 'j':
                      GrabShow();
                break;
                //case 'c': /* stop grab display */
                //    GrabClear();
                //break;
  #endif
                case 'H':
                      if(master_slave_mode == 0){
                            put_str("Slave\n");
                      }
                      else{
                            put_str("Master\n");
                      }
                      break;
          }
    }
  } /* end com_keyboard */


  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  1 second CPU timer interrupt.
  \author A.McIver
  \par History:
  \li 22/06/05 AM - initial creation (derived from k:startup.c)
  */
  #ifndef BUILD_RAM
  #pragma CODE_SECTION(isr_cpu_timer0, "ramfuncs");
  #endif
  interrupt void isr_cpu_timer0(void)
  {
      static struct
      {
            Uint16
                usec100,
                  msec,
```

```
                  msec10,
                  msec100,
                  sec;
      } i_count =
      {
            0, 0, 0, 0
      };

      /*for (ii=0; ii<WD_TIMER_MAX; ii++)
      {
            if (wd_timer[ii] > 0)
                  wd_timer[ii]--;
      }*/

            i_count.msec++;
      if (i_count.msec >= 10)
      {
            i_count.msec = 0;
            i_count.msec10++;
            if (i_count.msec10 >= 10)
            {
                  i_count.msec10 = 0;
                  i_count.msec100++;
                  if (i_count.msec100 >= 10)
                  {
                        i_count.msec100 = 0;
                        i_count.sec++;
                        if(i_count.sec >= 5){
                              time.sec5 = 1;
                        }
                        time.sec = 1;
                  }
                  time.sec0_1 = 1;
            }
            time.msec10 = 1;
      }
      time.msec = 1;

      // Acknowledge this interrupt to receive more interrupts from
group 1
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
  } /* end isr_cpu_timer0 */


  /*=================================================================
  __Grab_Functions()
  ================================================================= */
  #ifdef GRAB_INCLUDE

  void GrabInit(void)
  {
      Uint16
            i,j;

      for (i=0; i<GRAB_LENGTH; i++)
      {
            for (j=0; j<GRAB_WIDTH; j++)
            {
                  grab_array[i][j] = 0;
            }
      }
```

```
    GrabClear();
}

/* call with index == 0xFFFF for title line
else index = 0..GRAB_LENGTH-1 for data */
void GrabDisplay(int16 index)
{
    Uint16
          i;

    if (index == 0xFFFF)
    {
          put_str("\nindex");
          for (i=0; i<GRAB_WIDTH; i++)
          {
                put_str("\t");
                put_d(i);
          }
    }
    else
    {
          put_d(index);
          put_char('\t');
          for (i=0; i<GRAB_WIDTH; i++)
          {
                //put_char('');
                putl(grab_array[index][i]);
                put_char('\t');
          }
    }
    put_str("\n");
}

#endif
/* * * * * * * * * * * * * * * * * * * * * * * ** * * * * * * * * * */

void print_help(void)
{
/*  put_str("\th\thelp\n");
    put_str("\tq\tquit\n");
    put_str("\te/d\tenable/disable switching\n");
    put_str("\ti/m\treference magnitude increase/decrease\n");
    put_str("\tl/j\tref freq increase/decrease\n");
    put_str("\t>/<\tswitching freq increase/decrease\n");
    put_str("\tt/T\tintegral reset time slow/fast increase\n");
    put_str("\tv/V\tintegral reset time slow/fast decrease\n");
    put_str("\tr/R\tproportional constant slow/fast increase\n");
    put_str("\tc/C\tproportional constant slow/fast decrease\n");
    //puts("\tg\tto enable grab code and print grab data\n");
    put_str("\tu/b\tincrease/decrease magnitude of step change in
    reference\n");
    put_str("\ta/s\tincrease/decrease phase where step is
    applied\n");
    put_str("\tp\tenable/disable step change in reference every 5
    sec\n");
    put_str("\tf\tenable/disable feed forward\n");
    put_str("\t0\tDC current regulator (using leg A&B)\n");
    put_str("\t1\tSingle phase PI regulator mode (using leg
    A&B)\n");
    put_str("\t2\tSingle phase PI regulator mode (using leg A&B)
    with back EMF\n");
```

```
      put_str("\t3\tSingle phase PR regulator mode (using leg A&B)
      with back EMF\n");
      put_str("\t4\t3 phase PI regulator mode \n");
      put_str("\t5\t3 phase PI regulator mode with back EMF\n");
      put_str("\t6\t3 phase DQ regulator mode with back EMF\n");
      put_str("\t7\t3 phase PR regulator mode with back EMF\n");
      */
  } /* end print_help */


  /*Reza DAC initialization function */
  void init_dac_mini(void)
  {
      // Set SPI mode
      spi_init(MODE_DAC);
      // Initialise DAC
      dac_init();
      // Set internal reference
      dac_set_ref(DAC_MODULE_D1,DAC_INT_REF);
      dac_set_ref(DAC_MODULE_D2,DAC_INT_REF);
      // Power up DAC
      dac_power_down(DAC_MODULE_D1,0x0F);
      dac_power_down(DAC_MODULE_D2,0x0F);
      // Write to half voltage
      dac_write(DAC_MODULE_D1,DAC_WRn_UPDn,DAC_ADDR_ALL,2047);
  }

  /*================================================================
  CPLD_FAST_WRITE()
  ================================================================*/
  void CPLD_FAST_WRITE(char REGISTER_ADDRESS,char DATA_BYTE)
  {
      GpioDataRegs.GPASET.all = OC_SPI_EN;
      GpioDataRegs.GPASET.all = M_nS;
      GpioDataRegs.GPFCLEAR.all = nCPLD_CS;
      spi_fail_count_cpld = 65535;
      SpiaRegs.SPITXBUF = ((REGISTER_ADDRESS|MINIBUS_WRITE)<<8);
      SpiaRegs.SPITXBUF = ((DATA_BYTE)<<8);
      SpiaRegs.SPITXBUF = ((0x00)&0x00FF)<<8;
      while ((SpiaRegs.SPIFFRX.bit.RXFFST <
3)&&(spi_fail_count_cpld>0))
            {
                  spi_fail_count_cpld--;
            }
            trash_spi_cpld=SpiaRegs.SPIRXBUF;
            trash_spi_cpld=SpiaRegs.SPIRXBUF;
            trash_spi_cpld=SpiaRegs.SPIRXBUF;
            GpioDataRegs.GPFSET.all = nCPLD_CS;
  }

  Uint16 CPLD_FAST_READ(char REGISTER_ADDRESS)
  {
      Uint16
            data_read = 0x00;

      GpioDataRegs.GPASET.all = OC_SPI_EN;
      GpioDataRegs.GPASET.all = M_nS;
      GpioDataRegs.GPFCLEAR.all = nCPLD_CS;

       spi_fail_count_cpld = 65535;
      SpiaRegs.SPITXBUF = ((REGISTER_ADDRESS|MINIBUS_READ)<<8);
```

```
      SpiaRegs.SPITXBUF = ((0x00)<<8);
      SpiaRegs.SPITXBUF = ((0x00)&0x00FF)<<8;
      while ((SpiaRegs.SPIFFRX.bit.RXFFST <
3)&&(spi_fail_count_cpld>0))
            {
                  spi_fail_count_cpld--;
            }
            trash_spi_cpld=SpiaRegs.SPIRXBUF;
            data_read=SpiaRegs.SPIRXBUF;
            data_read=SpiaRegs.SPIRXBUF;
            GpioDataRegs.GPFSET.all = nCPLD_CS;

      GpioDataRegs.GPFSET.all = nCPLD_CS;

      return data_read;
  } /* end cpld_read */
```

B. 2 Foreground C code for Two-level Variable Band HCC

```
   /**
  \file
  \brief VSI Interrupt Service Routine
  This file contains the code for   the   core interrupt routine for
the Hysteresis    Current     Regulator.
  This interrupt is     the   central     system for the signal
      generation and
  measurement. The carrier timer for the VSI generation     also
triggers    the
  internal ADC conversion     at the peak of the carrier.   The
      end   of conversion     then
  triggers this   interrupt. Its tasks are:
  -   Read internal     ADC   results
  -   Perform     internal analog   averaging   and   RMS
      calculations
  -   Update VSI phase and switching time
  -   Read capture port interrupts
  \par Developed By:
      Creative Power Technologies, (C) Copyright 2009
  \author   A.McIver
  \author   R. Davoodnezhad
  \par History:
  \li 23/04/09 AM -     initial     creation
  */
  // compiler     standard include files
  #include <math.h>
  #include <DSP281x_Device.h>
  #include <DSP281x_Examples.h>
  //#include <lib_da2810.h>
  #include <lib_mini2810.h>
  #include <lib_cpld.h>
  #include <iib.h>
  // common project     include     files
  //#define AD5624
  #define   DAC_SHIFT   4
  #include <dac_ad56.h>
  // local include files
  #include "main.h"
  #include "conio.h"
```

```
   #include "vsi.h"
   #include "curreg.h"
   #include "cas.h"
   /*
=========================================================================
=====
   __Definitions()
   ========================================================================
========== */
   /// Shift from internal     modulation depth scaling
   #define   MOD_SHIFT                    14
   /* the phase is scaled so   that one fundamental is 2^32 counts.
*/
   //#define PHASE_STEP_SC_D               (65536.0*65536.0/SW_FREQ)
                        //Synchronous     switching   sampling freq
   #define   PHASE_STEP_SC_D               (65536.0*65536.0/SW_FREQ)
                  //Asynchronous switching sampling
   #define   PHASE_STEP
       (Uint32)(PHASE_STEP_SC_D*F_FREQ_MIN)
   /// ADC   calibration time
   #define   ADC_CAL_TIME                  0.1   // seconds
   #define   ADC_COUNT_CAL                 (Uint16)(ADC_CAL_TIME   *
       SW_FREQ)
   /// DC averaging time
   #define   ADC_DC_TIME                   0.1   // seconds
   #define   ADC_COUNT_DC                  (Uint16)(ADC_DC_TIME *
SW_FREQ)
   #define   ADC_REAL_SC                   1
   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
   /// RMS   scaling
   #define   ADC_RMS_PS             4
   #define   GRAB_INCLUDE
   #define   HIGH 1
   #define   LOW   0
   #define   ON                            1
   #define   OFF                           0
   #define   SYNC_PIN               BIT0
   #define   TPB2                   BIT2
   #define   TPB1                   BIT1
   #define   TPA0                   BIT0
   #define   SET_SYNC_PIN()         GpioDataRegs.GPBSET.all =
       SYNC_PIN
   #define   CLEAR_SYNC_PIN()  GpioDataRegs.GPBCLEAR.all     =
       SYNC_PIN
   #define   SET_TPB1()             GpioDataRegs.GPBSET.all =
       TPB1
   #define   CLEAR_TPB1()           GpioDataRegs.GPBCLEAR.all     =
       TPB1
   #define   SET_TPB2()             GpioDataRegs.GPBSET.all =
       TPB2
   #define   CLEAR_TPB2()           GpioDataRegs.GPBCLEAR.all     =
       TPB2
   /*
=========================================================================
=====
   __Macros()
   ================================================== */
   /// Disable     VSI   switching
   #define   VSI_DISABLE()          EvaRegs.ACTRA.all =     0x0000
   /// Enable VSI switching
```

```
  #define   VSI_ENABLE()            EvaRegs.ACTRA.all =     0x0666
  /// Enable VSI for single   phase operation
  #define   VSI_ENABLE_1P()         EvaRegs.ACTRA.all =     0x0066
  /// Turn low side     devices     on full     for   charge pump
      starting
  #define   VSI_GATE_CHARGE() EvaRegs.ACTRA.all =     0x00CC
  /* ============================
  __Controller_Macros()
  ============================================================ */
  /*****************************************************************
  *   This function perform a PI loop calculation,  the following
parameters needed
  *   to be provided to the   function in pointer     form:
  *   REF   =     reference   current     in signed   int   format
  *   ERR   =     error, this variable should   be in signed int
format, it  will be     wrote to
  *   MEA   =     measured current in     signed int format
  *   KP = proportional constant,   in signed   int   format
  *   KI = integral     constant,   in signed   int   format
  *   PROP_NOW = the proportional   part of     PI loop     for
      this pass   of the loop in long     int
  *   INT_NOW     =     the   integral action   calculated for this
      pass of     the   loop in     long int
  *   INT_TOTAL   =     the   integral action   in total (memory of
      integrator) in long     int
  *   FF = the amount   of feedfoward     to used     in signed
      int
  *   COMM_OUT = output command     in signed   int
  *   Note on     PI code:
  *   Multiple by 1<<PROP_DISCARD_BITS is to compensate     for
      the   division of the   same number
  *   at the calculation for proportional time constant
  *   The   division by 1<<PROP_DISCARD_BITS allow for the use of
      higher proportional     constant
  *   then otherwise possible, there should     not   be significant
loss  to accuracy provided
  *   that the number   for   PROP_DISCARD_BITS is smallish, the
calculation for lost    of accuracy
  *   is listed   in comment above
  *   Contain     desaturation and clamping     code.
  *   256   *     128   give 32767 which is     200% modulation
  *****************************************************************
************/
  //Delta   transform   definitions
  #define   LOG2_ALPHA_0                        11
  #define   ISS
      (LOG2_ALPHA_0-6)
  #define   LOG2_1_ON_DELTA               2
  #define   w_c_f                                         0.1
  #define   PR_INTERNAL_CLAMP       10000
  #define   PR_SCALING
      (int)(65536.0/I_NOM/4.0+0.5)                    //Scaling
      constant to per   unitised the controller output.
Division    of 4 is     there because     of the two left   shift in
the subsequence period  calculation
  #define   LOG2_ALPHA_0_COM        11
  #define   ISS_FILT                                  0
  /*============================================================
======================
  AB2DQ()
```

```
  ================================================================
======================*/
  // Change two   AB frame value into     DQ axis     value.
     Assume balance three phase system
  #define   AB2DQ(A, B, D, Q){ \
     D     =
     ((((((long)A<<1)*(long)(sin_PI_on_3_val))+(((long)B<<1)*(long)
(sin_val)))>>     16)*TWO_ON_ROOT3)>>16; \
     Q     =
     ((((((long)A<<1)*(long)(cos_PI_on_3_val))+(((long)B<<1)*(long)
(cos_val)))>>     16)*TWO_ON_ROOT3)>>16; \
  }
  /*================================================================
======================
  DQ2AB()
  ================================================================
======================*/
  //  Change two DQ     axis quantities   into AB     frame
     equivalent.  Assume     balance     three phase system
  #define   DQ2AB(D, Q, A, B){ \
     A     =     (((((long)D<<1)*(long)(cos_val))-
(((long)Q<<1)*(long)(sin_val)))>>16); \
     B     =
     (((((long)D<<1)*(long)(sin_PI_on_6_val))+(((long)Q<<1)*(long)(
sin_PI_on_3_val)))>>16); \
  }
  /*================================================================
======================
  SIN_TABLE_UPDATE()
  ================================================================
======================*/
  //Update INDEX to     point to the current location of sin table
  #define   SIN_TABLE_UPDATE(PHASE, INDEX){     \
     PHASE += phase_step;     \
     INDEX =     (PHASE>>22)|0x001;      \
  }
  /*================================================================
======================
  SIN_TABLE_READ()
  ================================================================
======================*/
  //Just read     the   value of sin table at   location indicated
by    INDEX
  #define   SIN_TABLE_READ(INDEX,   VAL){
     val_lo = sin_table[INDEX];
     val_diff = sin_table[INDEX+2] -     val_lo;
     VAL   =     (val_lo     +
     (int16)(((int32)(INDEX&0x007F)*(int32)val_diff)>>7))<<1;
  }
  /*================================================================
======================
  DSP_SET_SPI_SLAVE()
  ================================================================
======================*/
  // Setup the SPI core to function as a slave to     recieve
     date (note, this is     just the core, does     not   include
     setup for
  // Mini2810     and   GIIB, so if data needed to be recieve
     from GIIB, more   setup is needed)
  #define   DSP_SET_SPI_SLAVE(){
     GpioDataRegs.GPFCLEAR.all     =     BIT3;
```

```
      SpiaRegs.SPICCR.all     =     0x0000;
      /*Set SPI   to reset state*/
      SpiaRegs.SPICCR.bit.CLKPOLARITY     =     0;
      SpiaRegs.SPICCR.bit.SPICHAR   =     7;    /* 8 bit
characters*/
      SpiaRegs.SPICTL.all     =     0x0000;
      SpiaRegs.SPICTL.bit.CLK_PHASE =     1;
      SpiaRegs.SPICTL.bit.MASTER_SLAVE = 0;     /*Slave*/
      SpiaRegs.SPICTL.bit.TALK = 0;             /* enable
      transmission*/
      SpiaRegs.SPICTL.bit.SPIINTENA =     0;                /*Enable
SPI interrupt*/
      SpiaRegs.SPIBRR   =     4;                      /*
37.5/(SPIBRR+1)MHz      */
      SpiaRegs.SPIFFTX.all = 0x0000;
      SpiaRegs.SPIFFTX.bit.SPIFFENA =     1;    /* enable   tx
fifo*/
      SpiaRegs.SPIFFTX.bit.TXFIFO   =     1;          /* Enable
      Transmit channel*/
      SpiaRegs.SPIFFTX.bit.SPIRST   =     1;          /* enable
      SPI*/
      SpiaRegs.SPIFFRX.all = 0;
      SpiaRegs.SPIFFRX.bit.RXFFIL   =     3;    /* interrupt on   3
      bytes in fifo*/
      SpiaRegs.SPIFFRX.bit.RXFIFORESET = 1;      /*   Enable Receive
channel*/
      SpiaRegs.SPIFFCT.all = 0x00;
      SpiaRegs.SPICCR.bit.SPISWRESET = 1; /* relinquish     from
reset*/
   }
   /*=================================================================
======================
   DSP_SET_SPI_MASTER()
   =================================================================
======================*/
   // Setup the SPI core to function as a master to ready to send
data  (note, this is just     the   core, does not include setup for
   // Mini2810     and   GIIB, so if data needed to be sent out to
      GIIB, more setup is     needed)
   #define   DSP_SET_SPI_MASTER(){   \
      GpioDataRegs.GPFCLEAR.all     =     BIT3;
      SpiaRegs.SPICCR.all     =     0x0000;
      /*Set SPI   to reset state*/
      SpiaRegs.SPICCR.bit.CLKPOLARITY     =     0;
      SpiaRegs.SPICCR.bit.SPICHAR   =     7;    /* 8 bit
characters*/
      SpiaRegs.SPICTL.all     =     0x0000;
      SpiaRegs.SPICTL.bit.CLK_PHASE =     1;
      SpiaRegs.SPICTL.bit.MASTER_SLAVE = 1;     /*Slave*/
      SpiaRegs.SPICTL.bit.TALK = 1;             /* enable
      transmission*/
      SpiaRegs.SPICTL.bit.SPIINTENA =     0;                /*Enable
SPI interrupt*/
      SpiaRegs.SPIBRR   =     4;                      /*
37.5/(SPIBRR+1)MHz      */
      SpiaRegs.SPIFFTX.all = 0x0000;
      SpiaRegs.SPIFFTX.bit.SPIFFENA =     1;    /* enable   tx
fifo*/
      SpiaRegs.SPIFFTX.bit.TXFIFO   =     1;          /* Enable
      Transmit channel*/
```

```
      SpiaRegs.SPIFFTX.bit.SPIRST   =     1;          /* enable
      SPI*/
      SpiaRegs.SPIFFRX.all = 0;
      SpiaRegs.SPIFFRX.bit.RXFFIL   =     3;    /* interrupt on   3
      bytes in fifo*/
      SpiaRegs.SPIFFRX.bit.RXFIFORESET = 1;      /*   Enable Receive
channel*/
      SpiaRegs.SPIFFCT.all = 0x00;
      SpiaRegs.SPICCR.bit.SPISWRESET = 1; /* relinquish     from
reset*/
   }
   /*=================================================================
=====================
   SLAVE_RECIEVE_MASTER()
   =================================================================
=====================*/
   #define   SLAVE_RECIEVE_MASTER(){
      SET_OC_SPI_EN();
      DISABLE_CPLD();
      CLEAR_SPI_MASTER();
      DSP_SET_SPI_SLAVE();
      //SET_TP11();
      wait = 0;
      while(SpiaRegs.SPIFFRX.bit.RXFFST   != NO_SPI_CHAR){
            wait++;
            /*Wait for message to   arrive,     if they     don't
      arrive flag error */
            if(wait     >=300){
                  char_not_recieved =     1;
                  break;
            }
      }
      //CLEAR_TP11();
      checksum_test_tmp =     0;                          /*Reset
      checksum summing*/
      char_no     =     SpiaRegs.SPIFFRX.bit.RXFFST;
      for(i =     0; i<NO_SPI_CHAR; i++){
            tmp   =     SpiaRegs.SPIRXBUF;
            tmp   =     tmp   &     255;
            rec_char_array[i]=tmp;        /*Extracting the lower 8
bits  only*/
            checksum_test_tmp += tmp;           /*Calculate
      checksum*/
      }
      checksum_test_tmp =checksum_test_tmp & 255;
      /*Only lower 8 bits     of checksummatter*/
      if((checksum_test_tmp   &     255) !=     0){               /*
Flagging    Message     recieved is corrupted   */
            checksum_OK_1st   =     0;
      }
      else{
                  /*If checksum     OK,   process     message*/
            checksum_OK_1st   =     1;
            rec_master_status_1st   =     (Uint16)rec_char_array[0];
            rec_master_status_2nd   =     (Uint16)rec_char_array[1];
            index_sin   =     (Uint16)rec_char_array[2]     +
      (((Uint16)rec_char_array[3])<<8);
            mod_targ = (int16)((Uint16)rec_char_array[4] +
(((Uint16)rec_char_array[5])<<8));
            Kp_i = rec_char_array[6] + (rec_char_array[7]<<8);
            Ki_i = rec_char_array[8] + (rec_char_array[9]<<8);
```

```
            feed_forward_mag = rec_char_array[10]     +
      (rec_char_array[11]<<8);
      }

      if(rec_master_status_1st & SW_ENABLE){
            is_switching = 1;
      }else{
            is_switching = 0;
      }

      /*Finish listening switch     to sending mode   ready to send
      once timer compare is   triggered*/ \
      CLEAR_OC_SPI_EN();
      DSP_SET_SPI_MASTER();
      spi_set_mode(MODE_CPLD);
      wait = 0;
      while(wait < 20){
            wait++;
      }
      WriteControlLatch(!CTRL_SPI_SLAVE, LOAD);
      /*Set GIIB SPI direction to   sending     mode for next
      half of     interrupt   cycle*/\
  }
  /*=================================================================
======================
  SLAVE_SEND()
  =================================================================
======================*/
  #define   SLAVE_SEND(){
      SET_OC_SPI_EN();
      slave_status_1st = 100;
      slave_status_2nd = 0;
      curr_comp   =     42500;
      /*Composing message     */
      sync_message[0]    = slave_status_1st;
      sync_message[1]    = slave_status_2nd;
      sync_message[2]    = curr_comp;
      sync_message[3]    = curr_comp>>8;
      /* Generating     modular     sum*/
      checksum = 0;
      for(i =     0; i < 4;   i++){
            checksum += (Uint16)sync_message[i];
      }
      sum   =     checksum;
      checksum = ~checksum;
      checksum += 1;
      sync_message[4]   =     checksum;
      sync_message[5]    = 0;
      sync_message[6]    = 0;
      sync_message[7]    = 0;
      sync_message[8]   =     0;
      sync_message[9]   =     0;
      sync_message[10] = 0;
      sync_message[11] = 0;
      sync_message[12] = 0;
      sync_message[13] = 0;
      sync_message[14] = 0;
      sync_message[15] = 0;
      sum   =     checksum + sum;

      DSP_SET_SPI_MASTER();
```

```
      SET_SPI_MASTER();
      DISABLE_CPLD();
      wait = 0;
      while(wait < 50){
            wait++;
      }

      //SET_TP11();
      for(i =     0; i < NO_SPI_CHAR;     i++){
            spi_putc(sync_message[i]);
      }
      //CLEAR_TP11();

      CLEAR_OC_SPI_EN();
      /*Once sending is done switch to listening mode*/
      DSP_SET_SPI_MASTER();
      spi_set_mode(MODE_CPLD);
      wait = 0;
      while(wait < 10){
            wait++;
      }
      WriteControlLatch(CTRL_SPI_SLAVE,   LOAD);
      /*Set GIIB SPI direction to   listening   mode to     get
      ready for   next interrupt cycle*/\
   }
   /*================================================================
======================
   SLAVE_RECIEVE_SLAVE()
   ================================================================
======================*/
   #define   SLAVE_RECIEVE_SLAVE(){
       SET_OC_SPI_EN();
       DISABLE_CPLD();
       CLEAR_SPI_MASTER();
       DSP_SET_SPI_SLAVE();
       //SET_TP11();
       wait = 0;
       while(SpiaRegs.SPIFFRX.bit.RXFFST   !=    NO_SPI_CHAR){
             wait++;
             if(wait     >=300){
                   char_not_recieved =     1;
                   break;
             }
       }
       //CLEAR_TP11();
       checksum_test_tmp =     0;
       char_no     =     SpiaRegs.SPIFFRX.bit.RXFFST;
       for(i =     0; i<NO_SPI_CHAR; i++){
             tmp   =     SpiaRegs.SPIRXBUF;
             tmp   =     tmp   &255;
             rec_char_array[i]=tmp   &     255;
             checksum_test_tmp += tmp;
       }
       checksum_test_tmp =checksum_test_tmp & 255;
       /*Only lower 8 bits     matter*/

       if((checksum_test_tmp   &     255) !=     0){
             checksum_OK_2nd   =     0;
       }
       else{
             checksum_OK_2nd   =     1;
```

```
      }
      /*Finish listening switch     to sending mode   ready to send
      once timer compare is   triggered*/\
      CLEAR_OC_SPI_EN();
      DSP_SET_SPI_MASTER();
      spi_set_mode(MODE_CPLD);
      wait = 0;
      while(wait < 10){
            wait++;
      }
      WriteControlLatch(!CTRL_SPI_SLAVE, LOAD);
      /*Set GIIB SPI direction to   sending     mode for start of
      next interrup     cycle*/\
   }
   /*=================================================================
======================
   DAC1_FAST_WRITE()
   =================================================================
======================*/
   #define   DAC1_FAST_WRITE(DAC_COMMAND,DAC_ADDRESS,DAC_DATA)\
   {\
       GpioDataRegs.GPASET.all =     OC_SPI_EN;\
       GpioDataRegs.GPASET.all =     M_nS;\
       GpioDataRegs.GPDSET.all =     nDAC2;\
       GpioDataRegs.GPDCLEAR.all     =     nDAC1;\
       spi_fail_count_dac1     =     65535;\
       SpiaRegs.SPITXBUF =     (DAC_COMMAND|DAC_ADDRESS)<<8;\
       SpiaRegs.SPITXBUF =     ((((DAC_DATA<<4)>>8)&0x00FF)<<8);\
       SpiaRegs.SPITXBUF =     ((DAC_DATA<<4)&0x00FF)<<8;\
       while ((SpiaRegs.SPIFFRX.bit.RXFFST <
       3)&&(spi_fail_count_dac1>0)   )\
             {\
                   spi_fail_count_dac1--;\
             }\
             trash_spi_dac1=SpiaRegs.SPIRXBUF;\
             trash_spi_dac1=SpiaRegs.SPIRXBUF;\
             trash_spi_dac1=SpiaRegs.SPIRXBUF;\
       GpioDataRegs.GPDSET.all =     nDAC1;\
       GpioDataRegs.GPDSET.all =     nDAC2;\
   }\
   /*=================================================================
======================
   DAC2_FAST_WRITE()
   =================================================================
======================*/
   #define   DAC2_FAST_WRITE(DAC_COMMAND,DAC_ADDRESS,DAC_DATA)\
   {\
       GpioDataRegs.GPASET.all =     OC_SPI_EN;\
       GpioDataRegs.GPASET.all =     M_nS;\
       GpioDataRegs.GPDSET.all =     nDAC1;\
       GpioDataRegs.GPDCLEAR.all     =     nDAC2;\
       spi_fail_count_dac2     =     65535;\
       SpiaRegs.SPITXBUF =     (DAC_COMMAND|DAC_ADDRESS)<<8;\
       SpiaRegs.SPITXBUF =     ((((DAC_DATA<<4)>>8)&0x00FF)<<8);\
       SpiaRegs.SPITXBUF =     ((DAC_DATA<<4)&0x00FF)<<8;\
       while ((SpiaRegs.SPIFFRX.bit.RXFFST <
       3)&&(spi_fail_count_dac2>0)   )\
             {\
                   spi_fail_count_dac2--;\
             }\
             trash_spi_dac2=SpiaRegs.SPIRXBUF;\
```

```
            trash_spi_dac2=SpiaRegs.SPIRXBUF;\
            trash_spi_dac2=SpiaRegs.SPIRXBUF;\
      GpioDataRegs.GPDSET.all =     nDAC2;\
      GpioDataRegs.GPDSET.all =     nDAC1;\
   }\
   /*
=================================================================
=====
   __Types()
   ================================================================
========== */
   /// Internal ADC channel type
   /** This structure hold     variables   relating to a     single
ADC channel.      These
   variables are   used for filtering,     averaging, and scaling of
      this analog
   quantity. */
   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;
   /// Internal ADC storage type
   /** 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.
   The vout and iout     channels are for DC     measurements of
      the   VSI   outputs     when it
   is producing a DC     output.     */
   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     averaging
            count_dc_bak,     ///< background   copy of     DC
counter
            flag_cal,   ///< flag   set   to trigger background
      calibration averaging
            flag_rms,   ///< flag   set   to trigger background
      RMS   averaging
            flag_dc; ///<     flag set to trigger     background DC
      averaging
      type_adc_ch
```

```
            A0,   ///< ADC channel A0
            A1,   ///< ADC channel A1
            A2,   ///< ADC channel A2
            A3,   ///< ADC channel A3
            A4,   ///< ADC channel A4
            A5,   ///< ADC channel A5
            B0,   ///< ADC channel B0
            B1,   ///< ADC channel B1
            B2,   ///< ADC channel B2
            B3,   ///< ADC channel B3
            B4,   ///< ADC channel B4
            B5,   ///< ADC channel B5
            yHA, ///<   bank A high reference
            yLA, ///<   bank A low reference
            yHB, ///<   bank B high reference
            yLB; ///<   bank B low reference
  }   type_adc_int;
  /*
=========================================================================
=====
  __Variables()
  =========================================================================
========== */
  // state machine level variables
  Uint16
      vsi_status = 0,               ///   Status of   VSI   system
      int_count   =     0,
      is_switching = 0,       // flag     set   if PWM switching is
      active
      vsi_counter =     0;          // counter for timing   VSI
      regulation events
  //Timer   period and switching frequency related variable
  //Initialized   to default value as     set   by #define values at
top of      this file
  Uint16
      sw_freq     =     SW_FREQ,
      period_2 = PERIOD_2,
      period = PERIOD;
  Uint32
      PHASE_STEP_SC     =     PHASE_STEP_SC_D   ;
  /// Maximum     VSI   switching   time in     clock ticks
  int16
      MAX_TIME    =     (int16)(PERIOD_2-6)     ;
  int16
      V_Asat=0,   V_Bsat=0,   V_Csat=0;
  double
      Ref_freq_float = INIT_FF;
  // PWM Timer interrupt variables
  // 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
  // WYK note: Sin table contain 1024     entire with peak value of
      +-16384
  int16
      *sin_table = (int16     *)0x003FF000,     // pointer to
      sine table in     boot ROM
      *cos_table = (int16     *)0x003FF100,     // pointer to
      cos   table in boot     ROM
```

```
      phase_offset,                  // round off amount     from
sine  lookup
      val_diff,                      // interpolation temp   variable
      val_lo,                              // interpolation temp
      variable
      sin_val,                       // interpolated   sine table
value
      cos_val,                       // interpolated   cosine table
value
      sin_PI_on_12_val,
      sin_PI_on_3_val,
      sin_PI_on_6_val,
      sin_2PI_on_3_val,
      sin_4PI_on_3_val,
      sin_val_test,
      sin_2PI_on_3_val_test,
      sin_4PI_on_3_val_test,
      sin_90_val,
      sin_210_val,
      sin_330_val,
      sin_PI_on_2_val,
      cos_PI_on_3_val,
      ia_err,     ib_err,     ic_err,
      DC_val;
  Uint32
      phase_step = PHASE_STEP,// change   in phase angle each
      interrupt
      phase =     0L;                    // running phase angle
      (2^32 == 360degrees)
  //Calculate     phase offset to   initialized phase value to
enable      reading     of sin table to   generate various
differernt  trignometrey lookup     needed
  Uint32
      phase_sin   =     (long)65536.0     *     0.0   *     65536.0,
      phase_sin_PI_on_3 =     (long) 65536.0*(1.0/6.0) * 65536.0,
      phase_sin_PI_on_6 =     (long)65536.0*(1.0/12.0) * 65536.0,
      phase_sin_2PI_on_3 = (long)   65536.0/3.0 *     65536.0,
      phase_sin_4PI_on_3 = (long)   65536.0     *     (2.0/3.0)   *
      65536.0,
      phase_sin_PI_on_2 =     (long) 65536.0/2.0 * 65536.0,
      phase_sin_PI_on_12 = (long)65536.0*(1.0/24)     *     65536.0,
      phase_cos,//=     (long) 65536.0*(0.25)   *     65536.0,
      phase_cos_PI_on_3 =     (long) 65536.0*(0.25+1.0/6.0) *
      65536.0,
      phase_sin_test                 =     (long) 65536.0 *
(1.0/23.0)  *     65536.0,
      phase_sin_2PI_on_3_test =     (long) 65536.0 * (1.0/23.0)   *
      65536.0     +     (long) 65536.0/3.0 * 65536.0,
      phase_sin_4PI_on_3_test =     (long) 65536.0 * (1.0/23.0)   *
      65536.0     +     (long) 65536.0 * (2.0/3.0) * 65536.0,
      phase_sin_90      =     (long) 65536.0 * (1.0/4.0) * 65536.0,
      phase_sin_210     =     (long) 65536.0 * (21.0/36.0) *
65536.0,
      phase_sin_330     =     (long) 65536.0 * (33.0/36.0) *
65536.0;//(long)  65536.0     *     (1.0/4.0)   *     65536.0     +
      (long) 65536.0 * (2.0/3.0) * 65536.0;
  Uint16                     // index into     sine look-up table
(phase      >> 22),     this give   a     10 bit number     which
      can   be used     to read     sin   table
      index =     0,
      index_sin   =     0,
```

```
      index_sin_PI_on_3 =     0,
      index_sin_PI_on_6 =     0,
      index_sin_PI_on_12 = 0,
      index_sin_2PI_on_3 = 0,
      index_sin_4PI_on_3 = 0,
      index_cos   =     0,
      index_cos_PI_on_3 =     0,
      index_sin_test                =     0,
      index_sin_2PI_on_3_test =     0,
      index_sin_4PI_on_3_test =     0,
      index_sin_90      =     0,
      index_sin_210     =     0,
      index_sin_330     =     0;
  int16
      V_A,                              // demanded voltages
      t_A,                          // switching times
      t_B,
      t_C,
      Voff,
      Voff_prev,
      Voff_int,
      Voff_int_prev,
      Voff_scaled,                  //3rd harmonic offset
      Voff_scaled_prev,             //3rd harmonic offset
      mod_targ = 0,                 // target   modulation depth
      mod_ref     =     0;              // background
      reference   mod   depth
  /// fault variables
  Uint16
      detected_faults   =     0;    // bits     set   for   faults
detected    (possibly   cleared)
  /** @name Internal ADC Variables */
  //@{
  type_adc_int
      adc_int     =
      {
            0, // count_cal
            0, // count_rms
            0, // count_rms_bak
            0, // count_dc
            0, // count_dc_bak
            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
                  }, // #A0
  //        {     0, 0, 0L,   0L,   0L,   0L,   0.0   }, // #A1
  //        {     0, 0, 0L,   0L,   0L,   0L,   0.0   }, // #A2
  //        {     0, 0, 0L,   0L,   0L,   0L,   0.0   }, // #A3
  //        {     0, 0, 0L,   0L,   0L,   0L,   0.0   }, // #A4
  //        {     0, 0, 0L,   0L,   0L,   0L,   0.0   }, // #A5
            {     0, 0, 0L,   0L,   0L,   0L,   0.0   }, // #B0
  //        {     0, 0, 0L,   0L,   0L,   0L,   0.0   }, // #B1
  //        {     0, 0, 0L,   0L,   0L,   0L,   0.0   }, // #B2
  //        {     0, 0, 0L,   0L,   0L,   0L,   0.0   }, // #B3
```

```
  //        {     0, 0, 0L,   0L,   0L,   0L,   0.0   }, // #B4
  //        {     0, 0, 0L,   0L,   0L,   0L,   0.0   }, // #B5
            {     0, 0, 0L,   0L,   0L,   0L,   0.0   }, // yHA
            {     0, 0, 0L,   0L,   0L,   0L,   0.0   }, // yLA
            {     0, 0, 0L,   0L,   0L,   0L,   0.0   }, // yHB
            {     0, 0, 0L,   0L,   0L,   0L,   0.0   }, // yLB
      };
  // ADC calibration variables
  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
  double
      cal_gain_A, cal_gain_B,
      cal_offset_A,     cal_offset_B;
  //@}
  //ADC     value holder
  int16
      I_res_A,
      I_res_B,
      I_res_C,
      I_err_A,
      I_err_B,
      I_err_C,
      I_err_A_prev,
      I_err_B_prev,
      I_err_C_prev,
      I_err_err_prev,
      I_err_A_deriv,
      I_err_B_deriv,
      I_err_C_deriv,
      I_err_A_deriv_prev,
      I_err_B_deriv_prev,
      I_err_C_deriv_prev,
      filtered_output,
      filtered_output_prev,
            I_err_A_deriv_alpha,
      VAC_A,                          //Voltage   measuremnet of phase
A of  grid to     natural
      VAC_B,                          //Voltage   measuremnet of phase
B of  grid to     natural
      V_sw,
      Vdc;
  //Reference     mode for current regulator
  int16     refMode     =     THREE_PHASE_OL;
  signed int I_ref_Peak_AB = 0 , I_ref_Peak_AB_prev   =     0     ;
  signed int I_ref_Peak_AB_tr =     0;
  //Control loop variables
  //Stationary frame PI regulator   internal variables
  int16
      I_ref_A     =     0,
      I_ref_B     =     0,
      I_ref_C     =     0,
      Ma_virt     =     0,
      Mb_virt     =     0,
      Mc_virt     =     0,
```

```
      I_ref_D     =     0,
      ia_err,     ib_err,     ic_err,
      sw_a = LOW,
      sw_a_old = LOW,
      sw_b = LOW,
      sw_b_old = LOW,
      sw_c = LOW,
      sw_c_old = LOW,
      burn = 0,   actra_on = 0,     actra_off   =     0;
  int32
      err_i_prop_A=0,
      err_i_int_now_A=0,
      err_i_int_total_A=0;
  int32
      err_i_prop_B=0,
      err_i_int_now_B=0,
      err_i_int_total_B=0;
  int16
      Error_I_A,
      Error_I_B,
      Command_A,                          //Controller output
      variable
      Command_B;                          //Controller output
      variable
  //Hysteresis Controller     Parameter
  int16
      ia_err,     ib_err,     ic_err,
      ia,   ib,   ic,
      ia_cmd,     ib_cmd,     ic_cmd,temp_band,beta;
  //Controller tuning   in integer form   used by     controller
calculation
  int16
      Kp_i = 0,                           //Proportional constant in
ADC count and timer count
      Ki_i = 0,                           //Integral constant     in
ADC count and timer count
      MODMAX = 8191,          //Maximum   modulation index 100%
      modulation
      ADC_offset = ADC_OFFSET,
      // add_phase is   the   size of     step jump   in phase,
      the   extra_phase should be   added to
      // the phase for reading the sintable, as it is a     record
of    all   the   step change in phase
      // requested so   far
      add_phase   =     0;
  long o = 0 , j = 0 , z = 0;
  //Delta   transform   variables
  double
      delta, one_on_delta, w_c,     w_0, Ts;
  //S domain transfer   function of P+Resonant controller
  //Function of   form H(s)   =     (bs_0*s^2   +     bs_1*s +
bs_2)/(as_0*s^2 + as_1*s      +     as_0)
  double
      bs_0, bs_1, bs_2, as_0, as_1, as_2;
  //Z domain transfer   function of P+Resonant controller
  //Function of   form H(z)   =     (bz_0 +     bz_1*z^-1
      +bz_0*z^-2)/(az_0 +     az_1*z^-1   +az_0*z^-2)
  double
      bz_0, bz_1, bz_2, az_0, az_1, az_2;
  //Delta   domain transfer   function of P+Resonant controller   in
floating    point form
```

```
  //Function of   form H(d)   =     (beta_0_f   +     beta_1_f*d^-1
     +     beta_2_f*d^-2)/(1 +     alpha_1_f*d^-1 + alpha_2_f*d^-2)
  double
     alpha_1_f, alpha_2_f,   beta_0_f,   beta_1_f,   beta_2_f;
  int16
     beta_0,     beta_1,     beta_2,     alpha_0, alpha_1, alpha_2;
  double
     Ki_i_f,     Kp_i_f;
  //Interal variables   for   phase A     of P+R controller
  long
     Error_I_L_A=0;
  int
     s0_1_fp_A   =     0, s0_0_fp_A = 0,
     s1_1_fp_A   =     0, s1_0_fp_A = 0,
     s2_1_fp_A   =     0, s2_0_fp_A = 0;
  long
     branch1_A=0, branch2_A=0,     branch3_A=0, branch4_A=0,
     branch5_A=0;
  Uint16
     togg = 0,
     sw = 0,     Inom,
     step_tog=0;                          //Toggle channel 2 of
     Dig   IO for step change in   reference
  //Interface     variables   used to     recieve     controller
loop  parameters from   background
  //Controller loop     turning     parameters in     real floating
     pointer     number from background
  double
     real_KP           =KP_INIT,
     real_TINT   =TINT_INIT,
     real_Hband = BMAX ;
  //For     step change in reference in   AC reference modes
  int16     count_from_zero_for_step = 0;
  int16     new_mod_targ = 0;
  int16     step_ref_request = 0;
  int16     step_phase_request = 0;
  int16     prev_sin_table_sign     =     0;
  int16     step_0_ref_A = 0;
  //Feedforward   +     bus   compensation related variables
  int FFenable = DISABLE;
                 //Feedforward     status
  long FF_amount_A = 0;
  long FF_amount_B = 0;
  long ZERO =     0;
  int count_per_A =     ADC_I1_SC_SCALED;
     //Scaled version of     Amp   per   count used in
     calculation
  int VBUS = 0;                           //DC bus voltage
  int cond = 0;
  int one_on_vbus =0;
  int inverse_INOM = (int)((1.0/(float)I_NOM)*(long)(1l<<16));
  int DAC_out     =     0;
  int fundament_frequency     =     0;
     //Fundamental     frequency   multiple by 256
  //int     inverse_bus_v_array[BUS_ARRAY_SIZE];
            //Array     contain     the   inverse     of bus voltage
multiple    by a constant     for   bus   compensation calculation
  /* Zero   crossing variables */
  unsigned int
     in_sync,
     ZX_in_sync,
```

```
      ZX_state,
      ZX_count,
      ZX_seen,
      ZX_cycles,
      ZX_sum;
  signed int
      ZX_time,
      ZX_time_phase,
      ZX_phase_scale,
      ZX_phase_err,
      ZX_err_sum;
  int phase_trim = 0;
  int16     loop_back_character     =     0;
  int16     carrier_capture   =     0;
  int16     carrier_error     =     0;
  int16     carrier_adjust = 0;
  int16     period_load =     0;
  int16     carrier_count     =     0;
  int16     carrier_sync = 0;
  int16     carrier_offset = 50;
  int16     carrier_allowed_error   =     4;
  //Master slave mode   select.      0 being slave,   1     being
      master
  int16     master_slave_mode =     0;
  int16     loop_no     =     0;
  int16     adjust_count = 0;
  int16     sync_method =     0;
  int16     UF_P = 0;
  int16     adc_ready_flag = 0;
  int16     timer1_cmp_flag   =     0;
  //Internal variable   for   message     to be send via spi
  Uint16 sync_message[16];
  //Variables     for   sync message
  Uint16 mas_status_1st =     0;
  Uint16 mas_status_2nd =     0;
  Uint16 slave_status_1st     =     0;
  Uint16 slave_status_2nd     =     0;
  Uint16 curr_comp = 0;
  Uint16 feed_forward_mag     =     0;
  //Array   containing test   SPI   characters
  Uint16 rec_char_array[16];
  int16     NO_SPI_CHAR =     13;
  Uint16 checksum =     0;
  Uint16 checksum_test_tmp=0;
  //Checksum OK   flag for the two SPI blocks
  Uint16 checksum_OK_1st = 0;
  Uint16 checksum_OK_2nd = 0;
  extern int16 Unit_number;
  //Master status recieved from     comm link, invaild if   unit is
      master
  Uint16 rec_master_status_1st = 0;
  Uint16 rec_master_status_2nd = 0;
  //Recieved slave status     recieved from     comm link
  Uint16 rec_slave_status_1st =     0;
  Uint16 rec_slave_status_2nd =     0;
  //Recieved current compensation   signal from other slave, useless
if    unit is     master
  Uint16 rec_curr_comp = 0;
  Uint16 period_load_2 = 0;
  Uint16 mas_fault = 0;
  Uint16 slav_fault     =     0;
```

```
  Uint16 slave_delay = 0;                        //Flag to   indicate
if    unit's carrier is phase delayed     from master unit by     90
degree
  //Compensator   filter co-efficient
  int16
      beta_com_0, beta_com_1, beta_com_2, alpha_com_0, alpha_com_1,
      alpha_com_2;
  int16
      rect_i_A = 0,     com_filter_output;
  long
      rect_i_A_L = 0;
  int
      s0_1_com = 0,     s0_0_com = 0,
      s1_1_com = 0,     s1_0_com = 0,
      s2_1_com=   0, s2_0_com =     0;
  long
      branch1_A_com=0, branch2_A_com=0,   branch3_A_com=0,
branch4_A_com=0,  branch5_A_com=0;
  long
                I_res_DC_offset   =0,
                I_res_cycle_total =     0,
                I_res_cycle_count =     0;
  //Variable for high   pass PI
  double
      a0 = 0,     a1 = 0,     b1 = 0;
  long
      yn = 0,     yn_1 = 0;
  int16
      xn = 0,     xn_1 = 0;
  int32
      a0_fp =     0, a1_fp = 0,     b1_fp =     0, Kp_fp = 0;
  int32
      HP_PI_prop = 0,   HP_PI_integral = 0;
  int32
      one_on_INOM =     0;
  double
      yn_double   =     0, yn_1_double = 0,     xn_double   =     0,
xn_1_double = 0,  PI_output_double = 0;
  double
      Kp_double   =     0;
  Uint16
      prev_index_sin = 0;
  Uint16    PWMA1,
            PWMA2,
            PWMA3,
            PWMA4,
            PWMA5,
            PWMA6,
            PWMA7,
            PWMA8,
            PDPINT_Flag =     0     ,
            t     =     0,
            hyst_pwm_status   ;
  extern int16 real_sw_freq_serial ;
  extern int      real_bus_voltage_serial,
                  real_V3rd_serial,
                  real_V3rd_offset_serial,
                  average_offset_serial,
                  real_fixed_band_serial,
                  real_fixed_band_serial_B,
                  real_fixed_band_serial_C,
```

```
                mod_depth_serial ;
  int           mod_depth   ;
  extern int16    stepping   ;
  /*
=============================================================================
=====
  __Fast SPI Write Variables
  ============================================================================
========== */
  Uint16 spi_fail_count_dac1,spi_fail_count_dac2;
  int16    trash_spi_dac1,trash_spi_dac2;
  /*
=============================================================================
=====
  __Capture Port Variables
  ============================================================================
========== */
  long            PWM_STATE               =     0     ,
           PWM_OFF                        =     0     ,
           PWM_ON                         =     0     ,
           flag_R_synch      =     0     ,
           flag_F_synch      =     0     ,
           flag_R_cap        =     0     ,
           flag_F_cap        =     0     ,
           count_cap_select = 0 ,
           zero_crossing_clock            =     0     ,
           first_capture_A                                  =     0
     ,
           first_capture_new_A                        =     0     ,
           first_capture_new_new_A        =     0     ,
           first_capture_old_A                        =     0     ,
           second_capture_A                           =     0     ,
           second_capture_new_A                 =     0     ,
           zero_crossing_capture_A        =     0     ,
           zero_crossing_count                  =     0     ,
           capture_difference_A                 =     0     ,
           band_offset_A                                          =
     0     ,
           band_offset_A_OFF                                      =
     0     ,
           zero_crossing_on_flag_A        =     0     ,
           zero_crossing_on_flag_B        =     0     ,
           zero_crossing_on_flag_C        =     0     ,
           first_capture_B                                  =     0
     ,
           first_capture_new_B                        =     0     ,
           first_capture_new_new_B        =     0     ,
           first_capture_old_B                        =     0     ,
           second_capture_B                           =     0     ,
           second_capture_new_B                 =     0     ,
           zero_crossing_capture_B        =     0     ,
           capture_difference_B                 =     0     ,
           capture_difference_prev_B      =     0     ,
           band_offset_B                                          =
     0     ,
           band_offset_B_OFF                          =     0     ,
           first_capture_C                                  =     0
     ,
           first_capture_new_C                        =     0     ,
           first_capture_new_new_C        =     0     ,
           first_capture_old_C                        =     0     ,
```

```
            second_capture_C                        =     0     ,
            second_capture_new_C                =     0     ,
            zero_crossing_capture_C       =     0     ,
            capture_difference_C                =     0     ,
            band_offset_C                                       =
      0     ,
            band_offset_C_OFF                       =     0     ,
            on_time_A
      =     0     ,
            off_time_A                                          =
      0     ,
            off_time_A_temp                         =     0
      ,
            first_temp                                          =
0,
            second_temp                                         =
      0,
            on_time_A_prev                          =     0
      ,
            off_time_A_prev                         =     0
      ,
            tot_time_A                                          =
      0     ,
            on_time_B
      =     0     ,
            off_time_B                                          =
      0     ,
            on_time_B_prev                          =     0
      ,
            off_time_B_prev                         =     0
      ,
            tot_time_B                                          =
      0     ,
            on_time_C
      =     0     ,
            on_time_C_prev                          =     0
      ,
            off_time_C                                          =
      0     ,
            off_time_C_prev                         =     0
      ,
            tot_time_C                                          =
      0     ,
            bus_voltage                                         =
      0     ,
            V3rd_count                                          =
      0     ,
            V3rd_offset                                         =
      0     ,
            average_offset                          =     0
      ,
            hyst_band
      =     0     ,
            hyst_band_B                                         =
      0     ,
            hyst_band_C                                         =
      0     ,
            temp_stack_A                                        =
      0     ,
            temp_stack_B                                        =
      0     ,
```

```
            temp_stack_C                                                =
      0     ,
            third_capture                                               =
      0,
            counter_operation                           =     0,
            sw_frequency                                                =
      0,
            period_hysteresis                           =     0,
            PWM_edge_count_A                            =     0,
            PWM_edge_count_B                            =     0,
            PWM_edge_count_C                            =     0,
            first_edge_count                            =     1,
            first_edge_count_B                          =     1,
            first_edge_count_C                          =     1,
            PWM_status_A                                                =
      0,
            PWM_status_B                                                =
      0,
            PWM_status_C                                                =
      0,
            i                                                           =
      0     ,
            one_forth_count         =     0     ,
            one_second_count  =     0     ,
            DAC_REF_FLAG                =     0     ,
            peak_select                 =     0     ,
            tempp;
   int16                MDepth_max_main   =     0     ,
                              MDepth_C    =     0     ,     MDepth_B
= 0 , MDepth_A    =     0     ,
                              command_C_virt = 0 , duty_cycle_C_virt
= 0 ,
                              duty_cycle_C_virt_prev =
0,duty_cycle_C_virt_prev_prev = 0,
                              duty_cycle_A = 0,duty_cycle_prev_A = 0
,duty_cycle_A_3rd = 0,
                              duty_cycle_B      =
      0,duty_cycle_prev_B     =     0,duty_cycle_B_3rd = 0,
                              duty_cycle_C      =
      0,duty_cycle_prev_C     =     0,duty_cycle_C_3rd = 0,
                              Vband_A     =     0, Vband_prev_A   =
      0     ,     Vband_B     =     0, Vband_prev_B   =     0,
                              Vband_C     =     0     ,
      Vband_prev_C = 0,count_A = 0 ,count_B     =     0,count_C   =
      0     ,
                              on_flag_A   =     0     ,off_flag_A =
      0     ,on_flag_B = 0 ,off_flag_B = 0 ,on_flag_C =     0
      ,off_flag_C =     0,
                              V_L_A,V_L_B,V_L_C ,
      I_ref_L_A,I_ref_L_B,I_ref_L_C;
   signed int
            capture_top_stack =     0     ,
            capture_bot_stack =     0     ;
   int16    stepping_counter = 0 ;
   /*
==========================================================================
=====
   __Local_Function_Prototypes()
   ==========================================================================
========== */
   /* vsi state machine state functions */
```

```
   void
       st_vsi_init(void),            // initialises CFPP
       regulator
       st_vsi_stop(void),            // waiting for start trigger
       st_vsi_gate_charge(void),     // delay to charge the high
       side gate   drivers
       st_vsi_ramp(void),            // ramping to     target mod
depth
       st_vsi_run(void),       // maintaining target   mod   depth
       st_vsi_fault(void);           // delay after faults   are
       cleared
   /// ADC   and   VSI   interrupt
   interrupt void isr_adc(void);
   /// Gate fault (PDPINT)     interrupt
   interrupt void isr_gate_fault(void);
   /// Calibrates the adc for gain   and   offset using the reference
inputs.
   void calibrate_adc(void);
   /// Scales the RMS summations     to real     volts and   amps
   void scale_adc_rms(void);
   /// Scales the DC     summations to     real volts and amps
   void scale_adc_dc(void);
   //set     and   scale hystertesis band
   void SetHystBand(double);
   void SetBusVolts(double);
   // Timer 1 underflow interrupt
   //interrupt     void isr_T1UF(void);
   // Timer 1 period     interrupt
   //interrupt     void isr_T1P(void);
   // Capture port interrupt
   interrupt void isr_CAP1(void);
   interrupt void isr_CAP2(void);
   interrupt void isr_CAP4(void);
   interrupt void isr_CAP5(void);
   interrupt void isr_CAP6(void);
   interrupt void isr_T2P(void);
   interrupt void isr_SPIRX(void);
   interrupt void isr_pwm(void);
   interrupt void isr_T1CINT(void);
   /*
================================================================
===== */
   /* State Machine Variable   */
   /*
================================================================
===== */
   type_state
       vsi_state   =
       {
             &st_vsi_init,
             1
       };
   /*
================================================================
=====
   __Exported_ADC_Functions()
   ================================================================
========== */
   /**
   This function   initialises the   ADC   and   VSI   interrupt
       module.     It sets     the
```

```
  internal ADC to sample the DA-2810 analog     inputs and timer1 to
generate    a     PWM
  carrier   and   the   event manager     A     to generate the
      VSI   switching. It     also
  initialises     all   the   relevant variables and sets   up the
interrupt service
  routines.
  It initialises the Evant Manager A unit to:
  -   drive PWM1-4 as   PWM   pins not GPIO
  -   a     0.48ns deadtime   between     the   high and low side
      pins
  -   Timer 1     as an up/down     counter     for   the   PWM
      carrier
  It initialises the PIE unit to:
  -   Take PDPINTA as   a     power stage interrupt
  -   Use   the   internal ADC completion interrupt   to trigger the
main  ISR
  \author   A.McIver
  \par History:
  \li 12/10/07 AM -     initial     creation
  */
  void vsi_init(void)
  {
      EvaRegs.GPTCONA.all     =     0x0000;
      EvaRegs.EVAIMRA.all     =     0x0000;
      EvaRegs.EVAIFRA.all     =     BIT0;
      EvaRegs.COMCONA.all     =     0x0000;
      EvaRegs.ACTRA.all =     0x0000;
  // Set up ISRs
      EALLOW;
      PieVectTable.ADCINT     =     &isr_adc;
      PieVectTable.PDPINTA = &isr_gate_fault;
      PieVectTable.T1UFINT = &isr_pwm;
      PieVectTable.T1PINT     =      &isr_pwm;
      PieVectTable.CAPINT1 = &isr_CAP1;
      PieVectTable.CAPINT2 = &isr_CAP2;
      PieVectTable.CAPINT4 = &isr_CAP4;
      PieVectTable.CAPINT5 = &isr_CAP5;
      PieVectTable.CAPINT6 = &isr_CAP6;
      PieVectTable.T1CINT     =     &isr_T1CINT;
      EDIS;
  // Set up compare     outputs
      EALLOW;
      GpioMuxRegs.GPDMUX.all = BIT0;
      GpioMuxRegs.GPDQUAL.bit.QUALPRD     =     6; // 500ns
      qualification     period
      //Enable test     pin   on EVB
      GpioMuxRegs.GPBMUX.bit.T3PWM_GPIOB6 =     0;
      GpioMuxRegs.GPBMUX.bit.T4PWM_GPIOB7 =     0;
      GpioMuxRegs.GPBDIR.bit.GPIOB6 =     1;
      GpioMuxRegs.GPBDIR.bit.GPIOB7 =     1;
      //Enable first pin of   EVB   as digout   pins for outputing
sync  signal
      GpioMuxRegs.GPBMUX.bit.PWM7_GPIOB0 = 0;
      GpioMuxRegs.GPBDIR.bit.GPIOB0 =     1;
      //Enable second   pin   of EVB as   digout pins for   as a
test  point
      GpioMuxRegs.GPBMUX.bit.PWM8_GPIOB1 = 0;
      GpioMuxRegs.GPBDIR.bit.GPIOB1 =     1;
      //Enable second   pin   of EVB as   digout pins for   as a
test  point
```

```
      GpioMuxRegs.GPBMUX.bit.PWM9_GPIOB2 = 0;
      GpioMuxRegs.GPBDIR.bit.GPIOB2 =     1;
      //Allow     SPISTE pin to     be GPIO     pin   so it can   be
pull  low   to enable   SPI   in slave mode
      GpioMuxRegs.GPFMUX.bit.SPISTEA_GPIOF3     =     0;
      GpioMuxRegs.GPFDIR.bit.GPIOF3 =     1;
      GpioMuxRegs.GPBMUX.bit.CAP4Q1_GPIOB8       = 1 ;
      //Enabling Capture port 4     pins
      GpioMuxRegs.GPBMUX.bit.CAP5Q2_GPIOB9       = 1 ;
      //Enabling Capture port 5     pins
      GpioMuxRegs.GPBMUX.bit.CAP6QI2_GPIOB10 = 1 ;    //Enabling
Capture port      6     pins
      EDIS;
      EvaRegs.DBTCONA.bit.DBT       =     8;
      EvaRegs.DBTCONA.bit.EDBT1     =     1;
      EvaRegs.DBTCONA.bit.EDBT2     =     1;
      EvaRegs.DBTCONA.bit.EDBT3     =     1;
      EvaRegs.DBTCONA.bit.DBTPS     =     6;
      EvaRegs.CMPR1     =     PERIOD_2;
      EvaRegs.CMPR2     =     PERIOD_2;
      // Setup and load COMCONA
      EvaRegs.COMCONA.bit.ACTRLD = 2;           // reload   ACTR
immediately
      EvaRegs.COMCONA.bit.SVENABLE = 0;   // disable space vector
      PWM
      EvaRegs.COMCONA.bit.CLD =     1;          // reload   on
underflow or      period match
      EvaRegs.COMCONA.bit.FCOMPOE   =     1;    // full     compare
      enable
      EvaRegs.COMCONA.bit.CENABLE   =     1;    // enable   compare
      operation
  // Set up Timer 1
      EvaRegs.T1CON.all =     0x0000;
      EvaRegs.T1PR = period;
      EvaRegs.T1CMPR = 0;
      EvaRegs.T1CNT     =     0x0000;
      EvaRegs.GPTCONA.bit.T1CMPOE   =     1;
      EvaRegs.GPTCONA.bit.T1TOADC   =     2;          // Timer 1
period      starts ADC
      //EvaRegs.GPTCONA.bit.T2TOADC =     2;          // Timer 2
period      starts ADC
      EvaRegs.GPTCONA.bit.TCMPOE    =     1;
      EvaRegs.GPTCONA.bit.T1PIN      = 1;
  // Set up ADC
      AdcRegs.ADCMAXCONV.all = 0x0007;                    // 8
double      conv's (16 total)
      AdcRegs.ADCCHSELSEQ1.bit.CONV00     =     0x0;  // Setup
ADCINA0/B0  as 1st SEQ1 conv.
      AdcRegs.ADCCHSELSEQ1.bit.CONV01     =     0x1;  // Setup
ADCINA1/B1  as 2nd SEQ1 conv.
      AdcRegs.ADCCHSELSEQ1.bit.CONV02     =     0x2;  // Setup
ADCINA2/B2  as 3rd SEQ1 conv.
      AdcRegs.ADCCHSELSEQ1.bit.CONV03     =     0x3;  // Setup
ADCINA3/B3  as 4th SEQ1 conv.
      AdcRegs.ADCCHSELSEQ2.bit.CONV04     =     0x4;  // Setup
ADCINA4/B4  as 5th SEQ2 conv.
      AdcRegs.ADCCHSELSEQ2.bit.CONV05     =     0x5;  // Setup
ADCINA5/B5  as 6th SEQ2 conv.
      AdcRegs.ADCCHSELSEQ2.bit.CONV06     =     0x6;  // Setup
ADCINA6/B6  as 7th SEQ2 conv.
```

```
      AdcRegs.ADCCHSELSEQ2.bit.CONV07     =     0x7;  // Setup
ADCINA7/B7  as 8th SEQ2 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 = 0;           // disable
interrupt
      AdcRegs.ADCTRL2.bit.INT_MOD_SEQ1 = 1;           // int at
      end   of every SEQ1
      AdcRegs.ADCTRL2.bit.INT_MOD_SEQ2 = 1;
      AdcRegs.ADCTRL3.bit.SMODE_SEL =     1;                //
simultaneous      sampling mode
      AdcRegs.ADCTRL3.bit.ADCCLKPS = 0x04;            // ADCLK =
HSPCLK/8    (9.375MHz)
      /* ADC Registers:
   -   ADCRESULT0  =     ADCINA0     -     NC-A:I3     or NO-A:APOT1
   -   ADCRESULT1  =     ADCINB0     -     Vdc2
   -   ADCRESULT2  =     ADCINA1     -     NC-A:Vac3   or NO-A:Vdc3
   -   ADCRESULT3  =     ADCINB1     -     I5
   -   ADCRESULT4  =     ADCINA2     -     I1
   -   ADCRESULT5  =     ADCINB2     -     I4
   -   ADCRESULT6  =     ADCINA3     -     Vac1
   -   ADCRESULT7  =     ADCINB3     -     Vdc1
   -   ADCRESULT8  =     ADCINA4     -     I2
   -   ADCRESULT9  =     ADCINB4     -     NO-B:APOT2 or     NC-B:I6
   -   ADCRESULT10 =     ADCINA5     -     Vac2
   -   ADCRESULT11 =     ADCINB5     -     NO-B:Vgen   or NC-B:Vdc4
   -   ADCRESULT12 =     ADCINA6     -     yHA
   -   ADCRESULT13 =     ADCINB6     -     yHB
   -   ADCRESULT14 =     ADCINA7     -     yLA
   -   ADCRESULT15 =     ADCINB7     -     yLB
      /* ADC Registers& IIB   Mapping
   -   ADCRESULT0  =     ADCINA0     -     ADC0-VDC1
   -   ADCRESULT1  =     ADCINB0     -     ADC1-APOT1
   -   ADCRESULT2  =     ADCINA1     -     ADC2-I1
   -   ADCRESULT3  =     ADCINB1     -     ADC3-VAC1
   -   ADCRESULT4  =     ADCINA2     -     ADC4-I3
   -   ADCRESULT5  =     ADCINB2     -     ADC5-I4
   -   ADCRESULT6  =     ADCINA3     -     ADC6-VAC4
   -   ADCRESULT7  =     ADCINB3     -     ADC7-I6
   -   ADCRESULT8  =     ADCINA4     -     ADC8-VDC2
   -   ADCRESULT9  =     ADCINB4     -     ADC9-APOT2
   -   ADCRESULT10 =     ADCINA5     -     ADC10-I2
   -   ADCRESULT11 =     ADCINB5     -     ADC11-VAC2
   -   ADCRESULT12 =     ADCINA6     -     ADC12-VAC3
   -   ADCRESULT13 =     ADCINB6     -     ADC13-I5
   -   ADCRESULT14 =     ADCINA7     -     ADC14-VAC5
   -   ADCRESULT15 =     ADCINB7     -     ADC15-VAC6
  */
  // Enable interrupts
      DINT;
      EvaRegs.EVAIMRA.all     =     0;                      //
disable all interrupts
      EvaRegs.EVAIFRA.all     =     BIT0|BIT7|BIT9|BIT8;
            //PDPINTA, T1UFINT,     T1PINT,     T1CINTenabled
      EvaRegs.EVAIMRA.bit.T1UFINT   =     1;
      EvaRegs.EVAIMRA.bit.T1PINT = 1;
      EvaRegs.EVAIMRA.bit.T1CINT = 1;
```

```
      EvaRegs.EVAIMRB.bit.T2PINT = 1;
      PieCtrlRegs.PIEIER1.bit.INTx1 =     1;          // Enable
      PDPINTA     in PIE:     Group 1     interrupt   1
      //PieCtrlRegs.PIEIER1.bit.INTx6     =     1;    // Enable
      ADC   interrupt   in PIE:     Group 1     interrupt   6
      PieCtrlRegs.PIEIER2.bit.INTx6 =     1;          // Enable
      T1UFINT     in PIE:     Group 2     interrupt   6.    WYK
      20090525
      PieCtrlRegs.PIEIER2.bit.INTx4 =     1;          // Enable
      T1PINT in   PIE: Group 2 interrupt 4.      WYK 20091013
      PieCtrlRegs.PIEIER2.bit.INTx5 =     1;    // Enable   T1CINT
in    PIE: Group 2 interrupt 5.      WYK 20091207
      PieCtrlRegs.PIEIER3.bit.INTx5 =     1;          // Enable
      CAPINT1     in PIE:     Group 3     interrupt   5
      PieCtrlRegs.PIEIER3.bit.INTx6 =     1;          // Enable
      CAPINT2     in PIE:     Group 3     interrupt   6
      PieCtrlRegs.PIEIER5.bit.INTx5 =     1;                //
Enable      CAPINT4     in PIE:     Group 5     interrupt   5
      PieCtrlRegs.PIEIER5.bit.INTx6 =     1;          // Enable
      CAPINT5     in PIE:     Group 5     interrupt   6
      PieCtrlRegs.PIEIER5.bit.INTx7 =     1;          // Enable
      CAPINT6     in PIE:     Group 5     interrupt   7
      //PieCtrlRegs.PIEIER3.bit.INTx1     =     1;          //
Enable      T2PINT in   PIE: Group 3 interrupt 1
      //PieCtrlRegs.PIEIER6.bit.INTx1     =     1;;         //SPI
      interrupt   test
      IER   |= M_INT1|M_INT2|M_INT3|M_INT5;                //
Enable      CPU   Interrupts 1,2,3,5,6
      //IER |= M_INT1|M_INT2|M_INT3|M_INT6;          // Enable
      CPU   Interrupts 1,2,3,6
      EINT;
      AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1; // clear interrupt flag
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;         // Acknowledge
interrupt to      PIE
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP2;         // Acknowledge
interrupt to      PIE
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;         // Acknowledge
interrupt to      PIE
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP5;         // Acknowledge
interrupt to      PIE
      /*Setup     timer 2     for   capture     port operation */
      //Timer     period -1   =     switching   period (to match the
timer period      as period   of
      //a   up-down     is 2* period,     period of   continuous up
      is period   -1).  This allow for easier   decoding of capture
      results
      EvaRegs.T2PR = 65534;
      EvaRegs.T2CNT     =     0;
      EvaRegs.T2CON.all =     0x0000;
      EvaRegs.T2CON.bit.TCLKS10     =     0x00;
      EvaRegs.T2CON.bit.SET1PR = 0x00;
      EvaRegs.T2CON.bit.TMODE =     2;
      //Continuous up   count mode
      EvaRegs.T2CON.bit.TPS   =     7;
      //prescalar 1/128
      EvaRegs.T2CON.bit.T2SWT1 = 0x00;          //Use own   TENABLE
      bit
      EvaRegs.T2CON.bit.TENABLE     =     1;                //
enable      timer 2
      /*Capture   port setting */
            GpioMuxRegs.GPAMUX.bit.CAP1Q1_GPIOA8 = 1;
```

```
            GpioMuxRegs.GPAMUX.bit.CAP2Q2_GPIOA9 = 1;
      EvaRegs.CAPCONA.all     =     0x0000;
      EvaRegs.CAPCONA.bit.CAPRES = 0;                        //Reset
      capture     unit
      EvaRegs.CAPCONA.bit.CAP12EN   =     1;                //Enable
capture port      1     and   2
      EvaRegs.CAPCONA.bit.CAP1EDGE = 0x03;            //Detect both
      edge in     capture     port 1
      EvaRegs.CAPCONA.bit.CAP2EDGE = 0x03;            //Detect both
      edge in     capture     port 2
      //Make the capture unit believe     it already has 1 result.
      This causes the   capture     port to     generate an
      interrupt   when
      //it set another capture come along.      (Capture port
      only generate     interrupt   at second   capture)
      EvaRegs.CAPFIFOA.bit.CAP1FIFO =     1;
      EvaRegs.CAPFIFOA.bit.CAP2FIFO =     1;
      /*Setting   up capture port   1     interrupt   */
      EvaRegs.EVAIMRC.all     =     0;
      //Disable   all   capture     port interrupt
      EvaRegs.EVAIFRC.all     =     0;
      //Clearing interrupt flag     for   capture     port
      EvaRegs.EVAIMRC.bit.CAP1INT   =     1;          //Enabling
capture port      1     interrupt
      EvaRegs.EVAIMRC.bit.CAP2INT   =     1;          //Enabling
capture port      2     interrupt
      /*Setup     timer 3     for   capture     port 4 operation on
      EVB*/
      //Timer     period -1   =     switching   period (to match the
timer period      as period   of
      //a   up-down     is 2* period,     period of   continuous up
      is period   -1).  This allow for easier   decoding of capture
      results
      EvbRegs.T3PR = 65536 - 1;
      EvbRegs.T3CNT     =     0;
      EvbRegs.T3CON.all =     0x0000;
      EvbRegs.T3CON.bit.TCLKS10     =     0x00;
      EvbRegs.T3CON.bit.TMODE =     2;          //Continuous up
      count mode
      EvbRegs.T3CON.bit.TPS   =     7;
      //prescalar 1/128
      EvbRegs.T3CON.bit.TENABLE     =     1;           //   enable
timer3
            /*Setup     timer 4     for   capture     port 4
operation on      EVB*/
      //Timer     period -1   =     switching   period (to match the
timer period      as period   of
      //a   up-down     is 2* period,     period of   continuous up
      is period   -1).  This allow for easier   decoding of capture
      results
      EvbRegs.T4PR = 65535 - 1;
      EvbRegs.T4CNT     =     0;
      EvbRegs.T4CON.all =     0x0000;
      EvbRegs.T4CON.bit.TMODE       =     2;          //Continuous
up    count mode
      EvbRegs.T4CON.bit.TPS               =     7;
      //prescalar 1/128
      EvbRegs.T4CON.bit.T4SWT3      =     1;           //   enable
timer4
      EvbRegs.T4CON.bit.TENABLE     =     1;           //   enable
timer4
```

```
      //Capture   port B Setting
      EvbRegs.CAPCONB.all     =     0x0000;
      EvbRegs.CAPCONB.bit.CAP45EN         =     1;
      // Enable   captures 4 and 5
      EvbRegs.CAPCONB.bit.CAP6EN          =     0;
      // Enable   capture     6
      EvbRegs.CAPCONB.bit.CAPRES          =     0;
            // Release from   reset
      EvbRegs.CAPCONB.bit.CAP4EDGE  =     0x03;             //
fallin      edge on     capture     4
      EvbRegs.CAPCONB.bit.CAP5EDGE  =     0x03;             // both
      edge on     capture     5
      EvbRegs.CAPCONB.bit.CAP6EDGE  =     0x02;             //
falling edge      on capture 6
      EvbRegs.CAPCONB.bit.CAP45TSEL =     1;                      //
Capture 4 and 5 select  timer 3
      EvbRegs.CAPCONB.bit.CAP6TSEL  =     1;                      //
Capture 6 select  timer 3
       /*Setting up     capture     port 4,5,6 interrupt */
      EvbRegs.EVBIMRC.all     =     0;                      //
Disable all capture port      interrupt
      EvbRegs.EVBIFRC.all     =     0;                      //
Clearing    interrupt   flag for capture port
      EvbRegs.EVBIMRC.bit.CAP4INT   =     1;          // Enabling
      capture     port 4 interrupt
      EvbRegs.EVBIMRC.bit.CAP5INT   =     1;          // Enabling
      capture     port 5 interrupt
      EvbRegs.EVBIMRC.bit.CAP6INT   =     0;          // Enabling
      capture     port 6 interrupt
      //Make the capture unit believe     it already has 1 result.
      This causes the   capture     port to     generate an
      interrupt   when
      //it set another capture come along.      (Capture port
      only generate     interrupt   at second   capture)
      EvbRegs.CAPFIFOB.bit.CAP4FIFO =     1;
      EvbRegs.CAPFIFOB.bit.CAP5FIFO =     1;
      EvbRegs.CAPFIFOB.bit.CAP6FIFO =     1;
      //If slave and carrier is     delayed     start the   timer 1
      with a preloaded number
      if((master_slave_mode   == 0)&&(slave_delay     == 1)){
            EvaRegs.T1CNT     =     period>>1;
      }
  /* Setup and load     T1CON to start operation */
      EvaRegs.T1CON.bit.TMODE =     1;          // continous up/down
count mode
      EvaRegs.T1CON.bit.TPS   =     0;                // input clock
prescaler
      EvaRegs.T1CON.bit.TECMPR = 1;       // enable   time compare
      EvaRegs.T1CON.bit.TCLD10 = 1;             // timer compare
reload      at underflow and period
      EvaRegs.T1CON.bit.TENABLE     =     1;          // enable
      timer
      // Initialise     state machine
      vsi_state.first   =     1;
      vsi_state.f =     &st_vsi_init;
                        GrabShow();
                        GrabClear();
                        GrabStart();
                        GrabRun();
  }   /* end vsi_init   */
```

```
   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
   /**
   This function   is called   from the main     background loop
       once every millisecond.
   It performs     all   low   speed tasks associated with   running
       the   core interrupt
   process, including:
   -   checking for faults
   -   calling     the   VSI   state functions
   -   calling     internal analog   scaling     functions
   \author   A.McIver
   \par History:
   \li 13/10/07 AM -     derived     from 25kVA:vsi:vsi.c
   */
   void vsi_state_machine(void)
   {
       SS_DO(vsi_state);
       if (adc_int.flag_rms != 0) // rms   flag synched to   VSI
       fundamental
       {
             adc_int.flag_rms = 0;
             scale_adc_rms();
       }
       else if     (adc_int.flag_dc !=     0) // ADC_DC_TIME flag
       {
             adc_int.flag_dc   =     0;
             scale_adc_dc();
       }
       else if     (adc_int.flag_cal != 0)
       {
             adc_int.flag_cal = 0;
             calibrate_adc();
       }
   }   /* end vsi_state_machine */
   /*
=========================================================================
=====
   __Exported_VSI_Functions()
   =========================================================================
========== */
   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
   /**
   This function   switches the VSI from   the   stopped     state to
a running state.
   \author   A.McIver
   \par History:
   \li 13/10/07 AM -     derived     from 25kVA:vsi:vsi.c
   */
   void vsi_enable(void)
   {
       if (detected_faults     == 0)
             is_switching = 1;
   }   /* end vsi_enable */
   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
   /**
   This function   switches the VSI from   the   running     state to
a stop      state.
```

```
  The ramp down   process     has   the   side effect of resetting
the reference to  zero.
  \author   A.McIver
  \par History:
  \li 13/10/07 AM -     derived     from 25kVA:vsi:vsi.c
  */
  void vsi_disable(void)
  {
      is_switching = 0;
  }   /* end vsi_disable */
  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
  /**
  This function   sets the target   output modulation depth.
  The target is   passed in   tenths of   a     percent, so a
      value of 1000     corresponds to
  100% modulation depth.
  \author   A.McIver
  \par History:
  \li 24/04/09 AM -     initial     creation
  \param[in] m Target   output modulation depth
  */
  void vsi_set_mod(Uint16     m)
  {
      int32
            temp;
      if (m >     MOD_DEPTH_MAX)
      {
            m     =     MOD_DEPTH_MAX;
      }
      temp = (((int32)m) <<   MOD_SHIFT) / ((int32)MOD_DEPTH_MAX);
      mod_ref     =     (int16)temp;
  }   /* end vsi_set_mod */
  void vsi_set_mod_immediate(Uint16 m)
  {
      int32
            temp;
      if (m >     MOD_DEPTH_MAX)
      {
            m     =     MOD_DEPTH_MAX;
      }
      temp = (((int32)m) <<   MOD_SHIFT) / ((int32)MOD_DEPTH_MAX);
      mod_ref     =     (int16)temp;
      mod_targ = mod_ref;
      //Shift     left of     1     introduce   to deal     with the
fact  that mod_targ     200% at     full range
      I_ref_Peak_AB     =
      ((long)I_NOM*((long)mod_targ<<1))>>MOD_SHIFT;
  }   /* end vsi_set_mod */
  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
  /**
  This function   returns     the   target output     modulation
depth.
  \author   A.McIver
  \par History:
  \li 24/04/09 AM -     initial     creation
  \returns The VSI target     output modulation depth in tenths   of
a percent
  */
  Uint16 vsi_get_mod(void)
```

```
  {
      int32
            temp;
      temp = ((int32)mod_ref * (int32)MOD_DEPTH_MAX   +
      (1L<<(MOD_SHIFT-1))     )
                        >> MOD_SHIFT;
      return (Uint16)temp;
  }   /* end vsi_get_mod */
  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
  /**
  Set the   target output     frequency   in Hz.
  \author   A.McIver
  \par History:
  \li 12/10/07 AM -     initial     creation
  \li 04/03/08 AM -     added return of   new   frequency
  \returns The new frequency in     Hz
  \param[in] f Target   fundamental frequency   in Hz
  */
  double vsi_set_freq(double f)
  {
      if (f == 0.0||refMode   == DC_REF) //     DC output
      {
            phase_step = 0L;
            phase =     32768uL*65536uL;
      }
      else
      {
            if (f >     F_FREQ_MAX)
            {
                  f     =     F_FREQ_MAX;
            }
            else if     (f < F_FREQ_MIN)
            {
                  f     =     F_FREQ_MIN;
            }
            phase_step = (Uint32)(PHASE_STEP_SC *     f     +
      0.5); // atomic   load
      }
      Ref_freq_float = f;
      return (double)phase_step/PHASE_STEP_SC;
  }   /* end vsi_set_freq     */
  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
  /**
  This function   returns     the   VSI   fundamental frequency.
  \author   A.McIver
  \par History:
  \li 19/06/08 AM -     initial     creation
  \returns The VSI fundamental frequency
  */
  double vsi_get_freq(void)
  {
      return (double)phase_step/PHASE_STEP_SC;
  }   /* end vsi_get_freq     */
  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
  /**
  This function   returns     the   status of   the   VSI   output
system. It  returns
  -   stopped     or running
```

```
  -   fault code
  -   ramping     or settled
  \author   A.McIver
  \par History:
  \li 13/10/07 AM -     derived     from 25kVA:vsi:vsi.c
  \retval   VSI_RUNNING VSI   system switching with   output
  \retval   VSI_SETTLED Output has reached target
  \retval   VSI_FAULT   VSI   system has detected     a     fault
  */
  Uint16 vsi_get_status(void)
  {
      return vsi_status;
  }   /* end vsi_get_status   */
  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
  /**
  This function   returns     the   fault word of     the   VSI
      module.
  \author   A.McIver
  \par History:
  \li 04/03/08 AM -     initial     creation
  \returns The present fault word
  */
  /// Report what faults are present in   the   VSI
  Uint16 vsi_get_faults(void)
  {
      return detected_faults;
  }   /* end vsi_get_faults   */
  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
  /* void   vsi_clear_faults(void)
  Parameters:     none
  Returns: nothing
  Description: Clear the detected   faults.
  Notes:
  History:
      13/10/05 AM -     initial     creation
  \li 28/04/08 AM -     added event reporting
  */
  void vsi_clear_faults(void)
  {
      Uint16
            i;
      if (detected_faults     &     FAULT_VSI_PDPINT)
      {
            for   (i=0; i<100; i++)
                  i++; //     delay for   fault to clear
            EvaRegs.COMCONA.all     =     0;
            EvaRegs.COMCONA.all     =     0xAA00;
      }
      detected_faults   =     0;
  }   /* end vsi_clear_faults */
  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
  /* Uint16 vsi_get_vdc(void)
  Parameters:     none
  Returns: DC     bus   voltage     in Volts
  Description: Retrieves filtered   and   scaled Vh   measurements.
  Notes:
  History:
      13/10/05 AM -     initial     creation
```

```
   */
   /*
   Function is     commented   out   until scaling     is set up
    Uint16   vsi_get_vdc(void)
   {
       return (Uint16)(adc_int.vdc.real + 0.5);
   }*/ /* end vsi_get_vdc */
   /*
==========================================================================
===== */
   /* Interrupt Routines */
   /*
==========================================================================
===== */
   #ifndef   BUILD_RAM
   #pragma   CODE_SECTION(isr_adc,   "ramfuncs");
   #endif
   interrupt void isr_adc(void)
   {
       EvaRegs.EVAIFRA.all     =     BIT7;                    // clear
interrupt flag
       AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1;       // clear interrupt
flag
       PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;   // Acknowledge
interrupt to      PIE
       adc_ready_flag = 1;
   }
   /**
   \fn interrupt   void isr_adc(void)
   \brief Updates VSI and stores     ADC   results
   This interrupt is     triggered   by the completion of the
internal    ADC   conversions.
   It then:
   -   stores the internal     ADC   results
   -   applies     the   internal 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     on
internal    ADC   quantities
   -   updates     phase angle
   -   calculates switching times
   -   loads compare     registers   with switching times
   -   sets up     analogs     for   next interrupt
   \author   A.McIver
   \par History:
   \li 12/10/07 AM -     initial     creation
   */
   #ifndef   BUILD_RAM
   #pragma   CODE_SECTION(isr_T1CINT, "ramfuncs");
   #endif
   interrupt void isr_T1CINT(void){
       loop_no++;
       timer1_cmp_flag   =     1;
       EvaRegs.EVAIFRA.all     =     BIT8;
       //Clear     interrupt   flag
       PieCtrlRegs.PIEACK.all = PIEACK_GROUP2;   // Acknowledge
interrupt to      PIE
   }
   #ifndef   BUILD_RAM
```

```
  #pragma   CODE_SECTION(isr_pwm,   "ramfuncs");
  #endif
  interrupt void isr_pwm(void)
  {
      //Find out the direction which the timer is     going, used to
update      timer 1     compare     to
      //get ADC   to trigger at     right point.
      int   timer1_dir = EvaRegs.GPTCONA.bit.T1STAT;
      SET_TP10() ;
      zero_crossing_count     ++ ;
      if(zero_crossing_count ==     2)
      {
      zero_crossing_count     =     0     ;
      zero_crossing_clock     =     EvaRegs.T2CNT;
      if((zero_crossing_clock >     32000L)     &&
((zero_crossing_capture_A - 1000L)  <     0L)   )
            zero_crossing_capture_A =     zero_crossing_capture_A +
      65535L ;
      else if((zero_crossing_capture_A > 32000L) &&
      ((zero_crossing_clock   -     1000L) < 0L) )
            zero_crossing_clock     =     zero_crossing_clock     +
      65535L ;
      capture_difference_A = zero_crossing_clock -
zero_crossing_capture_A -     234L        ;
      band_offset_A     =     ((((long)((capture_difference_A) *
1118L)      >>7) * hyst_band)>>12);
      //band_offset_A   =     ((((long)((capture_difference_A     *
      hyst_band_C) ) >>8)     *     hyst_band)>>12);
      if (band_offset_A >     200L)
            band_offset_A     =     200L ;
      else if     (band_offset_A < -200L)
            band_offset_A     =     -200L ;
      if((zero_crossing_clock >     32000L)     &&
((zero_crossing_capture_B - 1000L)  <     0L)   )
            zero_crossing_capture_B =     zero_crossing_capture_B +
      65535L ;
      else if((zero_crossing_capture_B > 32000L) &&
      ((zero_crossing_clock   -     1000L) < 0L) )
            zero_crossing_clock     =     zero_crossing_clock     +
      65535L ;
      capture_difference_B = zero_crossing_clock -
zero_crossing_capture_B -234L ;
      band_offset_B     =     ((((long)((capture_difference_B) *
1118L)      >>7) * hyst_band)>>12);
      if (band_offset_B >     200L)
            band_offset_B     =     200L ;
      else if     (band_offset_B < -200L)
            band_offset_B     =     -200L ;
  }
            // Read     phase angle and   reading     sin   table
      for   generating three-phase sine waves
      SIN_TABLE_READ(index_sin,     sin_val);
      SIN_TABLE_READ(index_sin_4PI_on_3, sin_4PI_on_3_val);
      SIN_TABLE_READ(index_sin_2PI_on_3, sin_2PI_on_3_val);
      // update   phase angle and   reading     sin   table for
      generating three-phase sine waves
      SIN_TABLE_UPDATE(phase_sin,   index_sin);
      SIN_TABLE_UPDATE(phase_sin_4PI_on_3, index_sin_4PI_on_3);
      SIN_TABLE_UPDATE(phase_sin_2PI_on_3, index_sin_2PI_on_3);
      I_ref_Peak_AB_prev = I_ref_Peak_AB ;
```

```
      I_ref_A     =     ((((long)I_ref_Peak_AB<<1)*(long)sin_val) >>
16);
      I_ref_B     =
      ((((long)I_ref_Peak_AB<<1)*(long)sin_4PI_on_3_val) >> 16);
      I_ref_C     =
      ((((long)I_ref_Peak_AB<<1)*(long)sin_2PI_on_3_val) >> 16);
      //VSI Hysteresis Controller Arrangements
      if(is_switching   == TRUE){
            switch(refMode){
                  case THREE_PHASE_OL:
                        t_A   =     ((long)mod_targ*((long)sin_val))
>>    (MOD_SHIFT+1);
                        t_B   =
      ((long)mod_targ*((long)sin_2PI_on_3_val)) >> (MOD_SHIFT+1);
                        t_C   =     -t_A - t_B;
                        break;
                  case SINGLE_PHASE_HS:
                              Vband_prev_A = Vband_A ;
                              Vband_A     =     (long)((4096L     *
      (10000L     -     (((long)(duty_cycle_A)) *
      ((long)(duty_cycle_A))))/10000L) * (long)hyst_band)>>12     ;
                              Vband_A     =     Vband_A     +
      band_offset_A     ;
                        if(mod_depth ==   0)
                        {

      dac_fast_write(DAC_MODULE_D2,DAC_WRn_UPDn,DAC_ADDR_A,
      hyst_band   );
                        }
                        else
                        {

      dac_fast_write(DAC_MODULE_D2,DAC_WRn_UPDn,DAC_ADDR_A, Vband_A
      );
                        }

      DAC1_FAST_WRITE(DAC_WRn_UPDn,DAC_ADDR_D,I_ref_A +2047);
                              break;
                        case TPH_HCC_VB_2CNT:
                              Vband_prev_A = Vband_A ;
                              duty_cycle_A = duty_cycle_A   -
      V_L_A ;
                              Vband_A     =     (long)((4096L     *
      (10000L     -     (((long)(duty_cycle_A)) *
      ((long)(duty_cycle_A))))/10000L) * (long)hyst_band)>>12     ;
                              Vband_prev_B = Vband_B ;
                              duty_cycle_B = duty_cycle_B   -
      V_L_B ;
                              Vband_B     =     (long)((4096L     *
      (10000L     -     (((long)(duty_cycle_B)) *
      ((long)(duty_cycle_B))))/10000L) * (long)hyst_band)>>12     ;
                        Vband_A     =     Vband_A     +
      band_offset_A     ;
                        Vband_B     =     Vband_B     +
      band_offset_B     ;
                        if(mod_depth ==   0)
                        {

      dac_fast_write(DAC_MODULE_D2,DAC_WRn_UPDn,DAC_ADDR_A,
      hyst_band   );
```

```
      dac_fast_write(DAC_MODULE_D2,DAC_WRn_UPDn,DAC_ADDR_B,
      hyst_band   );

      //dac_fast_write(DAC_MODULE_D2,DAC_WRn_UPDn,DAC_ADDR_D,
      hyst_band   );
                        }
                        else
                        {

      dac_fast_write(DAC_MODULE_D2,DAC_WRn_UPDn,DAC_ADDR_A, Vband_A
      );

      dac_fast_write(DAC_MODULE_D2,DAC_WRn_UPDn,DAC_ADDR_B, Vband_B
      );
                        }
                        CLEAR_TP10();
                        /*Saturation Detection and Compensation
      for   this region (do   not   delete)*/
                        if(duty_cycle_A > 95L)
                              duty_cycle_A = 95L ;
                        if(duty_cycle_A < -95L)
                              duty_cycle_A = -95L ;
                        if(duty_cycle_B > 95L)
                              duty_cycle_B = 95L ;
                        if(duty_cycle_B < -95L)
                              duty_cycle_B = -95L ;
                        if(duty_cycle_C > 95L)
                              duty_cycle_C = 95L ;
                        if(duty_cycle_C < -95L)
                              duty_cycle_C = -95L ;
                        duty_cycle_A = ((duty_cycle_A) +
(duty_cycle_A_prev))/2 ;
                        duty_cycle_B = ((duty_cycle_B) +
(duty_cycle_B_prev))/2 ;
                        MDepth_A = (duty_cycle_A >>   1) + 50L ;
                        MDepth_B = (duty_cycle_B >>   1) + 50L ;

      dac_fast_write(DAC_MODULE_D2,DAC_WRn_UPDn,DAC_ADDR_C,
      bus_voltage);
                        duty_cycle_C_virt =     (-1L * (duty_cycle_A
+ duty_cycle_B))  ;
                        duty_cycle_C_virt_prev = duty_cycle_C_virt
      ;
                        duty_cycle_C_virt =     ((duty_cycle_C_virt)
      +     (duty_cycle_C_virt_prev))/ 2 ;
                        duty_cycle_C = duty_cycle_C_virt ;
                        command_C_virt = ((duty_cycle_C_virt ) *
140)  ; // Multiply by scaling factor
                        t_C   =     command_C_virt ;

      DAC1_FAST_WRITE(DAC_WRn_UPDn,DAC_ADDR_A,duty_cycle_C+2047);

      DAC1_FAST_WRITE(DAC_WRn_UPDn,DAC_ADDR_D,I_ref_A +2047);

      DAC1_FAST_WRITE(DAC_WRn_UPDn,DAC_ADDR_C,I_ref_B+2047);
                        break;
                  case TPH_HCC_VB_3CNT:
                              Vband_A     =     (long)((4096L     *
      (10000L     -     (((long)(duty_cycle_A)) *
      ((long)(duty_cycle_A))))/10000L) * (long)hyst_band)>>12     ;
```

```
                        Vband_prev_A = Vband_A ;
                        Vband_prev_B = Vband_B ;
                        Vband_B     =     (long)((4096L     *
(10000L     -     (((long)(duty_cycle_B   +
average_offset)) * ((long)(duty_cycle_B   +
average_offset))))/10000L) * (long)hyst_band)>>12     ;
                        Vband_C     =     (long)((4096L     *
(10000L     -     (((long)(duty_cycle_C   +
average_offset)) * ((long)(duty_cycle_C   +
average_offset))))/10000L) * (long)hyst_band)>>12     ;
                        Vband_A     =     Vband_A     +
band_offset_A     ;
                        Vband_B     =     Vband_B     +
band_offset_B     ;
                        Vband_C     =     Vband_C     +
band_offset_C     ;
                  /*Saturation Detection and Compensation
for   this region (do   not   delete)*/
                  if(duty_cycle_A > 95L)
                        duty_cycle_A = 95L ;
                  if(duty_cycle_A < -95L)
                        duty_cycle_A = -95L ;
                  if(duty_cycle_B > 95L)
                        duty_cycle_B = 95L ;
                  if(duty_cycle_B < -95L)
                        duty_cycle_B = -95L ;
                  if(duty_cycle_C > 95L)
                        duty_cycle_C = 95L ;
                  if(duty_cycle_C < -95L)
                        duty_cycle_C = -95L ;
                  if(mod_depth ==   0)
                  {

dac_fast_write(DAC_MODULE_D2,DAC_WRn_UPDn,DAC_ADDR_A,
hyst_band   );

dac_fast_write(DAC_MODULE_D2,DAC_WRn_UPDn,DAC_ADDR_B,
hyst_band   );

dac_fast_write(DAC_MODULE_D2,DAC_WRn_UPDn,DAC_ADDR_D,
hyst_band   );
                  }
                  else
                  {

dac_fast_write(DAC_MODULE_D2,DAC_WRn_UPDn,DAC_ADDR_A, Vband_A
);

dac_fast_write(DAC_MODULE_D2,DAC_WRn_UPDn,DAC_ADDR_B,
Vband_B);

dac_fast_write(DAC_MODULE_D2,DAC_WRn_UPDn,DAC_ADDR_D,
Vband_C);
                  }

dac_fast_write(DAC_MODULE_D2,DAC_WRn_UPDn,DAC_ADDR_C,
bus_voltage);

DAC1_FAST_WRITE(DAC_WRn_UPDn,DAC_ADDR_D,I_ref_A +2047);

DAC1_FAST_WRITE(DAC_WRn_UPDn,DAC_ADDR_C,I_ref_B +2047);
```

```
      DAC1_FAST_WRITE(DAC_WRn_UPDn,DAC_ADDR_A,I_ref_C +2047);
                        break;
                  case TRD_HARM:
                              Vband_A = (long)((4096L * (10000L -
(((long)(duty_cycle_A + V3rd_count)) * ((long)(duty_cycle_A +
V3rd_count))))/10000L) * (long)hyst_band)>>12 ;
                              Vband_B = (long)((4096L * (10000L -
(((long)(duty_cycle_B)) * ((long)(duty_cycle_B))))/10000L) *
(long)hyst_band)>>12 ;
                              Vband_A = Vband_A + band_offset_A ;
                              Vband_B = Vband_B + band_offset_B ;
                        if(mod_depth == 0)
                        {

      dac_fast_write(DAC_MODULE_D2,DAC_WRn_UPDn,DAC_ADDR_A,
hyst_band );

      dac_fast_write(DAC_MODULE_D2,DAC_WRn_UPDn,DAC_ADDR_B,
hyst_band );
                        }
                        else
                        {

      dac_fast_write(DAC_MODULE_D2,DAC_WRn_UPDn,DAC_ADDR_A, Vband_A
);

      dac_fast_write(DAC_MODULE_D2,DAC_WRn_UPDn,DAC_ADDR_B, Vband_B
);
                        }
      /*                duty_cycle_C_virt = (-1L * (duty_cycle_A +
duty_cycle_B)) ;
                        //duty_cycle_C_virt = ((duty_cycle_C_virt *
256L) + (duty_cycle_C_virt_prev * 768L)) >> 10 ;
                        duty_cycle_C_virt_prev_prev =
duty_cycle_C_virt_prev  ;
                        duty_cycle_C_virt_prev = duty_cycle_C_virt
      ;
                        if(stepping)
                        command_C_virt = ((-duty_cycle_C_virt ) *
140) ;
                        else
                        command_C_virt = ((duty_cycle_C_virt ) *
140) ;
                        t_C = command_C_virt ;*/
            // Read phase angle and reading sin table for generating
sin value for phase A and B
      SIN_TABLE_READ(index_sin, sin_val);
      SIN_TABLE_READ(index_sin_4PI_on_3, sin_4PI_on_3_val);
      SIN_TABLE_READ(index_sin_2PI_on_3, sin_2PI_on_3_val);
      // update phase angle and reading sin table for generating sin
value for phase A and B
      SIN_TABLE_UPDATE(phase_sin, index_sin);
      SIN_TABLE_UPDATE(phase_sin_4PI_on_3, index_sin_4PI_on_3);
      SIN_TABLE_UPDATE(phase_sin_2PI_on_3, index_sin_2PI_on_3);
      I_ref_Peak_AB_prev = I_ref_Peak_AB ;
      I_ref_A = ((((long)I_ref_Peak_AB<<1)*(long)sin_val) >> 16);
      I_ref_B = ((((long)I_ref_Peak_AB<<1)*(long)sin_4PI_on_3_val)
>> 16);
      I_ref_C = ((((long)I_ref_Peak_AB<<1)*(long)sin_2PI_on_3_val)
>> 16);
```

```
                        duty_cycle_A_3rd = duty_cycle_A + Voff;
                        duty_cycle_B_3rd = duty_cycle_A + Voff;
                        duty_cycle_C_3rd = duty_cycle_A + Voff;
                        if (duty_cycle_A_3rd > duty_cycle_B_3rd)
                        {
                              if (duty_cycle_A_3rd >
duty_cycle_C_3rd)
                              {
                                    if (duty_cycle_B_3rd >
duty_cycle_C_3rd)
                                          Voff =
duty_cycle_B_3rd>>1;
                                    else
                                          Voff =
duty_cycle_C_3rd>>1;
                              }
                              else
                              {
                                    Voff = duty_cycle_A_3rd>>1;
                              }
                        }
                        else
                        {
                              if (duty_cycle_B_3rd >
duty_cycle_C_3rd)
                              {
                                    if (duty_cycle_A_3rd >
duty_cycle_C_3rd)
                                          Voff =
duty_cycle_A_3rd>>1;
                                    else
                                          Voff =
duty_cycle_C_3rd>>1;
                              }
                              else
                              {
                                    Voff = duty_cycle_B_3rd>>1;
                              }
                        }
                        Voff_scaled = ((average_offset * Voff) >> 4)
;//+ V3rd_offset ;
                        //Voff_scaled =  Voff * average_offset ;//+
V3rd_offset ;
                        Voff_int = Voff * 4 ;
                        //Voff=0;Voff_scaled = 0 ;
                        duty_cycle_C_virt = (-1L * (duty_cycle_A +
duty_cycle_B)) ;
                        duty_cycle_C_virt_prev_prev =
duty_cycle_C_virt_prev  ;
                        duty_cycle_C_virt_prev = duty_cycle_C_virt
      ;
                        command_C_virt = ((duty_cycle_C_virt) +
((Voff * V3rd_offset)/10 )) * 140 ;
                        t_C = command_C_virt ;

      DAC1_FAST_WRITE(DAC_WRn_UPDn,DAC_ADDR_D,I_ref_A +2047);

      DAC1_FAST_WRITE(DAC_WRn_UPDn,DAC_ADDR_C,I_ref_B +2047);

      DAC1_FAST_WRITE(DAC_WRn_UPDn,DAC_ADDR_B,(duty_cycle_A * 10)
+2047);
```

```
                        DAC1_FAST_WRITE(DAC_WRn_UPDn,DAC_ADDR_A,
((Voff_scaled)) +2047);

      dac_fast_write(DAC_MODULE_D2,DAC_WRn_UPDn,DAC_ADDR_C,
bus_voltage);
                        break;
            }
      }
  /*
=============================================================================
=====
  isr_pwm_1st_SPI()
  =============================================================================
========== */
      I_res_A     =     (AdcRegs.ADCRESULT4>>4);
      I_res_A     =     (((int16)((((int32)(I_res_A- ADC_OFFSET   -
      cal_offsetA))*(int32)cal_gainA)     >> 14))     *     -1)-
BOARD_OFFSET_A ;
      I_res_B     =     (AdcRegs.ADCRESULT8>>4);
      I_res_B     =     (((int16)((((int32)(I_res_B- ADC_OFFSET   -
      cal_offsetA))*(int32)cal_gainA)     >> 14))     *     -1)-
BOARD_OFFSET_B ;
      I_res_C     =     (AdcRegs.ADCRESULT0>>4);
      I_res_C     =     ((int16)((((int32)(I_res_C-   ADC_OFFSET -
cal_offsetA))*(int32)cal_gainA) >>  14)) * -1   ;
      Vdc   =     (AdcRegs.ADCRESULT11>>4);
      Vdc   =     ((int16)((((int32)(Vdc- ADC_OFFSET_VDC -
cal_offsetA))*(int32)cal_gainA) >>  14));
      one_on_vbus =     (1<<13)/Vdc;
      Vdc   =     ((double)ADC_VDC_SC)*((double)Vdc) ;
      //Offset elimination for I_res_A
      if(is_switching   == 0){
            if(prev_index_sin >     index_sin){
                        I_res_DC_offset   =
      I_res_cycle_total/I_res_cycle_count;
                        I_res_cycle_total =     0;
                        I_res_cycle_count =     0;
            }
            I_res_cycle_total += I_res_A;
            I_res_cycle_count += 1;
      }
      if(I_res_DC_offset > 20 ){
            I_res_DC_offset   =     20;
      }
      if(I_res_DC_offset < -20){
            I_res_DC_offset   =     -20;
      }
      I_res_A     =     I_res_A     -     I_res_DC_offset;
      I_res_B     =     I_res_B     -     I_res_DC_offset;
      I_res_C     =     I_res_C     -     I_res_DC_offset;
      // calibration from     references
      adc_int.yHA.dc_sum +=   (Uint32)(AdcRegs.ADCRESULT12>>4);
      adc_int.yLA.dc_sum +=   (Uint32)(AdcRegs.ADCRESULT14>>4);
      adc_int.yHB.dc_sum +=   (Uint32)(AdcRegs.ADCRESULT13>>4);
      adc_int.yLB.dc_sum +=   (Uint32)(AdcRegs.ADCRESULT15>>4);
      adc_int.count_cal++;
      if (adc_int.count_cal   >     ADC_COUNT_CAL)
      {
            adc_int.count_cal =     0;
            adc_int.yHA.dc_sum_bak = adc_int.yHA.dc_sum;
            adc_int.yLA.dc_sum_bak = adc_int.yLA.dc_sum;
```

```
            adc_int.yHB.dc_sum_bak = adc_int.yHB.dc_sum;
            adc_int.yLB.dc_sum_bak = adc_int.yLB.dc_sum;
            adc_int.yHA.dc_sum = 0;
                  adc_int.yLA.dc_sum = 0;
                  adc_int.yHB.dc_sum = 0;
                  adc_int.yLB.dc_sum = 0;
                  adc_int.flag_cal = 1;
      }
      if(I_res_A <0){
            rect_i_A = I_res_A * -1;
      }
      else{
            rect_i_A = I_res_A;
      }
  /*
=========================================================================
=====
  isr_pwm_vsi()
  =========================================================================
========== */
      period_load       =     (Uint16) period_2<<0;   //+
      carrier_adjust);
      period_load_2           =     period_load>>1;
      MAX_TIME          =     period_load_2     -     6;
      EvaRegs.T1PR      =     period_load;
            //EvaRegs.T2PR    =     period_load*2-1;
      EvaRegs.T1CMPR    =     period_load_2;
  /* clamp switch times for   pulse deletion and saturation */
      // A phase
      if (t_A     >     MAX_TIME)
      {
            EvaRegs.CMPR1     =     0;
            //EvaRegs.T1CMPR = 0 ;
      }
      else if     (t_A < -MAX_TIME)
      {
            if (!V_Asat && (timer1_dir == 1)){
                  EvaRegs.CMPR1     =     (period_load_2<<1) - 1;
      }
            else{
                  EvaRegs.CMPR1     =     (period_load_2<<1);
            }
            V_Asat = 1;
      }
      else
      {
            if (V_Asat &&     (timer1_dir == 1)){
                  EvaRegs.CMPR1     =     (period_load_2<<1);
            }
            else{
                  EvaRegs.CMPR1     =     period_load_2     -
      t_A;
            }
            V_Asat = 0;
      }
  // B phase
      if (t_B     >     MAX_TIME)
      {
            EvaRegs.CMPR2     =     0;
      }
      else if     (t_B < -MAX_TIME)
```

```
      {
            if (!V_Bsat && (timer1_dir == 1)){
                  EvaRegs.CMPR2     =     (period_load_2<<1) - 1;
            }
            else{
                  EvaRegs.CMPR2     =     (period_load_2<<1);
            }
            V_Bsat = 1;
      }
      else
      {
            if (V_Bsat &&     (timer1_dir == 1)){
                  EvaRegs.CMPR2     =     (period_load_2<<1);
            }
            else{
                  EvaRegs.CMPR2     =     period_load_2     -
      t_B;
            }
            V_Bsat = 0;
      }
      // C phase
      if (t_C     >     MAX_TIME)
      {
            EvaRegs.CMPR3     =     0;
            //EvaRegs.T1CMPR = 0;
      }
      else if     (t_C < -MAX_TIME)
      {
            if (!V_Csat && (timer1_dir == 1)){
                  EvaRegs.CMPR3     =     (period_load_2<<1) - 1;
                  //EvaRegs.T1CMPR = (period_load_2<<1)     -     1;
            }
            else{
                  EvaRegs.CMPR3     =     (period_load_2<<1);
                  //EvaRegs.T1CMPR = (period_load_2<<1);
            }
            V_Csat = 1;
      }
      else
      {
            if (V_Csat &&     (timer1_dir == 1)){
                  EvaRegs.CMPR3     =     (period_load_2<<1);
                  //EvaRegs.T1CMPR = (period_load_2<<1);
            }
            else{
                  EvaRegs.CMPR3     =     period_load_2     -
      t_C;
                  //EvaRegs.T1CMPR = period_load_2 - t_C;
            }
            V_Csat = 0;
      }
      DSP_SET_SPI_MASTER();
      SET_SPI_MASTER();
      DISABLE_CPLD();
      hyst_band   =     (400L *     (int16)((double)ADC_VDC_SC *
(double)Vdc)      *     246L) >> 14 ;
      hyst_band         =     real_fixed_band_serial  ;
      hyst_band_B =     hyst_band         ;
      hyst_band_C =     hyst_band         ;
      mod_depth         =     mod_depth_serial              ;
  #ifdef GRAB_INCLUDE
```

```
      if (GrabRunning())
      {
            if ( (int_count&0x0007) == 0)
            {
                  GrabStore(0, duty_cycle_A_3rd);
                  GrabStore(1, duty_cycle_A);
                  GrabStore(2, Voff_scaled);
                  GrabStore(3, Voff);
                  GrabStore(4, Vdc);//zero_crossing_capture_B
);//EvbRegs.CAP4FIFO);
                  GrabStep();
            }
      }
  #endif
      SpiaRegs.SPICTL.bit.MASTER_SLAVE = 1;//Set SPI module to
Master
      //SET_SPI_MASTER();
      // Reinitialize   for   next ADC interrupt
      EvaRegs.EVAIFRA.all     =     BIT9|BIT7;        // clear
interrupt flag
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP2;   // Acknowledge
interrupt to      PIE
      prev_index_sin = index_sin;
      CLEAR_OC_SPI_EN();
      CLEAR_SYNC_PIN();
  }   /* end isr_adc */
  #ifndef   BUILD_RAM
  #pragma   CODE_SECTION(isr_CAP1, "ramfuncs");
  #endif
  interrupt void isr_CAP1(void)
  {
      temp_stack_A = EvaRegs.CAP1FBOT     ;
      PWM_status_A = GpioDataRegs.GPADAT.bit.GPIOA8   ;
      duty_cycle_prev_A =     duty_cycle_A ;
      if(PWM_status_A   == 1)
      {
            on_flag_A   =     1     ;
            first_capture_new_A     =     temp_stack_A;
            if(off_flag_A     ==1)
            {
            off_flag_A = 0 ;
            off_time_A_prev   =     off_time_A ;
            if((first_capture_new_A <     10000L)     &&
(second_capture_A > 60000L))
                  first_capture_new_A     =     first_capture_new_A
      +     65535L ;
            off_time_A = (first_capture_new_A   -
      second_capture_A) ;
            duty_cycle_A = (int) (((200L * on_time_A) /
      (on_time_A + off_time_A))-100L);

            }
            else
                  duty_cycle_A = (int) (((200L * on_time_A) /
      (on_time_A + off_time_A_prev))-100L);
            if(first_capture_new_A < second_capture_A)
                  zero_crossing_capture_A =     ((second_capture_A )
+ first_capture_new_A+  65535L)     >> 1 ;
            else
                  zero_crossing_capture_A =     (second_capture_A +
      first_capture_new_A) >> 1     ;
```

```
            first_capture_old_A     =     first_capture_new_A     ;
      }
      else if     (PWM_status_A     == 0)
      {
            off_flag_A = 1 ;
            second_capture_A = temp_stack_A;
            //second_capture_A = EvaRegs.CAP1FIFO;
            if(on_flag_A ==   1)
            {
            on_flag_A   =     0     ;
            on_time_A_prev = on_time_A ;
            on_time_A   =     abs(second_capture_A -
first_capture_old_A)    ;
            if(on_time_A >=   60000L)
                  on_time_A   =     on_time_A   -     65535L      ;
            if(second_capture_A     <     first_capture_old_A)
                  zero_crossing_capture_A =     ((second_capture_A +
65535L) + first_capture_old_A)      >> 1 ;
            else
                  zero_crossing_capture_A =     (second_capture_A +
      first_capture_old_A) >> 1     ;
            }
            else
                  on_time_A   =     on_time_A_prev ;
      }
      duty_cycle_A = ( 2 * duty_cycle_A) - (duty_cycle_prev_A) ;
      EvaRegs.CAPFIFOA.bit.CAP1FIFO =     2;
      EvaRegs.EVAIFRC.all     =     BIT0;
      //Clear     interrupt   flag
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
//    Acknowledge interrupt   to PIE
   }
   /*#ifndef BUILD_RAM
   #pragma   CODE_SECTION(isr_CAP2, "ramfuncs");
   #endif
   interrupt void isr_CAP2(void)
   {
       EvaRegs.CAPFIFOA.bit.CAP2FIFO =     1;
       EvaRegs.EVAIFRC.all     =     BIT1;
       //Clear     interrupt   flag
       PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
//     Acknowledge interrupt   to PIE
   }
   */
   #ifndef   BUILD_RAM
   #pragma   CODE_SECTION(isr_CAP4, "ramfuncs");
   #endif
   interrupt void isr_CAP4(void)
   {
       temp_stack_C = EvbRegs.CAP4FBOT     ;
       PWM_status_C = GpioDataRegs.GPBDAT.bit.GPIOB8   ;     ;
       duty_cycle_prev_C =     duty_cycle_C ;
       if(PWM_status_C   == 1)
       {
            on_flag_C   =     1     ;
            first_capture_new_C     =     temp_stack_C;
            off_flag_C = 0 ;
            off_time_C_prev   =     off_time_C ;
            if((first_capture_new_C <     10000L)     &&
(second_capture_C > 60000L))
```

```
                first_capture_new_C     =     first_capture_new_C
      +     65535L ;
            off_time_C = (first_capture_new_C   -
      second_capture_C) ;
            duty_cycle_C = (int) (((200L * on_time_C) /
      (on_time_C + off_time_C))-100L);

            if(first_capture_new_C < second_capture_C)
                  zero_crossing_capture_C =     ((second_capture_C )
+ first_capture_new_C+  65535L)     >> 1 ;
            else
                  zero_crossing_capture_C =     (second_capture_C +
      first_capture_new_C) >> 1     ;
            first_capture_old_C     =     first_capture_new_C     ;
      }
      else if     (PWM_status_C     == 0)
      {
            off_flag_C = 1 ;
            second_capture_C = temp_stack_C;
            on_flag_C   =     0     ;
            on_time_C_prev = on_time_C ;
            on_time_C   =     abs(second_capture_C -
first_capture_old_C)    ;
            if(on_time_C >=   60000L)
                  on_time_C   =     on_time_C   -     65535L      ;
            if(second_capture_C     <     first_capture_old_C)
                  zero_crossing_capture_C =     ((second_capture_C +
65535L) + first_capture_old_C)      >> 1 ;
            else
                  zero_crossing_capture_C =     (second_capture_C +
      first_capture_old_C) >> 1     ;
      }
      duty_cycle_C = ( 2 * duty_cycle_C) - (duty_cycle_prev_C) ;
      EvbRegs.CAPFIFOB.bit.CAP4FIFO =     2;
      EvbRegs.EVBIFRC.all     =     BIT0;
      //Clear     interrupt   flag
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP5;
//    Acknowledge interrupt   to PIE
   }
   #ifndef   BUILD_RAM
   #pragma   CODE_SECTION(isr_CAP2, "ramfuncs");
   #endif
   interrupt void isr_CAP2(void)
   {
       PWM_status_B = GpioDataRegs.GPADAT.bit.GPIOA9   ;
       temp_stack_B = EvaRegs.CAP2FBOT     ;
       duty_cycle_prev_B =     duty_cycle_B ;
       if(PWM_status_B   == 1)
       {
            on_flag_B   =     1     ;
            first_capture_new_B     =     temp_stack_B;
            if(off_flag_B     ==1)
            {
            off_flag_B = 0 ;
            if((first_capture_new_B <     10000L)     &&
(second_capture_B > 60000L))
                  first_capture_new_B     =     first_capture_new_B
      +     65535L ;
            off_time_B = (first_capture_new_B   -
      second_capture_B) ;
```

```
            duty_cycle_B = (int) (((200L * on_time_B) /
      (on_time_B + off_time_B))-100L);
            if(first_capture_new_B < second_capture_B)
                  zero_crossing_capture_B =     ((second_capture_B )
+ first_capture_new_B+  65535L)     >> 1 ;
            else
                  zero_crossing_capture_B =     (second_capture_B +
      first_capture_new_B) >> 1     ;
            }
            first_capture_old_B     =     first_capture_new_B     ;
      }
      else if     (PWM_status_B     == 0)
      {
            off_flag_B = 1 ;
            second_capture_B = temp_stack_B;
            if(on_flag_B ==1)
            {
            on_flag_B   =     0     ;
            on_time_B   =     abs(second_capture_B -
first_capture_old_B)    ;
            if(second_capture_B     <     first_capture_old_B)
                  zero_crossing_capture_B =     ((second_capture_B +
65535L) + first_capture_old_B)      >> 1 ;
            else
                  zero_crossing_capture_B =     (second_capture_B +
      first_capture_old_B) >> 1     ;
      }
            zero_crossing_on_flag_B =     1     ;
      }
      duty_cycle_B = ( 2 * duty_cycle_B) - (duty_cycle_prev_B) ;
      EvaRegs.CAPFIFOA.bit.CAP2FIFO =     2;
      EvaRegs.EVAIFRC.all     =     BIT1;
      //Clear     interrupt   flag
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
//    Acknowledge interrupt   to PIE
   }
   #ifndef   BUILD_RAM
   #pragma   CODE_SECTION(isr_CAP5, "ramfuncs");
   #endif
   interrupt void isr_CAP5(void)
   {
       //SET_TP11();
       PWM_status_C = GpioDataRegs.GPBDAT.bit.GPIOB9   ;
       temp_stack_C = EvbRegs.CAP5FBOT     ;
       duty_cycle_prev_C =     duty_cycle_C ;
       if(PWM_status_C   == 1)
       {
            //first_capture_new_C   =     EvbRegs.CAP5FIFO;
            first_capture_new_C     =     temp_stack_C;
            duty_cycle_prev_C =     duty_cycle_C ;
            off_time_C_prev   =     off_time_C ;
            off_time_C = abs(first_capture_new_C - second_capture_C)
;
            if(off_time_C     >= 65535L)
                  off_time_C = (long)off_time_C -     65535L      ;
            else if((off_time_C     >     60000L)     && (off_time_C
< 65535L))
                  off_time_C = 65535L     -     (long)off_time_C ;
            if(abs(off_time_C -     off_time_C_prev) > 1000)
                  off_time_C = off_time_C_prev;
```

```
            duty_cycle_C = (int) (((200L * on_time_C) /
      (on_time_C + off_time_C))-100L);
  //        if(abs(duty_cycle_C     -     duty_cycle_prev_C) > 20)
  //              duty_cycle_C = duty_cycle_prev_C ;
            //DAC1_FAST_WRITE(DAC_WRn_UPDn,DAC_ADDR_B, (duty_cycle_C
* 10)+2047);
            first_capture_old_C     =     first_capture_new_C     ;
      }
      else if     (PWM_status_C     == 0)
      {
            second_capture_C = temp_stack_C;
            on_time_C_prev = on_time_C ;
            on_time_C   =     abs(second_capture_C -
first_capture_old_C)    ;
            duty_cycle_prev_C =     duty_cycle_C ;
            if(on_time_C >=   65535L)
                  on_time_C   =     on_time_C   -     65535L      ;
            else if((on_time_C > 60000L) &&     (on_time_C <
65535L))
                  on_time_C   =     65535L - on_time_C ;
            if(abs(on_time_C - on_time_C_prev) > 1000)
                  on_time_C   =     on_time_C_prev;
            if(second_capture_C     <     first_capture_old_C)
                  zero_crossing_capture_C =     ((second_capture_C +
65535L) + first_capture_old_C)      >> 1 ;
            else
                  zero_crossing_capture_C =     (second_capture_C +
      first_capture_old_C) >> 1     ;
      }
      EvbRegs.CAPFIFOB.bit.CAP5FIFO =     1;
      EvbRegs.EVBIFRC.all     =     BIT1;
      //Clear     interrupt   flag
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP5;
//    Acknowledge interrupt   to PIE
  }
  #ifndef   BUILD_RAM
  #pragma   CODE_SECTION(isr_CAP6, "ramfuncs");
  #endif
  interrupt void isr_CAP6(void)
  {
      EvbRegs.CAPFIFOB.bit.CAP6FIFO =     1;
      EvbRegs.EVBIFRC.all     =     BIT2;
      //Clear     interrupt   flag
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP5;
//    Acknowledge interrupt   to PIE
  }
  #ifndef   BUILD_RAM
  #pragma   CODE_SECTION(isr_T2P,   "ramfuncs");
  #endif
  interrupt void isr_T2P(void){
      /*
      if(master_slave_mode == 1){
            SET_SYNC_PIN();
      }
      */
      EvaRegs.EVAIFRB.all     =     BIT0;
      //Clear     interrupt   flag
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;   // Acknowledge
interrupt to      PIE
  }
```

```
   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
   /**
   Handles   the   PDPINT interrupt caused by a gate   fault.
   \author   A.McIver
   \par History:
   \li 02/05/07 AM -     initial     creation
   */
   #ifndef   BUILD_RAM
   #pragma   CODE_SECTION(isr_gate_fault, "ramfuncs");
   #endif
   interrupt void isr_gate_fault(void)
   {
       PDPINT_Flag =     1     ;
       is_switching = 0;
       VSI_DISABLE();
       mod_targ = 0;
       detected_faults   |= FAULT_VSI_PDPINT;
       // Acknowledge this     interrupt   to receive more
       interrupts from   group 1
       PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
       EvaRegs.EVAIFRA.all     =     BIT0;
   }   /* end isr_gate_fault   */
   /*
==========================================================================
=====
   __VSI_State_Functions()
   ==========================================================================
========== */
   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
   /**
   This function   initialises the   VSI   system.     It resets
       the   target modulation
   depth     to zero.
   It is     followed by the   stop state.
   \author   A.McIver
   \par History:
   \li 12/10/07 AM -     initial     creation
   */
   void st_vsi_init(void)
   {
       mod_ref     =     0;
       mod_targ = 0;
       SetSwFreq(sw_freq);
       SS_NEXT(vsi_state,st_vsi_stop);
   }   /* end st_vsi_init */
   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
   /**
   This is   the   state where the   VSI   is stopped. There is no
       switching. It     waits
   for a     start trigger.
   \author   A.McIver
   \par History:
   \li 12/10/07 AM -     initial     creation
   */
   void st_vsi_stop(void)
   {
       if (SS_IS_FIRST(vsi_state))
       {
```

```
            SS_DONE(vsi_state);
            VSI_DISABLE();
            mod_targ = 0;
            vsi_status &=     ~(VSI_RUNNING|VSI_SETTLED);
      }
      if (detected_faults     != 0)
      {
            SS_NEXT(vsi_state,st_vsi_fault);
            return;
      }
      if (is_switching !=     0) // start trigger
      {
            SS_NEXT(vsi_state,st_vsi_gate_charge);
      }
   }   /* end st_vsi_stop */
   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
   /**
   In this   state the   VSI   gates are   enabled     and   the
      low   side gates held   on to
   charge the high side gate   drivers. The next state is either
      the   ramp state.
   \author   A.McIver
   \par History:
   \li 12/10/07 AM -     initial     creation
   */
   void st_vsi_gate_charge(void)
   {
      if (SS_IS_FIRST(vsi_state))
      {
            SS_DONE(vsi_state);
            vsi_counter =     0;
            //VSI_GATE_CHARGE();
            vsi_status |=     VSI_RUNNING;
      }
      if (detected_faults     != 0)
      {
            SS_NEXT(vsi_state,st_vsi_fault);
            return;
      }
      // check for stop signal
      if (is_switching ==     0)
      {
            SS_NEXT(vsi_state,st_vsi_stop);
            return;
      }
      vsi_counter++;
      if (vsi_counter   >     200)
      {
            SS_NEXT(vsi_state,st_vsi_ramp);
      }
   }   /* end st_vsi_gate_charge     */
   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
   /**
   This state ramps up   the   target modulation depth to match the
reference set by
   the background. It only     changes     the   target every 100ms
and synchronises  the
   change with     a     zero crossing     to avoid step     changes
      in the output.
```

```
   \author   A.McIver
   \par History:
   \li 12/10/07 AM -     initial     creation
   \li 28/04/08 AM -     added event reporting
   */
   void st_vsi_ramp(void)
   {
       if (SS_IS_FIRST(vsi_state))
       {
             SS_DONE(vsi_state);
             vsi_counter =     0;
             if(refMode ==     DC_REF|| refMode ==     SINGLE_AC   ||
refMode ==  SINGLE_AC_G || refMode ==     SINGLE_AC_PR ||   refMode
       == SINGLE_PI_HP){
                   VSI_ENABLE_1P();
             }
             else{
                   VSI_ENABLE();
             }
       }
       if (detected_faults     != 0)
       {
             SS_NEXT(vsi_state,st_vsi_fault);
             return;
       }
       // check for stop signal
       if (is_switching ==     0)
       {
             SS_NEXT(vsi_state,st_vsi_stop);
             return;
       }
       // check for target     reached
       if (mod_targ ==   mod_ref)
       {
             SS_NEXT(vsi_state,st_vsi_run);
             return;
       }
       // ramp     reference   towards     target
       if (mod_ref >     mod_targ + 5)
       {
             mod_targ += 5;
             //Shift     left of     1     introduce   to deal
       with the fact     that mod_targ     200% at     full range
             I_ref_Peak_AB     =
       ((long)I_NOM*((long)mod_targ<<1))>>MOD_SHIFT;
       }
       else if     (mod_ref < mod_targ     -     5)
       {
             mod_targ -= 5;
             //Shift     left of     1     introduce   to deal
       with the fact     that mod_targ     200% at     full range
             I_ref_Peak_AB     =
       ((long)I_NOM*((long)mod_targ<<1))>>MOD_SHIFT;
       }
       else
       {
             mod_targ = mod_ref;
             //Shift     left of     1     introduce   to deal
       with the fact     that mod_targ     200% at     full range
             I_ref_Peak_AB     =
       ((long)I_NOM*((long)mod_targ<<1))>>MOD_SHIFT;
```

```
      }
   }   /* end st_vsi_ramp */
   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
   /**
   This state has the VSI running with     the   target voltage
constant. The output    is
   now ready for   measurements to   begin. If   the   reference   is
changed then      the
   operation moves back to     the   ramp state.
   \author   A.McIver
   \par History:
   \li 12/10/07 AM -     initial     creation
   */
   void st_vsi_run(void)
   {
       if (SS_IS_FIRST(vsi_state))
       {
             SS_DONE(vsi_state);
             vsi_status |=     VSI_SETTLED;
       }
       if (detected_faults     != 0)
       {
             SS_NEXT(vsi_state,st_vsi_fault);
             return;
       }
       // check for stop signal
       if (is_switching ==     0)
       {
             SS_NEXT(vsi_state,st_vsi_stop);
       }
       // check for changes in reference
       if (mod_targ !=   mod_ref)
       {
             vsi_status &=     ~VSI_SETTLED;
             SS_NEXT(vsi_state,st_vsi_ramp);
       }
   }   /* end st_vsi_run */
   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
   /* void   st_vsi_fault(void)
   Parameters:     none
   Returns: nothing
   Description: Delays   for   a     while after faults are cleared.
   Notes:
   History:
       03/11/05 AM -     initial     creation
   \li 04/03/08 AM -     set   vsi_status with   fault bit
   \li 28/04/08 AM -     added event reporting
   */
   void st_vsi_fault(void)
   {
       if (SS_IS_FIRST(vsi_state))
       {
             SS_DONE(vsi_state);
             VSI_DISABLE();
             vsi_counter =     0;
             vsi_status |=     VSI_FAULT;
             vsi_status &=     ~(VSI_RUNNING|VSI_SETTLED);
             putxx(detected_faults);
             put_str("->VSI faults\n");
```

```
      }
      if (detected_faults     == 0)
            vsi_counter++;
      else
            vsi_counter =     0;
      if (vsi_counter   >     100)
      {
            //PDPINT_Flag     =     1     ;
            vsi_status &=     ~VSI_FAULT;
            SS_NEXT(vsi_state,st_vsi_stop);
      }
   }   /* end st_vsi_fault     */
   /*
=================================================================
=====
   __Local_Functions()
   =================================================================
========== */
   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
   /**
   This function   is called   every fundamental period to   perform
      the   RMS
   calculations and scale the analog quantities to     Volts and
      Amps for use in   the
   background.
   \author   A.McIver
   \par History:
   \li 12/10/07 AM -     derived     from IR25kVA:vsi:adc_scale
   \li 21/08/08 AM -     added VSI   DC offset   compensation
   \li 12/09/08 AM -     added stop_count and moved to floating point
data
   */
   void scale_adc_rms(void)
   {
      double
            val,
            temp;
      // calculate A0   RMS   quantity
      temp =
(double)adc_int.A0.dc_sum_bak/(double)adc_int.count_rms_bak;
      val   =
      (double)adc_int.A0.rms_sum_bak*(double)(1<<ADC_RMS_PS)
                              /     (double)adc_int.count_rms_bak -
      temp*temp;
      if (val     <     0.0) val = 0.0;
      adc_int.A0.real   =     ADC_REAL_SC *     sqrt(val);
   }   /* end scale_adc_rms */
   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
   /**
   This function   is called   every ADC_DC_TIME to perform the DC
      calculations and
   scale     the   analog quantities to Volts and Amps for   use   in
the background.
   \author   A.McIver
   \par History:
   \li 12/10/07 AM -     derived     from IR25kVA:vsi:adc_scale
   */
   void scale_adc_dc(void)
   {
```

```
      double
            val;
      // calculate B0   DC quantity
      val   =
      (double)adc_int.B0.dc_sum_bak/(double)ADC_COUNT_DC;
      adc_int.B0.real   =     ADC_REAL_SC *     val;
   }   /* end scale_adc_dc     */
   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
   /**
   Calibrates the adc for gain and   offset using the reference
inputs.
   See spra989a.pdf for calibration details
   \author   A.McIver
   \par History:
   \li 07/10/05 AM -     initial     creation
   */
   void calibrate_adc(void)
   {
   //  char
   //       str[60];
      double
            yHA   =     0.0,
            yLA,
            yHB,
            yLB;
      yHA   =
      (double)adc_int.yHA.dc_sum_bak/(double)ADC_COUNT_CAL;
      yLA   =
      (double)adc_int.yLA.dc_sum_bak/(double)ADC_COUNT_CAL;
      yHB   =
      (double)adc_int.yHB.dc_sum_bak/(double)ADC_COUNT_CAL;
      yLB   =
      (double)adc_int.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 > 0.94) &&     (cal_gain_A <     1.05) )
            && ( (cal_gain_B > 0.94) &&   (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_offset_A,cal_gain_B,cal_offset_B);
   //  put_str(str);
   }   /* end calibrate_adc */
   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
   void get_state(void){
      if(vsi_state.f == st_vsi_init){
            put_str("INIT     ");
      }
```

```
      else if(vsi_state.f     == st_vsi_stop){
            put_str("STOP     ");
      }
      else if(vsi_state.f     == st_vsi_gate_charge){
            put_str("GATE     ");
      }
      else if(vsi_state.f     == st_vsi_ramp){
            put_str("RAMP     ");
      }
      else if(vsi_state.f     == st_vsi_run){
            put_str("RUN      ");
      }
      else if(vsi_state.f     == st_vsi_fault){
            put_str("FAU      ");
      }
  }
  /* sets   the   switching   frequency   :     returns
      switching   frequency   achieved */
  /* fsw is in Hz */
  int SetSwFreq(int     fsw)
  {
      unsigned int half_period;
      half_period =     (unsigned   int)((HSPCLK/2.0/fsw));
      //if (half_period >     MAX_PER_2) half_period = MAX_PER_2;
      //else if   (half_period < MIN_PER_2)     half_period =
      MIN_PER_2;
      period_2 = half_period;
      period = period_2*2;
      sw_freq     =     fsw;              /* Write new switching
freq  to global   variable */
      MAX_TIME    =     (int16)(period_2-6)     ;
      //Recalculate     phase advance     speed for   sin   table
      read
      PHASE_STEP_SC     =     (65536.0*65536.0/(fsw*2.0));
      vsi_set_freq(Ref_freq_float);
      //If switching frequency is   changed     re-calculate
controller  parameters
      set_KI();
      set_KP();
      return (int)((HSPCLK/2/(long)half_period + 1)/2);
  }   /* end SetSwFreq */
  void step_toggle(int direction){
      if(direction ==   ON){
            step_tog = ON;
      }
      else if     (direction ==     OFF){
            step_tog = OFF;
      }
  }
  void SetHystBand(double     real_sw_freq_serial_in)
  {
       //4095     give 5V     so 1V gives 819
      hyst_band   =
      (int16)((double)ADC_VDC_SC*(double)Vdc*(double)Ih_CONST/(doubl
e)real_sw_freq_serial_in);
  }
  void set_ref_mode(unsigned int mode){
            refMode     =     mode;
  }
  int get_phase_step(void){
      return phase_step;
```

```
}
void step_ref_setup(unsigned long int   phase, unsigned   int
    req_new_mag){
    count_from_zero_for_step = phase / phase_step;
    new_mod_targ = req_new_mag;
    //mod_ref   =     new_mod_targ;
    step_ref_request = 1;
}
void step_phase_setup(unsigned long     int   phase, unsigned
    int   step_size){
    count_from_zero_for_step = phase / phase_step;
    add_phase   =     step_size;
    step_phase_request = 1;
}
void set_Feedforward(int status){
    FFenable = status;
}
int get_Feedforward_status(void){
    return FFenable;
}
void set_RefMode(int mode){
    refMode     =     mode;
}
int get_Ref_mode(void){
    return refMode;
}
void display_ref_mode(void){
    if(refMode ==     DC_REF){
          put_str("DC PI ");
    }
    else if(refMode   == SINGLE_AC){
          put_str("1P PI ");
    }
    else if(refMode   == THREE_PHASE_PI){
          put_str("3P PI ");
    }
    else if(refMode   == SINGLE_AC_G){
          put_str("1 PI     G     ");
    }
    else if(refMode   == SINGLE_AC_OL){
          put_str("1P OL ");
    }
    else if(refMode   == THREE_PHASE_DQ){
          put_str("3P DQ ");
    }
    else if(refMode   == THREE_PHASE_PI_G){
          put_str("3P G      ");
    }
    else if(refMode   == TPH_HCC_VB_3CNT){
          put_str("3P_HYS   ");
    }
    else if(refMode   == SINGLE_PHASE_HS){
          put_str("1P_HYS   ");
    }
    else if(refMode   == SINGLE_AC_PR){
          put_str("1P PR ");
    }
    else if(refMode   == THREE_PHASE_PR){
          put_str("3P PR ");
    }
    else if(refMode   == SQUARE_WAVE){
```

```
            put_str("SQ WV ");
        }
        else if(refMode   == THREE_PHASE_OL){
            put_str("3P OL ");
        }
        else if(refMode   == PHASE_A){
            put_str("P_A ");
        }
        else if(refMode   == PHASE_B){
            put_str("P_B ");
        }
        else if(refMode   == PHASE_C){
            put_str("P_C ");
        }
        else if(refMode   == SINGLE_PI_HP){
            put_str("HP PI ");
        }
        else if(refMode   == TPH_HCC_FB_2CNT){
            put_str("FB_2CNT ");
        }
        else if(refMode   == TPH_HCC_VB_2CNT){
            put_str("VB_2CNT ");
```

B. 3 Background C code for Three-level Variable Band HCC

```
/**
\file
\brief System software for the DA-2810 Demo code


\par Developed By:
   Creative Power Technologies, (C) Copyright 2009
\author A.McIver
\par History:
\li 23/04/09 AM - initial creation
*/

// compiler standard include files
#include <stdlib.h>
#include <stdio.h>

// processor standard include files
#include <DSP281x_Device.h>
#include <DSP281x_Examples.h>


// board standard include files
//#include <lib_da2810.h>
#include <lib_mini2810.h>
#include <lib_cpld.h>
#include <lib_giib.h>

// common project include files
//#define AD5624
#define DAC_SHIFT 4
#include <dac_ad56.h>

// common project include files
```

```
   // local include files
   #include "main.h"
   #include "conio.h"
   #include "vsi.h"
   #include "curreg.h"
   #include "cas.h"



   /*
=========================================================================
=====
   __Definitions()
   =======================================================================
========== */
   #define LCD_CTRL                  (ADD_MINICS2_BASE+MINIBUS_MA1)
   #define TPB2                      BIT2
   #define SET_TPB2()                GpioDataRegs.GPBSET.all = TPB2
   #define CLEAR_TPB2()        GpioDataRegs.GPBCLEAR.all = TPB2
   /*
=========================================================================
=====
   __Typedefs()
   =======================================================================
========== */

   /// Time related flag type
   /** This structure holds flags used in background timing. */
   typedef struct
   {
       Uint16
             usec100:1,   ///<1/10 millisecond
             msec:1,           ///< millisecond flag
             msec10:1,   ///< 10ms flag
             sec0_1:1,   ///< tenth of a second flag
             sec:1,            ///< second flag
             sec5:1;
   } type_time_flag;


   /*
=========================================================================
=====
   __Variables()
   =======================================================================
========== */

   #ifndef BUILD_RAM
   // These are defined by the linker (see F2812.cmd)
   extern Uint16 RamfuncsLoadStart;
   extern Uint16 RamfuncsLoadEnd;
   extern Uint16 RamfuncsRunStart;
   #endif

   // Background variables
   Uint16
       quit = 0;                   ///< exit flag

   /// timing variable
   type_time_flag
```

```
      time =
      {
            0,0,0,0 // flags
      };

  Uint32
      idle_count = 0,                     ///< count of idle time in
the background
      idle_count_old = 0,                 ///< previous count of
idle time
      idle_diff = 0;                      ///< change in idle time
btwn low speed tasks

  char
      str[40];                      // string for displays

  extern signed int
      ZX_time;


  extern int16
      loop_back_character;

  //testing variable for CPLD

  extern int16 master_slave_mode;

  extern int16 sync_method;
  extern int16 NO_SPI_CHAR;

  int16 Unit_number = 0;

  extern Uint16 slave_delay;
  extern int16 Vdc ;
  long k = 0;
  int second_watch = 0 ;
  int second_watch_flag = 0 ;
  /*
=========================================================================
=====
  __Serial_input_variable()
  =========================================================================
========== */
  int mod_depth_serial =0;                    //In 2810 modulation
depth go from 0 to 1000 (0-100%)
  int step_mod_depth_serial = 20;
  int final_mod_depth_serial = 50;
  int mod_depth_max = MOD_DEPTH_MAX;

  double mod_f_freq_serial = INIT_FF;
      //Fundamental modulation frequency in Hz
  double step_f_freq_serial = 0.5;
  double mod_f_freq_max = 100.0;

  int sw_freq_serial                          = SW_FREQ;
  int step_sw_freq_serial                 = 50;
  int real_sw_freq_serial                 =  SW_FREQ;
  int step_real_sw_freq_serial            = 50;
  int real_bus_voltage_serial             =  1450 ;
  int level_select_point_serial           =  50 ;
  int real_fixed_band_serial_B            =  150;
```

```
   int real_fixed_band_serial_C           =  610;
   int real_step_fixed_band_serial   = 5 ;
   int real_V3rd_serial                   = 0 ;
   int real_V3rd_offset_serial            = 0 ;
   int average_offset_serial              = 1 ;

   Uint16 real_fixed_band_serial                =  1450;

   double pf_adj_serial                   = 0 ;
   extern Uint16 is_switching;

   double
       real_Kp_serial = KP_INIT,
       real_Tint_serial = TINT_INIT,
       real_Hband_serial = BMAX;

   long
       step_at_phase_serial = 0;

   int
       step_enable_flag = FALSE,
       step_direction = 1;

   Uint16 spi_fail_count_cpld;
   int16 trash_spi_cpld;

   extern int16
       carrier_offset, carrier_allowed_error;

   extern int16
       ctrl_latch;
   extern int16
          Flag_max_finder;
   extern int16    MDepth_max_main ;
   //extern int16
   //  fault ;
   int16 stepping = 0 ;
   int display_help_counter = 0 ;
   int screen_flag = 0 ;
   int help_flag = 0 ;

   /*
=========================================================================
=====
   __Local_Function_Prototypes()
   =======================================================================
========== */

   /* 1 second interrupt for display */
   interrupt void isr_cpu_timer0(void);

   /// display operating info
   void com_display(void);

   /* process keyboard input */
   void com_keyboard(void);
   void display_help(void) ;
   void EnablePowerRelay(void);
   void init_dac_mini(void);
   void SetHystBand(double);
   void SetBusVolts(double);
```

```
   void CPLD_FAST_WRITE(char,char);
   Uint16 CPLD_FAST_READ(char);
   /*
==========================================================================
=====
   __Grab_Variables()
   ==========================================================================
========== */

   #ifdef GRAB_INCLUDE
   //#pragma DATA_SECTION(grab_array, "bss_grab")
   int16
       grab_mode = GRAB_STOPPED,
       grab_index;
   int32
       grab_array[GRAB_LENGTH][GRAB_WIDTH];
   #endif


   /*
==========================================================================
===== */
   /* Main */
   /*
==========================================================================
===== */
   /* Idle time benchmark:
   \li Ram based program with only bios interrupt and an empty main
loop gives an
   idle_diff of 4.69M (4,685,900)
   \li 23/03/09 V1.02 1.23M with no modbus running
   */
   void main(void)
   {
       static int i = 0;
       Uint32 while_counter=0;

   // Disable CPU interrupts
       DINT;
   // Initialise DSP for PCB
       lib_mini2810_init(150/*MHz*/,37500/*kHz*/,150000/*kHz*/,LIB_EV
AENCLK

       |LIB_EVBENCLK|LIB_ADCENCLK|LIB_SCIAENCLK|LIB_SCIBENCLK|LIB_MCB
SPENCLK);

       InitGpio();
       spi_init(MODE_CPLD);
       //SpiaRegs.SPICCR.bit.SPILBK = 1;                      //Set
SPI on loop back for testing
       cpld_reg_init();
       //giib_init();
       giib_init();
   // 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();

   #ifndef BUILD_RAM
   // 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

       InitAdc();
       InitCpuTimers();

       // Initialise COM port
       bios_init(9600L);

   // Configure CPU-Timer 0 to interrupt every tenth of a second:
   // 150MHz CPU Freq, 1ms Period (in uSeconds)
       ConfigCpuTimer(&CpuTimer0, 150.0/*MHz*/, 1000.0/*us*/);
       StartCpuTimer0();

   // Interrupts that are used in this example are re-mapped to
   // ISR functions found within this file.
       EALLOW;  // This is needed to write to EALLOW protected
register
       PieVectTable.TINT0 = &isr_cpu_timer0;
       EDIS;    // This is needed to disable write to EALLOW
protected registers

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

       //put_str("\n\nCPT-DA2810 Demo VSI\n\n");


       //Setting up off board SPI mode: WYK 20090729
       spi_set_mode(MODE_CPLD);                    //Use mode setting
for CPLD for SPI to initialize SPI setting

       //Read the DIP switch on GIIB to determine the unit number of
the unit
       Unit_number = ReadDigIn();

       /* Unit 0 is always the master module of the network
responsible for sending sync pulse and master message */
       if(Unit_number == 0)
       {
             master_slave_mode = 1;
       }
       else
```

```
	{
		master_slave_mode = 0;
	}

	// Force DIGIN5-6 to CAP1-2
	//CPLD.CAPQEP.bit.CP = 1;
	//cpld_write(ADD_CAPQEP,CPLD.CAPQEP.all);

	CAS_init();

	if (master_slave_mode == 0)
	{
		SPI_Secondary_Master_DSP_Slave();
	}
	else
	{
		SPI_Secondary_Slave_DSP_Master();
	}
	DISABLE_CPLD();

	EALLOW;
	GpioDataRegs.GPFDAT.bit.GPIOF3 = 0;
	GpioMuxRegs.GPFMUX.bit.SPISTEA_GPIOF3 = 0;
	GpioMuxRegs.GPFDIR.bit.GPIOF3 = 1;
	EDIS;
	//EnableMOSFET2();

	if (master_slave_mode == 0)
	{
		DINT;

		SPI_Primary_en_Secondary_dis();

		CPLD_FAST_WRITE(ADD_CAPQEP,0x00);
		CPLD_FAST_WRITE(ADD_HSPWMDB,0xFA);
		CPLD_FAST_WRITE(ADD_SPECIAL,0x0A);
		CPLD_FAST_WRITE(ADD_ANLGSW,0x90);
		CPLD_FAST_WRITE(ADD_EVACOMCON,0xFF);
		SPI_Primary_dis_Secondary_en();
		set_RefMode(SPH_3L_HCC);
		vsi_disable();
		step_enable_flag = FALSE;
		EALLOW ;
		GpioMuxRegs.GPADIR.bit.GPIOA0 = 0 ;
		GpioMuxRegs.GPADIR.bit.GPIOA1 = 1 ;
		GpioMuxRegs.GPADIR.bit.GPIOA2 = 0 ;
		GpioMuxRegs.GPADIR.bit.GPIOA3 = 0 ;
		GpioMuxRegs.GPADIR.bit.GPIOA4 = 0 ;
		GpioMuxRegs.GPADIR.bit.GPIOA5 = 0 ;
		GpioMuxRegs.GPADIR.bit.GPIOA6 = 1 ;
		GpioMuxRegs.GPADIR.bit.GPIOA7 = 0 ;


		GpioMuxRegs.GPAMUX.bit.PWM1_GPIOA0 = 0;   // enable PWM1
pin
		GpioMuxRegs.GPAMUX.bit.PWM2_GPIOA1 = 0;   // enable PWM2
pin
		GpioMuxRegs.GPAMUX.bit.PWM3_GPIOA2 = 0;   // enable PWM3
pin
		GpioMuxRegs.GPAMUX.bit.PWM4_GPIOA3 = 0;   // enable PWM4
pin
```

```
            GpioMuxRegs.GPAMUX.bit.PWM5_GPIOA4 = 0;   // enable PWM5
pin
            GpioMuxRegs.GPAMUX.bit.PWM6_GPIOA5 = 0;   // enable PWM6
pin
            GpioMuxRegs.GPAMUX.bit.T1PWM_GPIOA6= 0;   // enable PWM7
pin
            //GpioMuxRegs.GPAMUX.bit.T2PWM_GPIOA7= 1; // enable PWM8
pin
            EDIS;
            EINT;

            SPI_Primary_en_Secondary_dis();
            put_str("SPECIAL: ");
            putxx(cpld_read(ADD_SPECIAL));
            put_str("EVACOMCON: ");
            putxx(cpld_read(ADD_EVACOMCON));
            SPI_Primary_dis_Secondary_en();
            put_str("\n");
            // Set SPI to slave mode
            SpiaRegs.SPICTL.bit.MASTER_SLAVE = 0;
      }
      else
      {
                        DINT;

                              SPI_Primary_en_Secondary_dis();

                              CPLD_FAST_WRITE(ADD_CAPQEP,0x00);
                              CPLD_FAST_WRITE(ADD_HSPWMDB,0xFA);
                              CPLD_FAST_WRITE(ADD_SPECIAL,0x0A);
                              CPLD_FAST_WRITE(ADD_ANLGSW,0x90);
                              CPLD_FAST_WRITE(ADD_EVACOMCON,0xFF);

                              SPI_Primary_dis_Secondary_en();
                              set_RefMode(SPH_3L_HCC);
                              vsi_disable();
                              step_enable_flag = FALSE;
                              EALLOW ;
                              GpioMuxRegs.GPADIR.bit.GPIOA0 = 0 ;
                              GpioMuxRegs.GPADIR.bit.GPIOA1 = 1 ;
                              GpioMuxRegs.GPADIR.bit.GPIOA2 = 0 ;
                              GpioMuxRegs.GPADIR.bit.GPIOA3 = 0 ;
                              GpioMuxRegs.GPADIR.bit.GPIOA4 = 0 ;
                              GpioMuxRegs.GPADIR.bit.GPIOA5 = 0 ;
                              GpioMuxRegs.GPADIR.bit.GPIOA6 = 1 ;
                              GpioMuxRegs.GPADIR.bit.GPIOA7 = 0 ;


                              GpioMuxRegs.GPAMUX.bit.PWM1_GPIOA0 =
0;    // enable PWM1 pin
                              GpioMuxRegs.GPAMUX.bit.PWM2_GPIOA1 =
0;    // enable PWM2 pin
                              GpioMuxRegs.GPAMUX.bit.PWM3_GPIOA2 =
0;    // enable PWM3 pin
                              GpioMuxRegs.GPAMUX.bit.PWM4_GPIOA3 =
0;    // enable PWM4 pin
                              GpioMuxRegs.GPAMUX.bit.PWM5_GPIOA4 =
0;    // enable PWM5 pin
                              GpioMuxRegs.GPAMUX.bit.PWM6_GPIOA5 =
0;    // enable PWM6 pin
```

```
                                GpioMuxRegs.GPAMUX.bit.T1PWM_GPIOA6=
0;    // enable PWM7 pin
                                //GpioMuxRegs.GPAMUX.bit.T2PWM_GPIOA7=
1;    // enable PWM8 pin
                                EDIS;
                                EINT;

                                SPI_Primary_en_Secondary_dis();
                                put_str("\n");
                                put_str("SPECIAL: ");
                                putxx(cpld_read(ADD_SPECIAL));
                                put_str("\n");
                                put_str("EVACOMCON: ");
                                putxx(cpld_read(ADD_EVACOMCON));
                                SPI_Primary_dis_Secondary_en();
                                put_str("\n");
            }

      // 3rd DIGIO socket pin is fault
      // Routed to master through NPC SPI Comms board
      EALLOW;
      GpioDataRegs.GPBDAT.bit.GPIOB2 = 0;
      GpioMuxRegs.GPBMUX.bit.PWM9_GPIOB2 = 0;
      GpioMuxRegs.GPBDIR.bit.GPIOB2 = 1;
      EDIS;

      SPI_Primary_dis_Secondary_en();
      SPI_Primary_en_Secondary_dis();

  #ifdef GRAB_INCLUDE
      GrabInit();
  #endif
      vsi_init();



      // initialise the chip selects on the board and initialise the
spi
      DISABLE_CPLD();                                          //
Disable the CPLD interface
  //  SET_SPI_MASTER();                                 // Set the
system to Master mode
      CLEAR_OC_SPI_EN();                                       // Turn
off external SPI access
      DISABLE_DAC1();
      DISABLE_DAC2();

      spi_init(MODE_CPLD);
      init_dac_mini();

      //set_compensation_filter();


      put_str("\n\n\tCS-IIB V1.0 --> Three Phase Multilevel
Hysteresis Controller Board\n");
      while(while_counter < 250000) while_counter++;
      put_str("\n\n\tReza Davoodnezhad\n");
      while(while_counter < 250000) while_counter++;
  /*
  void main_loop(void)
  */
```

```
      while(quit == 0)
      {
            com_keyboard(); // process keypresses

          if (time.msec != 0) // millisecond events
            {
                  k++;
                  if(k%2==0)
                  SET_TPB2();
                  else
                  CLEAR_TPB2();

                  time.msec = 0;
                  vsi_state_machine();

            }
            else if (time.msec10 != 0) // ten millisecond events
            {
                  time.msec10 = 0;
            }
            else if (time.sec0_1 != 0) // tenth of second events
            {
                  time.sec0_1 = 0;
                  if(help_flag)display_help() ;
                  if(GrabShowTrigger() && i < GRAB_LENGTH){
                        GrabDisplay(i);
                        i++;
                  }
                  else if(GrabShowTrigger() && i == GRAB_LENGTH){
                        GrabStop();
                        i = 0;
                  }
            }
            else if (time.sec != 0) // one second events
            {
                  second_watch ++ ;
                  if(second_watch == 10)
                        second_watch_flag = 1 ;
                  time.sec = 0;
                  idle_diff = idle_count - idle_count_old;
                  idle_count_old = idle_count;
                  if(screen_flag == 0) com_display(); // one second
display
            }
            else if (time.sec5 != 0){                 //five second
events5
                  time.sec5 = 0;
                  if(step_enable_flag == TRUE){
                        if(step_direction == 1){
      //Stepping the reference to final value
                              step_direction = 0;
                              step_ref_setup(step_at_phase_serial,
final_mod_depth_serial);
                        }
                        else{
                              //Stepping the reference to initial
value
                              step_direction = 1;
                              step_ref_setup(step_at_phase_serial,
mod_depth_serial);
                        }
```

```
                }
            }
            else // low priority events
            {
                idle_count++;
            }
        } /* end while quit == 0 */

    //  DISABLE_PWM();
        EvaRegs.T1CON.bit.TENABLE = 0;
        EvaRegs.ACTRA.all = 0x0000;
        DINT;
    } /* end main */


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

    /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
    /**
    Display operating information out COM1.

    \author A.McIver
    \par History:
    \li 22/06/05 AM - initial creation5

    \param[in] mode Select whether to start a new display option
    */
    void com_display(void)
    {

        if(GrabShowTrigger())
        {}
        else
        {

            if(master_slave_mode ==1)
            {
                put_str("M");
            }
            else
            {
                put_str("S");
            }
            putu(Unit_number);
            put_str(",");

            if (is_switching == 1)
                put_str(" En");
            else
                put_str("Dis");
            put_str(" ");

            display_ref_mode();
            put_str(",");
            put_str(" Iref:");
```

```
          putdbl((double)vsi_get_mod()*20.00/32768.00,2);
            put_str(" , ");

            put_str(" Vb:");
          putdbl(Vdc,0);
            put_str("V");
            put_str(" , ");

            put_str("FF:");
            putdbl(vsi_get_freq(),2);
            put_str(" , ");

            put_str("Bi:");
            putdbl(real_fixed_band_serial,2);
            put_str(" , ");

            put_str("Vbus:");
            putdbl(real_bus_voltage_serial,2);
            put_str(" , ");

            put_str("adj:");
            putdbl(average_offset_serial,2);
            put_str(" , ");

            put_str("LSP");
            putdbl(level_select_point_serial,2);
            put_str(" , ");

            put_str("\r");

      }
  } /* end com_display */


  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
  /* void com_keyboard
  Parameters: none
  Returns: nothing
  Description: Process characters from COM0.
  Notes:
  History:
      22/06/05 AM - initial creation
  \li 27/11/07 PM - added in testing of the digital I/O
  */
  void select_operating_mode(void)
  {

      char c;
      Uint16 valid = 0;
      Uint32 while_counter = 0;

      // Clear anything on buffer
      while(Kbhit())    c = get_char();

      while(valid!=1)
      {
            if (Kbhit())
            {
                  c = get_char();
                  switch(c)
```

```
                    {
                          case '1': valid = 1 ;
                                DINT;
                                SpiaRegs.SPICTL.bit.MASTER_SLAVE = 1;
                                SPI_Primary_en_Secondary_dis();
                                CPLD_FAST_WRITE(ADD_CAPQEP,0x00);
                                CPLD_FAST_WRITE(ADD_HSPWMDB,0xFA);
                                CPLD_FAST_WRITE(ADD_SPECIAL,0x0A);
                                CPLD_FAST_WRITE(ADD_ANLGSW,0x90);
                                CPLD_FAST_WRITE(ADD_EVACOMCON,0xFF);
                                SPI_Primary_dis_Secondary_en();
                                set_RefMode(SPH_3L_HCC);
                                vsi_disable();
                                step_enable_flag = FALSE;
                                EALLOW ;
                                GpioMuxRegs.GPADIR.bit.GPIOA0 = 0 ;
                                GpioMuxRegs.GPADIR.bit.GPIOA1 = 1 ;
                                GpioMuxRegs.GPADIR.bit.GPIOA2 = 0 ;
                                GpioMuxRegs.GPADIR.bit.GPIOA3 = 0 ;
                                GpioMuxRegs.GPADIR.bit.GPIOA4 = 0 ;
                                GpioMuxRegs.GPADIR.bit.GPIOA5 = 0 ;
                                GpioMuxRegs.GPADIR.bit.GPIOA6 = 1 ;
                                GpioMuxRegs.GPADIR.bit.GPIOA7 = 0 ;


                                GpioMuxRegs.GPAMUX.bit.PWM1_GPIOA0 =
0;    // enable PWM1 pin
                                GpioMuxRegs.GPAMUX.bit.PWM2_GPIOA1 =
0;    // enable PWM2 pin
                                GpioMuxRegs.GPAMUX.bit.PWM3_GPIOA2 =
0;    // enable PWM3 pin
                                GpioMuxRegs.GPAMUX.bit.PWM4_GPIOA3 =
0;    // enable PWM4 pin
                                GpioMuxRegs.GPAMUX.bit.PWM5_GPIOA4 =
0;    // enable PWM5 pin
                                GpioMuxRegs.GPAMUX.bit.PWM6_GPIOA5 =
0;    // enable PWM6 pin
                                GpioMuxRegs.GPAMUX.bit.T1PWM_GPIOA6=
0;    // enable PWM7 pin
                                //GpioMuxRegs.GPAMUX.bit.T2PWM_GPIOA7=
1;    // enable PWM8 pin
                                EDIS;
                                SpiaRegs.SPICTL.bit.MASTER_SLAVE = 0;
                                EINT;
                          break;

                          case '2': valid = 1 ;
                                DINT;
                                CPLD_FAST_WRITE(ADD_CAPQEP,0x00);
                                CPLD_FAST_WRITE(ADD_HSPWMDB,0xFA);
                                CPLD_FAST_WRITE(ADD_SPECIAL,0x0A);
                                CPLD_FAST_WRITE(ADD_ANLGSW,0x90);
                                CPLD_FAST_WRITE(ADD_EVACOMCON,0xFF);
                                set_RefMode(SPH_3L_HCC);
                                vsi_disable();
                                step_enable_flag = FALSE;
                                EALLOW ;
                                GpioMuxRegs.GPADIR.bit.GPIOA0 = 0 ;
                                GpioMuxRegs.GPADIR.bit.GPIOA1 = 0 ;
                                GpioMuxRegs.GPADIR.bit.GPIOA2 = 0 ;
                                GpioMuxRegs.GPADIR.bit.GPIOA3 = 0 ;
```

```
                                GpioMuxRegs.GPADIR.bit.GPIOA4 = 0 ;
                                GpioMuxRegs.GPADIR.bit.GPIOA5 = 0 ;
                                GpioMuxRegs.GPADIR.bit.GPIOA6 = 1 ;
                                GpioMuxRegs.GPADIR.bit.GPIOA7 = 0 ;


                                GpioMuxRegs.GPAMUX.bit.PWM1_GPIOA0 =
0;    // enable PWM1 pin
                                GpioMuxRegs.GPAMUX.bit.PWM2_GPIOA1 =
0;    // enable PWM2 pin
                                GpioMuxRegs.GPAMUX.bit.PWM3_GPIOA2 =
0;    // enable PWM3 pin
                                GpioMuxRegs.GPAMUX.bit.PWM4_GPIOA3 =
0;    // enable PWM4 pin
                                GpioMuxRegs.GPAMUX.bit.PWM5_GPIOA4 =
0;    // enable PWM5 pin
                                GpioMuxRegs.GPAMUX.bit.PWM6_GPIOA5 =
0;    // enable PWM6 pin
                                GpioMuxRegs.GPAMUX.bit.T1PWM_GPIOA6=
0;    // enable PWM7 pin
                                //GpioMuxRegs.GPAMUX.bit.T2PWM_GPIOA7=
1;    // enable PWM8 pin
                                EDIS;
                                EINT;
                          break;

                          case '3': DINT; valid = 1 ;
CPLD_FAST_WRITE(ADD_CAPQEP,0x00);CPLD_FAST_WRITE(ADD_HSPWMDB,0xFA);C
PLD_FAST_WRITE(ADD_SPECIAL,0x0A);CPLD_FAST_WRITE(ADD_ANLGSW,0x90);CP
LD_FAST_WRITE(ADD_EVACOMCON,0xFF);
                          set_RefMode(SINGLE_PHASE_HS);
vsi_disable();  step_enable_flag = FALSE;EINT;break;

                          case '4': DINT; valid = 1 ;
CPLD_FAST_WRITE(ADD_CAPQEP,0x00);CPLD_FAST_WRITE(ADD_HSPWMDB,0xFA);C
PLD_FAST_WRITE(ADD_SPECIAL,0x09);CPLD_FAST_WRITE(ADD_ANLGSW,0x80);CP
LD_FAST_WRITE(ADD_EVACOMCON,0x9F);
                          set_RefMode(THREE_PH_2CONT_HS);
vsi_disable();  step_enable_flag = FALSE;EINT;break;
                    }
              }
        }
    }
    /**
    Displays help menu

    \author R.Davoodnezhad

    \par History:
    \li 05/05/11 RD - Creation

    */
    void display_help(void)
    {
        switch(display_help_counter)
        {
              /* case 0 never happens */
              case 1: put_str("\n\tHelp Menu\n");
                                                    break;
              case 2: put_str("\t Press 'e' to enable VSI\n");
                                                 break;
```

```
            case 3: put_str("\t Press 'd' to disable VSI\n");
                                                  break;
            case 4: put_str("\t Press 'o' to change operation
mode\n") ;                                    break;
            case 5: put_str("\t Press 'f' to set a 50Hz
fundamental\n");                                  break;
            case 6: put_str("\t Press 'i/I' to increase/decrease
reference current\n");              break;
            case 7: put_str("\t Press 's/S' to increase/decrease
Phase A hysteresis band\n");  break;
            case 8: put_str("\t Press 'z/Z' to increase/decrease
Phase B hysteresis band\n");  break;
            case 10:put_str("\t Press 'a/A' to increase/decrease
Phase C hysteresis band\n\n");screen_flag = 0;  break;
            default: break;
      }
      if (display_help_counter<11) display_help_counter++;

   }

   void com_keyboard(void)
   {

       char c;

   //  put_str("KEY");
       if (Kbhit())
       {
             c = get_char();
             switch (c)
             {
                         case '!':
                               DINT;
                               spi_init(MODE_CPLD);
                               SPI_Primary_en_Secondary_dis();
                               put_str("SPECIAL: ");
                               putxx(cpld_read(ADD_SPECIAL));
                               SPI_Primary_dis_Secondary_en();
                               put_str("\n");
                               EINT;
                         break ;
                         case '@':
                         DINT;
                               SpiaRegs.SPICTL.bit.MASTER_SLAVE = 1;
                               spi_init(MODE_CPLD);
                               SPI_Primary_en_Secondary_dis();
                               put_str("EVACOMCON: ");
                               putxx(cpld_read(ADD_EVACOMCON));
                               SPI_Primary_dis_Secondary_en();
                               put_str("\n");
                               SpiaRegs.SPICTL.bit.MASTER_SLAVE = 0;
                               EINT;

                         break;
                         case '#' :
                         DINT;
                               SPI_Primary_en_Secondary_dis();
                               put_str("HSPWMDB: ");
                               putxx(cpld_read(ADD_HSPWMDB));
                               SPI_Primary_dis_Secondary_en();
                               put_str("\n");
```

```
                              EINT;
                        break;
                        case '$' :
                        DINT;
                              SPI_Primary_en_Secondary_dis();
                              put_str("ANLGSW: ");
                              putxx(cpld_read(ADD_VERSION));
                              SPI_Primary_dis_Secondary_en();
                              put_str("\n");
                              EINT;
                        break;

                        case '%' :
                        DINT;
                              SPI_Primary_en_Secondary_dis();
                              put_str("HSPWM_STATUS: ");
                              putxx(cpld_read(ADD_HSPWM_STATUS));
                              SPI_Primary_dis_Secondary_en();
                              put_str("\n");
                              EINT;
                        break;

                  case 'e':
                        vsi_enable();
                        put_str("\nENABLED\n\n");
                        put_str("\n");
                        EnableMOSFET2();
                  break;

                  case 'd':
                         vsi_disable();
                         put_str("\nDISABLED\n\n");
                         put_str("\n");
                         DisableMOSFET2();
                  break;
                  //    Modulation depth set up
                  case 'i':
                        Flag_max_finder = 1 ;
                        if((mod_depth_serial+step_mod_depth_serial)
< mod_depth_max){ mod_depth_serial +=step_mod_depth_serial; }
                        else{ mod_depth_serial = mod_depth_max;   }
                        vsi_set_mod(mod_depth_serial);
                  break;
                  case 'I':
                        Flag_max_finder = 1 ;
                        if((mod_depth_serial-step_mod_depth_serial)
> 0){ mod_depth_serial -=step_mod_depth_serial; }
                        else{ mod_depth_serial = 0;   }
                        vsi_set_mod(mod_depth_serial);
                  break;

                  // Set modulation step that the system will step
to
                  case 'u':

      if((final_mod_depth_serial+step_mod_depth_serial) <
mod_depth_max){   final_mod_depth_serial +=step_mod_depth_serial; }
                        else{ final_mod_depth_serial =
mod_depth_max;    }
                  break;
                  case 'U':
```

```
                        if((final_mod_depth_serial-
step_mod_depth_serial) > 0){  final_mod_depth_serial-
=step_mod_depth_serial; }
                        else{ final_mod_depth_serial = 0;   }
                  break;

                  //Fundamental frequency of modulation signal
                  case 'f':
                        if((mod_f_freq_serial+step_f_freq_serial) <
mod_f_freq_max){ mod_f_freq_serial +=step_f_freq_serial;}
                        else{ mod_f_freq_serial = mod_f_freq_max; }
                        //vsi_set_freq(mod_f_freq_serial);
                        vsi_set_freq(50.0);
                  break;
                  case 'F':
                        if((mod_f_freq_serial-step_f_freq_serial)
>0){ mod_f_freq_serial -=step_f_freq_serial;}
                        else{ mod_f_freq_serial = 0;}
                        vsi_set_freq(mod_f_freq_serial);
                  break;

                  case 'b':
                        if (real_bus_voltage_serial <
MAX_FIXED_BAND)
real_bus_voltage_serial+=real_step_fixed_band_serial;
SetBusVolts(real_bus_voltage_serial);
                  break;

                  case 'B':
                        if (real_bus_voltage_serial > -
1000/*MIN_FIXED_BAND*/) real_bus_voltage_serial-
=real_step_fixed_band_serial; SetBusVolts(real_bus_voltage_serial);
                  break;

                  case 's':
                        if (real_fixed_band_serial < MAX_FIXED_BAND)
real_fixed_band_serial+=real_step_fixed_band_serial;
SetHystBand(real_fixed_band_serial);
                  break;

                  case 'S':
                        if (real_fixed_band_serial > MIN_FIXED_BAND)
real_fixed_band_serial-=real_step_fixed_band_serial;
SetHystBand(real_fixed_band_serial);
                  break;
                  case 'z':
                        if (real_fixed_band_serial_B <
MAX_FIXED_BAND)
real_fixed_band_serial_B+=real_step_fixed_band_serial;
SetHystBand(real_fixed_band_serial_B);
                  break;

                  case 'Z':
                        if (real_fixed_band_serial_B >
MIN_FIXED_BAND) real_fixed_band_serial_B-
=real_step_fixed_band_serial; SetHystBand(real_fixed_band_serial_B);
                  break;
                  case 'a':
                        if (real_fixed_band_serial_C <
MAX_FIXED_BAND)
```

```
real_fixed_band_serial_C+=real_step_fixed_band_serial;
SetHystBand(real_fixed_band_serial_C);
                  break;

                  case 'A':
                        if (real_fixed_band_serial_C >
MIN_FIXED_BAND) real_fixed_band_serial_C-
=real_step_fixed_band_serial; SetHystBand(real_fixed_band_serial_C);
                  break;

                  case 'n':
                         level_select_point_serial += 10 ;
                  break;

                  case 'N':
                         level_select_point_serial -= 10 ;
                  break;

                  case 'm':
                         real_V3rd_offset_serial+= 1 ;
                  break;

                  case 'M':
                        real_V3rd_offset_serial-= 1 ;
                  break;

                  case 'l':
                         stepping = 1 ;
                         average_offset_serial+= 1 ;
                  break;

                  case 'L':
                         stepping = 0 ;
                         average_offset_serial-= 1 ;
                  break;
                  case 'h':
                              display_help_counter = 0 ;
                              screen_flag = 1 ;
                              help_flag = 1 ;
                              break;

                  case '+':
                              stepping = 1 ;
                  break;
                  case '-':
                              stepping = 0 ;
                  break;

                  case 'p':
                              SPI_Primary_dis_Secondary_en();
                              put_str("ANLGSW: ");
                              putxx(cpld_read(ADD_VERSION));
                              SPI_Primary_en_Secondary_dis();
                              put_str("\n");
                  break;

                  /*case '4':
CPLD_FAST_WRITE(ADD_CAPQEP,0x00);CPLD_FAST_WRITE(ADD_HSPWMDB,0xFA);C
PLD_FAST_WRITE(ADD_SPECIAL,0x00);CPLD_FAST_WRITE(ADD_ANLGSW,0x00);

CPLD_FAST_WRITE(ADD_EVACOMCON,0x01);set_RefMode(SINGLE_AC_OL);vsi_di
```

```
sable(); set_KP_var(real_Kp_serial);step_enable_flag =
FALSE;break;*/


   #ifdef GRAB_INCLUDE
                  case 'g': /* grab interrupt data */
                        GrabClear();
                        GrabStart();
                        GrabRun();
                  break;
                  case 'j':
                        GrabShow();
                  break;
                  //case 'c': /* stop grab display */
                  //    GrabClear();
                  //break;
   #endif
                  case 'H':
                        if(master_slave_mode == 0){
                              put_str("Slave\n");
                        }
                        else{
                              put_str("Master\n");
                        }
                        break;
            }
      }
   } /* end com_keyboard */


   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
   /**
   1 second CPU timer interrupt.

   \author A.McIver
   \par History:
   \li 22/06/05 AM - initial creation (derived from k:startup.c)
   */
   #ifndef BUILD_RAM
   #pragma CODE_SECTION(isr_cpu_timer0, "ramfuncs");
   #endif
   interrupt void isr_cpu_timer0(void)
   {
       static struct
       {
             Uint16
                 usec100,
                   msec,
                   msec10,
                   msec100,
                   sec;
       } i_count =
       {
             0, 0, 0, 0
       };

       /*for (ii=0; ii<WD_TIMER_MAX; ii++)
       {
             if (wd_timer[ii] > 0)
                   wd_timer[ii]--;
```

```
      }*/

            i_count.msec++;
      if (i_count.msec >= 10)
      {
            i_count.msec = 0;
            i_count.msec10++;
            if (i_count.msec10 >= 10)
            {
                  i_count.msec10 = 0;
                  i_count.msec100++;
                  if (i_count.msec100 >= 10)
                  {
                        i_count.msec100 = 0;
                        i_count.sec++;
                        if(i_count.sec >= 5){
                              time.sec5 = 1;
                        }
                        time.sec = 1;
                  }
                  time.sec0_1 = 1;
            }
            time.msec10 = 1;
      }
      time.msec = 1;

      // Acknowledge this interrupt to receive more interrupts from
group 1
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
   } /* end isr_cpu_timer0 */


   /*
=========================================================================
=====
   __Exported_Functions()
   =======================================================================
========== */


   /*
=========================================================================
=====
   __Grab_Functions()
   =======================================================================
========== */
   #ifdef GRAB_INCLUDE

   void GrabInit(void)
   {
       Uint16
            i,j;

       for (i=0; i<GRAB_LENGTH; i++)
       {
            for (j=0; j<GRAB_WIDTH; j++)
            {
                  grab_array[i][j] = 0;
            }
       }
       GrabClear();
```

```
  }

  /* call with index == 0xFFFF for title line
  else index = 0..GRAB_LENGTH-1 for data */
  void GrabDisplay(int16 index)
  {
      Uint16
            i;

      if (index == 0xFFFF)
      {
            put_str("\nindex");
            for (i=0; i<GRAB_WIDTH; i++)
            {
                  put_str("\t");
                  put_d(i);
            }
      }
      else
      {
            put_d(index);
            put_char('\t');
            for (i=0; i<GRAB_WIDTH; i++)
            {
                  //put_char('');
                  putl(grab_array[index][i]);
                  put_char('\t');
            }
      }
      put_str("\n");
  }

  #endif
  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */

  void print_help(void)
  {
  /*  put_str("\th\thelp\n");
      put_str("\tq\tquit\n");
      put_str("\te/d\tenable/disable switching\n");
      put_str("\ti/m\treference magnitude increase/decrease\n");
      put_str("\tl/j\tref freq increase/decrease\n");
      put_str("\t>/<\tswitching freq increase/decrease\n");
      put_str("\tt/T\tintegral reset time slow/fast increase\n");
      put_str("\tv/V\tintegral reset time slow/fast decrease\n");
     put_str("\tr/R\tproportional constant slow/fast increase\n");
      put_str("\tc/C\tproportional constant slow/fast decrease\n");
      //puts("\tg\tto enable grab code and print grab data\n");
      put_str("\tu/b\tincrease/decrease magnitude of step change in
reference\n");
      put_str("\ta/s\tincrease/decrease phase where step is
applied\n");
      put_str("\tp\tenable/disable step change in reference every 5
sec\n");
      put_str("\tf\tenable/disable feed forward\n");
      put_str("\t0\tDC current regulator (using leg A&B)\n");
      put_str("\t1\tSingle phase PI regulator mode (using leg
A&B)\n");
      put_str("\t2\tSingle phase PI regulator mode (using leg A&B)
with back EMF\n");
```

```
      put_str("\t3\tSingle phase PR regulator mode (using leg A&B)
with back EMF\n");
      put_str("\t4\t3 phase PI regulator mode \n");
      put_str("\t5\t3 phase PI regulator mode with back EMF\n");
      put_str("\t6\t3 phase DQ regulator mode with back EMF\n");
      put_str("\t7\t3 phase PR regulator mode with back EMF\n");
      */
  } /* end print_help */

  void init_dac_mini(void)
  {
      // Set SPI mode
      spi_init(MODE_DAC);
      // Initialise DAC
      dac_init();
      // Set internal reference
      dac_set_ref(DAC_MODULE_D1,DAC_INT_REF);
      dac_set_ref(DAC_MODULE_D2,DAC_INT_REF);
      // Power up DAC
      dac_power_down(DAC_MODULE_D1,0x0F);
      dac_power_down(DAC_MODULE_D2,0x0F);
      // Write to half voltage
      dac_write(DAC_MODULE_D1,DAC_WRn_UPDn,DAC_ADDR_ALL,2047);
  }

  /*
==========================================================================
===== */
  /* MACROS added by 1001 */
  /*
==========================================================================
===== */
  /*========================================================================
======================
  CPLD_FAST_WRITE()
  ========================================================================
======================*/
  void CPLD_FAST_WRITE(char REGISTER_ADDRESS,char DATA_BYTE)
  {
      GpioDataRegs.GPASET.all = OC_SPI_EN;
      GpioDataRegs.GPASET.all = M_nS;
      GpioDataRegs.GPFCLEAR.all = nCPLD_CS;
      spi_fail_count_cpld = 65535;
      SpiaRegs.SPITXBUF = ((REGISTER_ADDRESS|MINIBUS_WRITE)<<8);
      SpiaRegs.SPITXBUF = ((DATA_BYTE)<<8);
      SpiaRegs.SPITXBUF = ((0x00)&0x00FF)<<8;
      while ((SpiaRegs.SPIFFRX.bit.RXFFST <
3)&&(spi_fail_count_cpld>0))
            {
                  spi_fail_count_cpld--;
            }
            trash_spi_cpld=SpiaRegs.SPIRXBUF;
            trash_spi_cpld=SpiaRegs.SPIRXBUF;
            trash_spi_cpld=SpiaRegs.SPIRXBUF;
            GpioDataRegs.GPFSET.all = nCPLD_CS;
  }

  Uint16 CPLD_FAST_READ(char REGISTER_ADDRESS)
  {
      Uint16
            data_read = 0x00;
```

```
      GpioDataRegs.GPASET.all = OC_SPI_EN;
      GpioDataRegs.GPASET.all = M_nS;
      GpioDataRegs.GPFCLEAR.all = nCPLD_CS;

       spi_fail_count_cpld = 65535;
      SpiaRegs.SPITXBUF = ((REGISTER_ADDRESS|MINIBUS_READ)<<8);
      SpiaRegs.SPITXBUF = ((0x00)<<8);
      SpiaRegs.SPITXBUF = ((0x00)&0x00FF)<<8;
      while ((SpiaRegs.SPIFFRX.bit.RXFFST <
3)&&(spi_fail_count_cpld>0))
            {
                  spi_fail_count_cpld--;
            }
            trash_spi_cpld=SpiaRegs.SPIRXBUF;
            data_read=SpiaRegs.SPIRXBUF;
            data_read=SpiaRegs.SPIRXBUF;
            GpioDataRegs.GPFSET.all = nCPLD_CS;

      GpioDataRegs.GPFSET.all = nCPLD_CS;

      return data_read;
  } /* end cpld_read */
```

B. 4 Foreground C code for Three-level Variable Band HCC

```
  /**
  \file
  \brief VSI Interrupt Service Routine

  This file contains the code for the core interrupt routine for the
Hysteresis Current Regulator.
  This interrupt is the central system for the signal generation and
  measurement. The carrier timer for the VSI generation also
triggers the
  internal ADC conversion at the peak of the carrier. The end of
conversion then
  triggers this interrupt. Its tasks are:

  - Read internal ADC results
  - Perform internal analog averaging and RMS calculations
  - Update VSI phase and switching times

  \par Developed By:
     Creative Power Technologies, (C) Copyright 2009
  \author A.McIver
  \par History:
  \li 23/04/09 AM - initial creation
  */


  // compiler standard include files
  #include <math.h>

  // processor standard include files
  #include <DSP281x_Device.h>
  #include <DSP281x_Examples.h>

  //#include <lib_da2810.h>
```

```
  #include <lib_mini2810.h>
  #include <lib_cpld.h>
  #include <lib_giib.h>

  // common project include files
  //#define AD5624
  #define DAC_SHIFT 4
  #include <dac_ad56.h>

  // local include files
  #include "main.h"
  #include "conio.h"
  #include "vsi.h"
  #include "curreg.h"
  #include "cas.h"



  /*
=========================================================================
=====
  __Definitions()
  =========================================================================
========== */



  /// Shift from internal modulation depth scaling
  #define MOD_SHIFT                       14

  /* the phase is scaled so that one fundamental is 2^32 counts. */
  //#define PHASE_STEP_SC_D               (65536.0*65536.0/SW_FREQ)
                      //Synchronous switching sampling freq
  #define PHASE_STEP_SC_D                 (65536.0*65536.0/SW_FREQ)
                 //Asynchronous switching sampling
  #define PHASE_STEP
      (Uint32)(PHASE_STEP_SC_D*F_FREQ_MIN)

  /// ADC calibration time
  #define ADC_CAL_TIME              0.1 // seconds
  #define ADC_COUNT_CAL             (Uint16)(ADC_CAL_TIME * SW_FREQ)

  /// DC averaging time
  #define ADC_DC_TIME                     0.1 // seconds
  #define ADC_COUNT_DC              (Uint16)(ADC_DC_TIME * SW_FREQ)

  #define   ADC_REAL_SC                   1

  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
  /// RMS scaling
  #define ADC_RMS_PS                4

  #define GRAB_INCLUDE

  #define HIGH 1
  #define LOW 0
  #define ON                              1
  #define OFF                       0
```

```
   #define SYNC_PIN                BIT0
   #define TPB2                    BIT2
   #define TPB1                    BIT1
   #define TPA0                    BIT0

   #define SET_SYNC_PIN()          GpioDataRegs.GPBSET.all =
SYNC_PIN
   #define CLEAR_SYNC_PIN()    GpioDataRegs.GPBCLEAR.all = SYNC_PIN

   #define SET_TPB1()              GpioDataRegs.GPBSET.all = TPB1
   #define CLEAR_TPB1()        GpioDataRegs.GPBCLEAR.all = TPB1

   #define SET_TPB2()              GpioDataRegs.GPBSET.all = TPB2
   #define CLEAR_TPB2()        GpioDataRegs.GPBCLEAR.all = TPB2


   /*
=========================================================================
=====
   __Macros()
   =======================================================================
========== */

   /// Disable VSI switching
   #define VSI_DISABLE()       EvaRegs.ACTRA.all = 0x0000

   /// Enable VSI switching
   #define VSI_ENABLE()        EvaRegs.ACTRA.all = 0x0666
       //    output pin 1 CMPR1 - active high
       //    output pin 2 CMPR1 - active low
       //    output pin 3 CMPR2 - active high
       //    output pin 4 CMPR2 - active low
       //    output pin 5 CMPR3 - active high
       //    output pin 6 CMPR3 - active low

   /// Enable VSI for single phase operation
   #define VSI_ENABLE_1P()         EvaRegs.ACTRA.all = 0x0066
       //    output pin 1 CMPR1 - active high
       //    output pin 2 CMPR1 - active low
       //    output pin 3 CMPR2 - active high
       //    output pin 4 CMPR2 - active low
       //    output pin 5 force low
       //    output pin 6 force low

   /// Turn low side devices on full for charge pump starting
   #define VSI_GATE_CHARGE()   EvaRegs.ACTRA.all = 0x00CC

   /*
=========================================================================
=====
   __Controller_Macros()
   =======================================================================
========== */
   /*************************************************************
**************
   * This function perform a PI loop calculation, the following
parameters needed
   * to be provided to the function in pointer form:
   * REF = reference current in signed int format
   * ERR = error, this variable should be in signed int format, it
will be wrote to
```

```
   * MEA = measured current in signed int format
   * KP = proportional constant, in signed int format
   * KI = integral constant, in signed int format
   * PROP_NOW = the proportional part of PI loop for this pass of the
loop in long int
   * INT_NOW = the integral action calculated for this pass of the
loop in long int
   * INT_TOTAL = the integral action in total (memory of integrator)
in long int
   * FF = the amount of feedfoward to used in signed int
   * COMM_OUT = output command in signed int
   * Note on PI code:
   * Multiple by 1<<PROP_DISCARD_BITS is to compensate for the
division of the same number
   * at the calculation for proportional time constant
   *   The division by 1<<PROP_DISCARD_BITS allow for the use of
higher proportional constant
   *   then otherwise possible, there should not be significant loss
to accuracy provided
   *   that the number for PROP_DISCARD_BITS is smallish, the
calculation for lost of accuracy
   *   is listed in comment above
   * Contain desaturation and clamping code.
   * 256 * 128 give 32767 which is 200% modulation
   ****************************************************************
************/
   /*================================================================
======================
   PI_LOOP()
   ================================================================
======================*/
   #define PI_LOOP(ERR, REF, MEA, KP, KI, PROP_NOW, INT_NOW,
INT_TOTAL,FF, COMM_OUT)  {\
       ERR = REF - MEA; \
       PROP_NOW = ((long)(ERR)
*(long)(KP))*(long)(1<<PROP_DISCARD_BITS); \
       INT_NOW = ((long)(ERR)
*(long)(KI))*(long)(1<<INT_DISCARD_BITS); \
       INT_TOTAL += INT_NOW; \
       COMM_OUT = ((INT_TOTAL + PROP_NOW)>>16)+FF; \
       if(COMM_OUT > MODMAX){ \
             COMM_OUT = MODMAX; \
             INT_TOTAL -= INT_NOW; \
       } \
       else if (COMM_OUT < (-1*MODMAX)){ \
             COMM_OUT = -1*MODMAX; \
             INT_TOTAL -= INT_NOW; \
       } \
   }

   //Delta transform definitions
   #define LOG2_ALPHA_0                    11

   #define ISS
       (LOG2_ALPHA_0-6)
   #define LOG2_1_ON_DELTA                 2
   #define w_c_f                                                0.1
   #define PR_INTERNAL_CLAMP         10000
   #define PR_SCALING
       (int)(65536.0/I_NOM/4.0+0.5)                    //Scaling
constant to per unitised the controller output.  Division of 4 is
```

```
there because of the two left shift in the subsequence period
calculation

   #define   LOG2_ALPHA_0_COM        11
   #define ISS_FILT                                    0


   /*=================================================================
======================
   SIN_TABLE_UPDATE()
   =================================================================
======================*/
   //Update INDEX to point to the current location of sin table
   #define SIN_TABLE_UPDATE(PHASE, INDEX){ \
       PHASE += phase_step;    \
       INDEX = (PHASE>>22)|0x001;    \
   }

   /*=================================================================
======================
   SIN_TABLE_READ()
   =================================================================
======================*/
   //Just read the value of sin table at location indicated by INDEX
   #define SIN_TABLE_READ(INDEX, VAL){                 \
       val_lo = sin_table[INDEX];    \
       val_diff = sin_table[INDEX+2] - val_lo;   \
       VAL = (val_lo +
(int16)(((int32)(INDEX&0x007F)*(int32)val_diff)>>7))<<1;    \
   }


   /*=================================================================
======================
   DSP_SET_SPI_SLAVE()
   =================================================================
======================*/
   // Setup the SPI core to function as a slave to recieve date
(note, this is just the core, does not include setup for
   // Mini2810 and GIIB, so if data needed to be recieve from GIIB,
more setup is needed)
   #define DSP_SET_SPI_SLAVE(){      \
       GpioDataRegs.GPFCLEAR.all = BIT3;

             \
       SpiaRegs.SPICCR.all = 0x0000;                   /*Set SPI to
reset state*/                  \
       SpiaRegs.SPICCR.bit.CLKPOLARITY = 0;

       \
       SpiaRegs.SPICCR.bit.SPICHAR = 7;    /* 8 bit characters*/
                                   \



             \
       SpiaRegs.SPICTL.all = 0x0000;

                   \
```

```
      SpiaRegs.SPICTL.bit.CLK_PHASE = 1;

      \
      SpiaRegs.SPICTL.bit.MASTER_SLAVE = 0; /*Slave*/
                                                \
      SpiaRegs.SPICTL.bit.TALK = 0;             /* enable
transmission*/                              \
      SpiaRegs.SPICTL.bit.SPIINTENA = 0;              /*Enable SPI
interrupt*/       \



            \
      SpiaRegs.SPIBRR = 4;                      /*
37.5/(SPIBRR+1)MHz      */
      \



            \
      SpiaRegs.SPIFFTX.all = 0x0000;

                  \
      SpiaRegs.SPIFFTX.bit.SPIFFENA = 1;  /* enable tx fifo*/
                                    \
      SpiaRegs.SPIFFTX.bit.TXFIFO = 1;          /* Enable Transmit
channel*/         \
      SpiaRegs.SPIFFTX.bit.SPIRST = 1;          /* enable SPI*/
                                                \



            \
      SpiaRegs.SPIFFRX.all = 0;

                                \
      SpiaRegs.SPIFFRX.bit.RXFFIL = 3;    /* interrupt on 3 bytes in
fifo*/      \
      SpiaRegs.SPIFFRX.bit.RXFIFORESET = 1;  /* Enable Receive
channel*/   \



            \
      SpiaRegs.SPIFFCT.all = 0x00;

                  \



            \
      SpiaRegs.SPICCR.bit.SPISWRESET = 1; /* relinquish from reset*/
                  \
   }


   /*=============================================================
======================
   DSP_SET_SPI_MASTER()
   =============================================================
======================*/
```

```
  // Setup the SPI core to function as a master to ready to send
data (note, this is just the core, does not include setup for
  // Mini2810 and GIIB, so if data needed to be sent out to GIIB,
more setup is needed)
  #define DSP_SET_SPI_MASTER(){     \
      GpioDataRegs.GPFCLEAR.all = BIT3;

            \
      SpiaRegs.SPICCR.all = 0x0000;                   /*Set SPI to
reset state*/                  \
      SpiaRegs.SPICCR.bit.CLKPOLARITY = 0;

      \
      SpiaRegs.SPICCR.bit.SPICHAR = 7;    /* 8 bit characters*/
                                  \



            \
      SpiaRegs.SPICTL.all = 0x0000;

                  \
      SpiaRegs.SPICTL.bit.CLK_PHASE = 1;

      \
      SpiaRegs.SPICTL.bit.MASTER_SLAVE = 1; /*Slave*/
                                                \
      SpiaRegs.SPICTL.bit.TALK = 1;             /* enable
transmission*/                              \
      SpiaRegs.SPICTL.bit.SPIINTENA = 0;              /*Enable SPI
interrupt*/       \



            \
      SpiaRegs.SPIBRR = 4;                        /*
37.5/(SPIBRR+1)MHz      */
      \



            \
      SpiaRegs.SPIFFTX.all = 0x0000;

                  \
      SpiaRegs.SPIFFTX.bit.SPIFFENA = 1;  /* enable tx fifo*/
                                  \
      SpiaRegs.SPIFFTX.bit.TXFIFO = 1;          /* Enable Transmit
channel*/         \
      SpiaRegs.SPIFFTX.bit.SPIRST = 1;          /* enable SPI*/
                                                \



            \
      SpiaRegs.SPIFFRX.all = 0;

                              \
      SpiaRegs.SPIFFRX.bit.RXFFIL = 3;    /* interrupt on 3 bytes in
fifo*/      \
```

```
      SpiaRegs.SPIFFRX.bit.RXFIFORESET = 1;  /* Enable Receive
channel*/   \



            \
      SpiaRegs.SPIFFCT.all = 0x00;

                  \



            \
      SpiaRegs.SPICCR.bit.SPISWRESET = 1; /* relinquish from reset*/
                  \
   }


   /*=================================================================
======================
   SLAVE_RECIEVE_MASTER()
   =================================================================
======================*/
   #define SLAVE_RECIEVE_MASTER(){\
      SET_OC_SPI_EN();

                                    \
      DISABLE_CPLD();
                                          \
      CLEAR_SPI_MASTER();
                                          \
      DSP_SET_SPI_SLAVE();
                               \
      //SET_TP11();
                                                      \
      wait = 0;
                                                      \
      while(SpiaRegs.SPIFFRX.bit.RXFFST != NO_SPI_CHAR){

                                    \
            wait++;
                                                            \
            /*Wait for message to arrive, if they don't arrive flag
error */
      \
            if(wait >=300){
                                          \
                  char_not_recieved = 1;
                        \
                  break;
                                                      \
            }
                                                                  \
      }
      \
      //CLEAR_TP11();


                                                \
```

```
      \
      checksum_test_tmp = 0;                              /*Reset
checksum summing*/
                                                \
      char_no = SpiaRegs.SPIFFRX.bit.RXFFST;

                                                                  \
      for(i = 0; i<NO_SPI_CHAR; i++){


            \
            tmp = SpiaRegs.SPIRXBUF;


                        \
            tmp = tmp & 255;


                                       \
            rec_char_array[i]=tmp;        /*Extracting the lower 8
bits only*/
            \
            checksum_test_tmp += tmp;           /*Calculate
checksum*/
                                                \
      }

                                                                        \
      checksum_test_tmp =checksum_test_tmp & 255;
      /*Only lower 8 bits of checksummatter*/   \
      \
      if((checksum_test_tmp & 255) != 0){             /* Flagging
Message recieved is corrupted */          \
            checksum_OK_1st = 0;


                               \
      }

      \
      else{
                  /*If checksum OK, process message*/
                                                      \
            checksum_OK_1st = 1;


                               \
            rec_master_status_1st = (Uint16)rec_char_array[0];

                                       \
            rec_master_status_2nd = (Uint16)rec_char_array[1];

                                       \
            index_sin = (Uint16)rec_char_array[2] +
(((Uint16)rec_char_array[3])<<8);
      \
            mod_targ = (int16)((Uint16)rec_char_array[4] +
(((Uint16)rec_char_array[5])<<8));             \
```

```
            Kp_i = rec_char_array[6] + (rec_char_array[7]<<8);

                                    \
            Ki_i = rec_char_array[8] + (rec_char_array[9]<<8);

                                    \
            feed_forward_mag = rec_char_array[10] +
(rec_char_array[11]<<8);
                        \
      }



      \

      \
      if(rec_master_status_1st & SW_ENABLE){

                                                                  \
            is_switching = 1;


                                    \
      }else{


                                                                  \
            is_switching = 0;


                                    \
      }

      \
      \
      /*Finish listening switch to sending mode ready to send once
timer compare is triggered*/  \
      CLEAR_OC_SPI_EN();


                                    \
      DSP_SET_SPI_MASTER();


                               \
      spi_set_mode(MODE_CPLD);


                        \
      wait = 0;


                                                            \
      while(wait < 20){


                                    \
            wait++;


                                                                  \
```

```
      }
                                                                        \
      WriteControlLatch(!CTRL_SPI_SLAVE, LOAD);
      /*Set GIIB SPI direction to sending mode for next half of
interrupt cycle*/\
   }


   /*=================================================================
======================
   SLAVE_SEND()
   =================================================================
======================*/
   #define SLAVE_SEND(){


                              \
      SET_OC_SPI_EN();

                                    \
      slave_status_1st = 100;


                             \
      slave_status_2nd = 0;


                              \
      curr_comp = 42500;


                                    \
      /*Composing message */


                             \
      sync_message[0]  = slave_status_1st;

                                                                        \
      sync_message[1]  = slave_status_2nd;

                                                                        \
      sync_message[2]  = curr_comp;


            \
      sync_message[3]  = curr_comp>>8;


      \
      /* Generating modular sum*/


                \
      checksum = 0;


                                                  \
      for(i = 0; i < 4; i++){
```

```
            \
      checksum += (Uint16)sync_message[i];

                                                                    \
}



\
sum = checksum;


                                        \
checksum = ~checksum;


                          \
checksum += 1;


                                        \
sync_message[4] = checksum;


            \
sync_message[5]  = 0;


                          \
sync_message[6]  = 0;


                          \
sync_message[7]  = 0;


                          \
sync_message[8] = 0;


                          \
sync_message[9] = 0;


                          \
sync_message[10] = 0;


                          \
sync_message[11] = 0;


                          \
sync_message[12] = 0;


                          \
sync_message[13] = 0;
```

```
                                \
sync_message[14] = 0;

                                \
sync_message[15] = 0;

                                \
sum = checksum + sum;

                                \

\
DSP_SET_SPI_MASTER();

                                \
SET_SPI_MASTER();

                                    \
DISABLE_CPLD();

                                        \
wait = 0;

                                                \
while(wait < 50){

                                    \
      wait++;

                                                        \
}


\


\
//SET_TP11();

                                                \
for(i = 0; i < NO_SPI_CHAR; i++){

\
      spi_putc(sync_message[i]);

            \
```

```
      }



      \
      //CLEAR_TP11();

                                                \



      \
      CLEAR_OC_SPI_EN();


                                    \
      /*Once sending is done switch to listening mode*/

                                          \
      DSP_SET_SPI_MASTER();


                                 \
      spi_set_mode(MODE_CPLD);


                         \
      wait = 0;


                                                      \
      while(wait < 10){


                                    \
            wait++;


                                                            \
      }

      \
      WriteControlLatch(CTRL_SPI_SLAVE, LOAD);
      /*Set GIIB SPI direction to listening mode to get ready for
next interrupt cycle*/\
   }


   /*=================================================================
======================
   SLAVE_RECIEVE_SLAVE()
   =================================================================
======================*/
   #define SLAVE_RECIEVE_SLAVE(){


            \
      SET_OC_SPI_EN();
```

```
                                    \
DISABLE_CPLD();

                                          \
CLEAR_SPI_MASTER();

                                    \
DSP_SET_SPI_SLAVE();

                               \
//SET_TP11();

                                                    \
wait = 0;

                                                    \
while(SpiaRegs.SPIFFRX.bit.RXFFST !=  NO_SPI_CHAR){
                                    \
      wait++;

                                                          \
      if(wait >=300){

                                          \
            char_not_recieved = 1;

                  \
            break;

                                                          \
      }


\
}


\
//CLEAR_TP11();

                                                \
checksum_test_tmp = 0;

                  \
```

```
char_no = SpiaRegs.SPIFFRX.bit.RXFFST;

                                                                \
for(i = 0; i<NO_SPI_CHAR; i++){

      \
      tmp = SpiaRegs.SPIRXBUF;

                  \
      tmp = tmp &255;

                                        \
      rec_char_array[i]=tmp & 255;

      \
      checksum_test_tmp += tmp;

                  \
}



\
checksum_test_tmp =checksum_test_tmp & 255;
/*Only lower 8 bits matter*/                                    \



\
if((checksum_test_tmp & 255) != 0){

                                                                \
      checksum_OK_2nd = 0;

                        \
}



\
else{

                                                                \
      checksum_OK_2nd = 1;

                        \
}



\
```

```
      /*Finish listening switch to sending mode ready to send once
timer compare is triggered*/\
      CLEAR_OC_SPI_EN();


                                        \
      DSP_SET_SPI_MASTER();


                                  \
      spi_set_mode(MODE_CPLD);


                           \
      wait = 0;


                                                          \
      while(wait < 10){


                                        \
            wait++;


                                                                \
      }



      \
      WriteControlLatch(!CTRL_SPI_SLAVE, LOAD);
      /*Set GIIB SPI direction to sending mode for start of next
interrup cycle*/\
   }
   /*=====================================================================
======================
   SPI Sends data
   =====================================================================
======================*/
   #define MASTER_SEND_DATA(SPI_DATA)\
   {\
       i       = 0;\
       while (SpiaRegs.SPISTS.bit.BUFFULL_FLAG == 1);\
       SpiaRegs.SPITXBUF = SPI_DATA<<8;\
       while ((SpiaRegs.SPIFFRX.bit.RXFFST == 0)&&(i<65000))\
       {\
             i++;\
       }\
       SpiaRegs.SPIRXBUF;\
   }\
   /*=====================================================================
======================
   DAC1_FAST_WRITE()
   =====================================================================
======================*/
   #define DAC1_ZAKI_WRITE(DAC_COMMAND,DAC_ADDRESS,DAC_DATA)\
   {\
       GpioDataRegs.GPDSET.all = nDAC2;\
       GpioDataRegs.GPDCLEAR.all = nDAC1;\
       spi_fail_count_dac1 = 65535;\
```

```
      SpiaRegs.SPITXBUF = (DAC_COMMAND|DAC_ADDRESS)<<8;\
      SpiaRegs.SPITXBUF = ((((DAC_DATA<<4)>>8)&0x00FF)<<8);\
      SpiaRegs.SPITXBUF = ((DAC_DATA<<4)&0x00FF)<<8;\
      while ((SpiaRegs.SPIFFRX.bit.RXFFST <
3)&&(spi_fail_count_dac1>0) )\
            {\
                  spi_fail_count_dac1--;\
            }\
      GpioDataRegs.GPDSET.all = nDAC1;\
      GpioDataRegs.GPDSET.all = nDAC2;\
  }\
  /*=================================================================
======================
  DAC1_FAST_WRITE()
  =================================================================
======================*/
  #define DAC1_FAST_WRITE(DAC_COMMAND,DAC_ADDRESS,DAC_DATA)\
  {\
      GpioDataRegs.GPDSET.all = nDAC2;\
      GpioDataRegs.GPDCLEAR.all = nDAC1;\
      spi_fail_count_dac1 = 65535;\
      SpiaRegs.SPITXBUF = (DAC_COMMAND|DAC_ADDRESS)<<8;\
      SpiaRegs.SPITXBUF = ((((DAC_DATA<<4)>>8)&0x00FF)<<8);\
      SpiaRegs.SPITXBUF = ((DAC_DATA<<4)&0x00FF)<<8;\
      while ((SpiaRegs.SPIFFRX.bit.RXFFST <
3)&&(spi_fail_count_dac1>0) )\
            {\
                  spi_fail_count_dac1--;\
            }\
            trash_spi_dac1=SpiaRegs.SPIRXBUF;\
            trash_spi_dac1=SpiaRegs.SPIRXBUF;\
            trash_spi_dac1=SpiaRegs.SPIRXBUF;\
      GpioDataRegs.GPDSET.all = nDAC1;\
      GpioDataRegs.GPDSET.all = nDAC2;\
  }\
  /*=================================================================
======================
  DAC2_FAST_WRITE()
  =================================================================
======================*/
  #define DAC2_FAST_WRITE(DAC_COMMAND,DAC_ADDRESS,DAC_DATA)\
  {\
      GpioDataRegs.GPASET.all = OC_SPI_EN;\
      GpioDataRegs.GPASET.all = M_nS;\
      GpioDataRegs.GPDSET.all = nDAC1;\
      GpioDataRegs.GPDCLEAR.all = nDAC2;\
      spi_fail_count_dac2 = 65535;\
      SpiaRegs.SPITXBUF = (DAC_COMMAND|DAC_ADDRESS)<<8;\
      SpiaRegs.SPITXBUF = ((((DAC_DATA<<4)>>8)&0x00FF)<<8);\
      SpiaRegs.SPITXBUF = ((DAC_DATA<<4)&0x00FF)<<8;\
      while ((SpiaRegs.SPIFFRX.bit.RXFFST <
3)&&(spi_fail_count_dac2>0) )\
            {\
                  spi_fail_count_dac2--;\
            }\
            trash_spi_dac2=SpiaRegs.SPIRXBUF;\
            trash_spi_dac2=SpiaRegs.SPIRXBUF;\
            trash_spi_dac2=SpiaRegs.SPIRXBUF;\
      GpioDataRegs.GPDSET.all = nDAC2;\
      GpioDataRegs.GPDSET.all = nDAC1;\
  }\
```

```
   /*
================================================================================
   __Types()
   ================================================================================ */

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

   /// Internal ADC storage type
   /** 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.
   The vout and iout channels are for DC measurements of the VSI
outputs when it
   is producing a DC output. */
   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 averaging
             count_dc_bak, ///< background copy of DC counter
             flag_cal, ///< flag set to trigger background
calibration averaging
             flag_rms, ///< flag set to trigger background RMS
averaging
             flag_dc; ///< flag set to trigger background DC
averaging
       type_adc_ch
             A0, ///< ADC channel A0
   //        A1, ///< ADC channel A1
   //        A2, ///< ADC channel A2
   //        A3, ///< ADC channel A3
   //        A4, ///< ADC channel A4
   //        A5, ///< ADC channel A5
             B0, ///< ADC channel B0
```

```
  //        B1, ///< ADC channel B1
  //        B2, ///< ADC channel B2
  //        B3, ///< ADC channel B3
  //        B4, ///< ADC channel B4
  //        B5, ///< ADC channel B5
            yHA, ///< bank A high reference
            yLA, ///< bank A low reference
            yHB, ///< bank B high reference
            yLB; ///< bank B low reference
  } type_adc_int;


  /*
==========================================================================
=====
  __Variables()
  ==========================================================================
========== */

  // state machine level variables
  Uint16
      vsi_status = 0,               /// Status of VSI system
      int_count = 0,
      is_switching = 0,       // flag set if PWM switching is active
      vsi_counter = 0;        // counter for timing VSI regulation
events

  //Timer period and switching frequency related variable
  //Initialized to default value as set by #define values at top of
this file
  Uint16
      sw_freq = SW_FREQ,
      period_2 = PERIOD_2,
      period = PERIOD;
  Uint32
      PHASE_STEP_SC = PHASE_STEP_SC_D     ;

  /// Maximum VSI switching time in clock ticks
  int16
      MAX_TIME    =     (int16)(PERIOD_2-6) ;

  int16
      V_Asat=0, V_Bsat=0, V_Csat=0;

  double
      Ref_freq_float = INIT_FF;

  // PWM Timer interrupt variables

  // 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
  // WYK note: Sin table contain 1024 entire with peak value of +-
16384
  int16
      *sin_table = (int16 *)0x003FF000, // pointer to sine table in
boot ROM
```

```
      *cos_table = (int16 *)0x003FF100, // pointer to cos table in
boot ROM
      phase_offset,                  // round off amount from sine
lookup
      val_diff,                      // interpolation temp variable
      val_lo,                              // interpolation temp
variable
      sin_val,                       // interpolated sine table value
      cos_val,                       // interpolated cosine table
value
      sin_PI_on_3_val,
      sin_PI_on_6_val,
      sin_2PI_on_3_val,
      sin_4PI_on_3_val,
      sin_pf_val,
      sin_PI_on_2_val,


      cos_PI_on_3_val,
      ia_err, ib_err, ic_err,
      DC_val;

  Uint32
      phase_step = PHASE_STEP,// change in phase angle each
interrupt
      phase = 0L;                    // running phase angle  (2^32 ==
360degrees)

  //Calculate phase offset to initialized phase value to enable
reading of sin table to generate various differernt trignometrey
lookup needed
  Uint32
      phase_sin = (long)65536.0*0.0* 65536.0,
      phase_sin_PI_on_3 = (long) 65536.0*(1.0/6.0) * 65536.0,
      phase_sin_PI_on_6 = (long)65536.0*(1.0/12.0) * 65536.0,
      phase_sin_2PI_on_3 = (long) 65536.0/3.0 * 65536.0,
      phase_sin_4PI_on_3 = (long) 65536.0*(2.0/3.0) * 65536.0,
      phase_sin_PI_on_2 = (long) 65536.0/2.0 * 65536.0,

      phase_sin_A_alpha = (long) 65536.0 * (1.0/20.0) * 65536.0,


      phase_cos = (long) 65536.0*(0.25) * 65536.0,
      phase_cos_PI_on_3 = (long) 65536.0*(0.25+1.0/6.0) * 65536.0,
      phase_pf_sin ;

  Uint16                       // index into sine look-up table
(phase >> 22), this give a 10 bit number which can be used to read
sin table
      index = 0,
      index_sin = 0,
      index_sin_PI_on_3 = 0,
      index_sin_PI_on_6 = 0,

      index_sin_PI_on_2 = 0,

      index_sin_2PI_on_3 = 0,
      index_sin_4PI_on_3 = 0,
      index_sin_pf = 0,
      index_cos = 0,
      index_cos_PI_on_3 = 0;
```

```
  int16
      V_A,                          // demanded voltages
      t_A,                          // switching times
      t_B,
      t_C,
      Voff,
      Voff_prev,
      Voff_int,
      Voff_int_prev,
      Voff_scaled,                  //3rd harmonic offset
      Voff_scaled_prev,             //3rd harmonic offset
      mod_targ = 0,                 // target modulation depth
      mod_ref = 0;                  // background reference mod
depth

  /// fault variables
  Uint16
      detected_faults = 0;    // bits set for faults detected
(possibly cleared)


  /** @name Internal ADC Variables */
  //@{
  type_adc_int
      adc_int =
      {
            0, // count_cal
            0, // count_rms
            0, // count_rms_bak
            0, // count_dc
            0, // count_dc_bak
            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
         }, // #A0
  //        { 0, 0, 0L, 0L, 0L, 0L, 0.0 }, // #A1
  //        { 0, 0, 0L, 0L, 0L, 0L, 0.0 }, // #A2
  //        { 0, 0, 0L, 0L, 0L, 0L, 0.0 }, // #A3
  //        { 0, 0, 0L, 0L, 0L, 0L, 0.0 }, // #A4
  //        { 0, 0, 0L, 0L, 0L, 0L, 0.0 }, // #A5
            { 0, 0, 0L, 0L, 0L, 0L, 0.0 }, // #B0
  //        { 0, 0, 0L, 0L, 0L, 0L, 0.0 }, // #B1
  //        { 0, 0, 0L, 0L, 0L, 0L, 0.0 }, // #B2
  //        { 0, 0, 0L, 0L, 0L, 0L, 0.0 }, // #B3
  //        { 0, 0, 0L, 0L, 0L, 0L, 0.0 }, // #B4
  //        { 0, 0, 0L, 0L, 0L, 0L, 0.0 }, // #B5
            { 0, 0, 0L, 0L, 0L, 0L, 0.0 }, // yHA
            { 0, 0, 0L, 0L, 0L, 0L, 0.0 }, // yLA
            { 0, 0, 0L, 0L, 0L, 0L, 0.0 }, // yHB
            { 0, 0, 0L, 0L, 0L, 0L, 0.0 }, // yLB
      };
```

```
  // ADC calibration variables
  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
  double
      cal_gain_A, cal_gain_B,
      cal_offset_A, cal_offset_B;
  //@}

  //ADC value holder
  int16
      I_res_A,
      I_res_B,
      I_res_C,
      I_err_A,
      I_err_B,
      I_err_C,
      I_err_A_prev,
      I_err_B_prev,
      I_err_C_prev,
      I_err_err_prev,
      I_err_A_deriv,
      I_err_B_deriv,
      I_err_C_deriv,
      I_err_A_deriv_prev,
      I_err_B_deriv_prev,
      I_err_C_deriv_prev,
      filtered_output,
      filtered_output_prev,
       I_err_A_deriv_alpha,
      VAC_A,                        //Voltage measuremnet of phase A
of grid to natural
      VAC_B,                        //Voltage measuremnet of phase B
of grid to natural
      V_sw,
      V_half_bus_1,
      V_half_bus_2,
      Vnp,
      Vnp_prev,
      Vdc_2 ,
      Vdc;


  //Reference mode for current regulator
  int16 refMode = THREE_PHASE_OL;

  signed int I_ref_Peak_AB = 0 , I_ref_Peak_AB_prev = 0 ;
  signed int I_ref_Peak_AB_tr = 0;
  //Control loop variables
  //Stationary frame PI regulator internal variables
  int16
      I_ref_A = 0,
      I_ref_B = 0,
      I_ref_B_prev = 0,
      I_ref_C = 0,
      I_ref_A_1 = 0,
```

```
      I_ref_A_1_up = 0,
      I_ref_A_1_low = 0,
      I_ref_A_1_prev = 0,
      I_ref_B_1 = 0,
      I_ref_B_1_up = 0,
      I_ref_B_1_low = 0,
      I_ref_C_1 = 0,
      I_ref_C_1_up = 0,
      I_ref_C_1_low = 0,
      I_ref_C_1_prev = 0,

      polarity_change_A = 0 ,
      polarity_change_A_trans = 0 ,
      polarity_change_A_up = 0 ,
      polarity_change_A_down = 0 ,

      polarity_change_B = 0 ,
      polarity_change_B_trans = 0 ,
      polarity_change_B_up = 0 ,
      polarity_change_B_down = 0 ,

      polarity_change_C = 0 ,
      polarity_change_C_trans = 0 ,
      polarity_change_C_up = 0 ,
      polarity_change_C_down = 0 ,

      bus_voltage                                    = 0 ,
      bus_voltage_low                          = 0 ,
      bus_voltage_up                           = 0 ,
      bus_voltage_trans                  = 0 ,

      I_ref_C_2 = 0,
      I_ref_D = 0,
      Vavg_virt = 0,
      ia_err, ib_err, ic_err,
      sw_a = LOW,
      sw_a_old = LOW,
      sw_b = LOW,
      sw_b_old = LOW,
      sw_c = LOW,
      sw_c_old = LOW,
      burn = 0, actra_on = 0, actra_off = 0;

  int32
      err_i_prop_A=0,
      err_i_int_now_A=0,
      err_i_int_total_A=0;
  int32
      err_i_prop_B=0,
      err_i_int_now_B=0,
      err_i_int_total_B=0;

  int16
      Error_I_A,
      Error_I_B,
      Command_A,                         //Controller output
variable
      Command_B;                         //Controller output
variable

  //Hysteresis Controller Parameter
```

```
  int16
      ia_err, ib_err, ic_err,
      ia, ib,     ic,
      ia_cmd, ib_cmd, ic_cmd,temp_band,beta;

  //Controller tuning in integer form used by controller calculation
  int16
      Kp_i = 0,                          //Proportional constant in
ADC count and timer count
      Ki_i = 0,                          //Integral constant in ADC
count and timer count
      MODMAX = 8191,          //Maximum modulation index 100%
modulation
      ADC_offset = ADC_OFFSET,
      // add_phase is the size of step jump in phase, the
extra_phase should be added to
      // the phase for reading the sintable, as it is a record of
all the step change in phase
      // requested so far
      add_phase = 0;
  long o = 0 , j = 0 , z = 0;
  //Delta transform variables
  double
      delta, one_on_delta, w_c, w_0, Ts;
  //S domain transfer function of P+Resonant controller
  //Function of form H(s) = (bs_0*s^2 + bs_1*s + bs_2)/(as_0*s^2 +
as_1*s + as_0)
  double
      bs_0, bs_1, bs_2, as_0, as_1, as_2;
  //Z domain transfer function of P+Resonant controller
  //Function of form H(z) = (bz_0 + bz_1*z^-1 +bz_0*z^-2)/(az_0 +
az_1*z^-1 +az_0*z^-2)
  double
      bz_0, bz_1, bz_2, az_0, az_1, az_2;
  //Delta domain transfer function of P+Resonant controller in
floating point form
  //Function of form H(d) = (beta_0_f + beta_1_f*d^-1 + beta_2_f*d^-
2)/(1 + alpha_1_f*d^-1 + alpha_2_f*d^-2)
  double
      alpha_1_f, alpha_2_f, beta_0_f, beta_1_f, beta_2_f;
  int16
      beta_0, beta_1, beta_2, alpha_0, alpha_1, alpha_2;
  double
      Ki_i_f, Kp_i_f;
  //Interal variables for phase A of P+R controller
  long
      Error_I_L_A=0;
  int
      s0_1_fp_A = 0, s0_0_fp_A = 0,
      s1_1_fp_A = 0, s1_0_fp_A = 0,
      s2_1_fp_A = 0, s2_0_fp_A = 0;
  long
      branch1_A=0, branch2_A=0, branch3_A=0, branch4_A=0,
branch5_A=0;


  Uint16
      togg = 0,
      sw = 0, Inom,
      step_tog=0;                        //Toggle channel 2 of Dig
IO for step change in reference
```

```
  //Interface variables used to recieve controller loop parameters
from background
  //Controller loop turning parameters in real floating pointer
number from background
  double
      real_KP           =KP_INIT,
      real_TINT =TINT_INIT,
      real_Hband = BMAX ;

  //For step change in reference in AC reference modes
  int16 count_from_zero_for_step = 0;
  int16 new_mod_targ = 0;
  int16 step_ref_request = 0;
  int16 step_phase_request = 0;
  int16 prev_sin_table_sign = 0;
  int16 step_0_ref_A = 0;

  //Feedforward + bus compensation related variables
  int FFenable = DISABLE;
                 //Feedforward status
  long FF_amount_A = 0;
  long FF_amount_B = 0;
  long ZERO = 0;
  int count_per_A = ADC_I1_SC_SCALED;
      //Scaled version of Amp per count used in calculation
  int VBUS = 0;                            //DC bus voltage
  int cond = 0;
  int one_on_vbus =0;
  int inverse_INOM = (int)((1.0/(float)I_NOM)*(long)(1l<<16));
  int DAC_out = 0;
  int fundament_frequency = 0;                         //Fundamental
frequency multiple by 256
  //int inverse_bus_v_array[BUS_ARRAY_SIZE];
      //Array contain the inverse of bus voltage multiple by a
constant for bus compensation calculation

  /* Zero crossing variables */
  unsigned int
      in_sync,
      ZX_in_sync,
      ZX_state,
      ZX_count,
      ZX_seen,
      ZX_cycles,
      ZX_sum;
  signed int
      ZX_time,
      ZX_time_phase,
      ZX_phase_scale,
      ZX_phase_err,
      ZX_err_sum;
  int phase_trim = 0;


  int16 loop_back_character = 0;
```

```
   int16 carrier_capture = 0;
   int16 carrier_error = 0;
   int16 carrier_adjust = 0;
   int16 period_load = 0;

   int16 carrier_count = 0;
   int16 carrier_sync = 0;

   int16 carrier_offset = 50;
   int16 carrier_allowed_error = 4;

   //Master slave mode select.  0 being slave, 1 being master
   int16 master_slave_mode = 0;
   int16 loop_no = 0;

   int16 adjust_count = 0;

   int16 sync_method = 0;


   int16 UF_P = 0;

   int16 adc_ready_flag = 0;
   int16 timer1_cmp_flag = 0;

   //Internal variable for message to be send via spi
   Uint16 sync_message[16];

   //Variables for sync message
   Uint16 mas_status_1st = 0;
   Uint16 mas_status_2nd = 0;
   Uint16 slave_status_1st = 0;
   Uint16 slave_status_2nd = 0;
   Uint16 curr_comp = 0;
   Uint16 feed_forward_mag = 0;

   //Array containing test SPI characters
   Uint16 rec_char_array[16];
   int16 NO_SPI_CHAR = 13;

   Uint16 checksum = 0;
   Uint16 checksum_test_tmp=0;
   //Checksum OK flag for the two SPI blocks
   Uint16 checksum_OK_1st = 0;
   Uint16 checksum_OK_2nd = 0;


   extern int16 Unit_number;
   extern second_watch ;
   extern second_watch_flag ;
   //Master status recieved from comm link, invaild if unit is master
   Uint16 rec_master_status_1st = 0;
   Uint16 rec_master_status_2nd = 0;
   //Recieved slave status recieved from comm link
   Uint16 rec_slave_status_1st = 0;
   Uint16 rec_slave_status_2nd = 0;
   //Recieved current compensation signal from other slave, useless
if unit is master
   Uint16 rec_curr_comp = 0;

   Uint16 period_load_2 = 0;
```

```
  Uint16 mas_fault = 0;
  Uint16 slav_fault = 0;

  Uint16 slave_delay = 0;                        //Flag to indicate
if unit's carrier is phase delayed from master unit by 90 degree

  //Compensator filter co-efficient
  int16
      beta_com_0, beta_com_1, beta_com_2, alpha_com_0, alpha_com_1,
alpha_com_2;

  int16
      rect_i_A = 0, com_filter_output;

  long
      rect_i_A_L = 0;

  int
      s0_1_com = 0, s0_0_com = 0,
      s1_1_com = 0, s1_0_com = 0,
      s2_1_com= 0, s2_0_com = 0;
  long
      branch1_A_com=0, branch2_A_com=0, branch3_A_com=0,
branch4_A_com=0, branch5_A_com=0;

  long
                I_res_DC_offset =0,
                I_res_cycle_total = 0,
                I_res_cycle_count = 0;


  //Variable for high pass PI
  double
      a0 = 0, a1 = 0, b1 = 0;

  long
      yn = 0, yn_1 = 0;

  int16
      xn = 0, xn_1 = 0;

  int32
      a0_fp = 0, a1_fp = 0, b1_fp = 0, Kp_fp = 0;

  int32
      HP_PI_prop = 0, HP_PI_integral = 0;

  int32
      one_on_INOM = 0;

  double
      yn_double = 0, yn_1_double = 0, xn_double = 0, xn_1_double =
0, PI_output_double = 0;

  double
      Kp_double = 0;

  Uint16
      prev_index_sin = 0;
```

```
  Uint16    PWMA1,
            PWMA2,
            PWMA3,
            PWMA4,
            PWMA5,
            PWMA6,
            PWMA7,
            PWMA8,
            PDPINT_Flag = 0 ,
            t = 0,
            hyst_pwm_status ;
  extern int16 real_sw_freq_serial ;
  extern int      real_bus_voltage_serial,
                  real_V3rd_serial,
                  real_V3rd_offset_serial,
                  average_offset_serial,
                  level_select_point_serial ;

  extern Uint16   real_fixed_band_serial ;

  extern int16     stepping ;

  extern double pf_adj_serial ;

  /*
=========================================================================
=====
  __Fast SPI Write Variables
  =========================================================================
========== */
  Uint16 spi_fail_count_dac1,spi_fail_count_dac2;
  int16 trash_spi_dac1,trash_spi_dac2;

  /*
=========================================================================
=====
  __Capture Port Variables
  =========================================================================
========== */
  long



            on_flag_A = 0 ,
            off_flag_A = 0 ,
            on_time_A   =     0     ,
            off_time_A  =     0     ,
            expected_off_time_A     =     0     ,
            on_time_A_prev    =     0     ,
            off_time_A_prev   =     0     ,
            tot_time_A  =     0     ,
            temp_stack_B = 0 ,
            PWM_STATE         = 0 ,
            PWM_OFF                     = 0 ,
            PWM_ON                      = 0 ,
            flag_R_synch      = 0 ,
            flag_F_synch      = 0 ,
            flag_R_cap        = 0 ,
            flag_F_cap        = 0 ,
            count_cap_select = 0 ,
```

```
            temp_stack_A                                    = 0 ,
            first_capture_A                                 = 0 ,
            first_capture_new_A                 = 0 ,
            first_capture_new_new_A       = 0 ,
            first_capture_old_A                 = 0 ,
            second_capture_A                          = 0 ,
            second_capture_new_A                = 0 ,
            zero_crossing_capture_A       = 0 ,
            zero_crossing_capture_on_A          = 0 ,
            zero_crossing_capture_off_A         = 0 ,
            capture_difference_A                = 0 ,
            band_offset_A                                       =
0 ,
            band_offset_A_OFF                                   =
0 ,
            period_a    =           0     ,
            interrupt_count                                 = 0 ,
            off_time_B                                      = 0 ,
            off_time_B_prev                                 = 0 ,
            on_time_B                                       = 0 ,
            period_B                                        = 0 ,
            period_B_prev                             = 0 ,
            off_time_expected_B                             = 0 ,
            timer_run                                       = 0 ,
            timer3_value                                    = 0 ,
            first_capture_B                                 = 0 ,
            first_capture_new_B                       = 0 ,
            first_capture_new_new_B                   = 0 ,
            first_capture_old_B                       = 0 ,
            second_capture_B                          = 0 ,
            second_capture_new_B                      = 0 ,
            zero_crossing_capture_on_B                = 0 ,
            zero_crossing_capture_off_B               = 0 ,
            capture_difference_B                      = 0 ,
            capture_difference_prev_B                 = 0 ,
            band_offset_B                                   = 0 ,
            band_offset_B_OFF                         = 0 ,

            first_capture_C                                 = 0 ,
            first_capture_new_C                       = 0 ,
            first_capture_new_new_C                   = 0 ,
            first_capture_old_C                       = 0 ,
            second_capture_C                          = 0 ,
            second_capture_new_C                      = 0 ,
            zero_crossing_capture_C                   = 0 ,
            capture_difference_C                      = 0 ,
            band_offset_C                                   = 0 ,
            band_offset_C_OFF                         = 0 ,
            level_select                                    = 0 ,
            level_select_B                                  = 0 ,
            level_select_temp_B                                 =
0 ,
            level_select_C                                  = 0 ,
            level_select_point                              = 0 ,
            pulse_count_B                                   = 0 ,

            V3rd_count                                      = 0 ,
            V3rd_offset                                     = 0 ,
            average_offset                                  = 0 ,
            fixed_hyst_band                                 = 0 ,
            fixed_hyst_band_transmitted               = 0 ,
```

```
            fixed_hyst_band_transmitted_up            = 0 ,
            fixed_hyst_band_transmitted_low           = 0 ,

            first_capture_synch                       = 0,
            first_capture_new_synch                   = 0,
            first_capture_new_new_synch         = 0,
            first_capture_old_synch                   = 0,
            second_capture_synch                      = 0,
            second_capture_new_synch                  = 0,
            zero_crossing_clock                       = 0,
            zero_crossing_clock_count                 = 0,
            zero_crossing_capture_synch_B             = 0,
            zero_crossing_capture_synch_C             = 0,
            interrupt_counter                         = 0,
            third_capture                                   = 0,
            counter_operation                               = 0,
            sw_frequency                                    = 0,
            period_hysteresis                               = 0,
            PWM_edge_count_A                          = 0,
            PWM_edge_count_B                          = 0,
            PWM_edge_count_C                          = 0,
            first_edge_count                          = 1,
            first_edge_count_B                              = 1,
            first_edge_count_C                              = 1,
            PWM_status_A                                    = 0,
            PWM_status_B                              = 0,
            PWM_status_C                              = 0,

            i                                                       =
0 ,
            one_forth_count   = 0 ,
            one_second_count  = 0 ,
            DAC_REF_FLAG                  = 0 ,
            peak_select                   = 0 ,
            tempp;

  int16     MDepth_max_main = 0 , MDepth_B = 0 , MDepth_A = 0 ,
Flag_max_finder = 1 , command_A_virt = 0 , duty_cycle_A_virt = 0 ,
duty_cycle_A = 0,duty_cycle_prev_A = 0 ,duty_cycle_A_3rd =
0,duty_cycle_B = 0,duty_cycle_prev_B = 0,duty_cycle_B_3rd =
0,duty_cycle_C = 0,duty_cycle_prev_C = 0,duty_cycle_C_3rd =
0,duty_cycle_A_3rd_prev = 0,duty_cycle_B_3rd_prev =
0,duty_cycle_C_3rd_prev = 0;
  long      Vband_A = 0, Vband_prev_A = 0 , Vband_B = 0,
Vband_prev_B = 0, Vband_B_test = 550, Vband_B_test_prev = 550,
Vband_C = 0 , Vband_prev_C = 0, duty_cycle_A_new = 0,
duty_cycle_A_new_prev = 0 ;
  signed int
            capture_top_stack = 0 ,
            capture_bot_stack = 0 ;

  int16 CAP5_read = 0 ,
  carrier = 0 ;

  int16 stepping_counter = 0 ;
  /*Comunication Parameters*/
  Uint16 wait = 0 ;

  //int16 spibuf[15];
  //int16 waste;
  //int16 checksum;
```

```
   /*
=========================================================================
=====
   __Local_Function_Prototypes()
   =======================================================================
========== */

   /* vsi state machine state functions */
   void
       st_vsi_init(void),            // initialises CFPP regulator
       st_vsi_stop(void),            // waiting for start trigger
       st_vsi_gate_charge(void), // delay to charge the high side
gate drivers
       st_vsi_ramp(void),            // ramping to target mod depth
       st_vsi_run(void),       // maintaining target mod depth
       st_vsi_fault(void);           // delay after faults are
cleared

   /// ADC and VSI interrupt
   interrupt void isr_adc(void);

   /// Gate fault (PDPINT) interrupt
   interrupt void isr_gate_fault(void);

   /// Calibrates the adc for gain and offset using the reference
inputs.
   void calibrate_adc(void);

   /// Scales the RMS summations to real volts and amps
   void scale_adc_rms(void);

   /// Scales the DC summations to real volts and amps
   void scale_adc_dc(void);

   //set and scale hystertesis band
   void SetHystBand(double);
   void SetBusVolts(double);

   // Timer 1 underflow interrupt
   //interrupt void isr_T1UF(void);

   // Timer 1 period interrupt
   //interrupt void isr_T1P(void);

   // Capture port interrupt
   interrupt void isr_CAP1(void);
   interrupt void isr_CAP2(void);
   interrupt void isr_CAP4(void);
   interrupt void isr_CAP5(void);
   interrupt void isr_CAP6(void);

   interrupt void isr_T2P(void);
   interrupt void isr_SPIRX(void);

   interrupt void isr_pwm(void);

   interrupt void isr_T1CINT(void);
   /*
=========================================================================
===== */
```

```
   /* State Machine Variable */
   /*
================================================================
===== */

   type_state
       vsi_state =
       {
             &st_vsi_init,
             1
       };


   /*
================================================================
=====
   __Exported_ADC_Functions()
   ================================================================
========== */

   /**

   This function initialises the ADC and VSI interrupt module. It
sets the
   internal ADC to sample the DA-2810 analog inputs and timer1 to
generate a PWM
   carrier and the event manager A to generate the VSI switching. It
also
   initialises all the relevant variables and sets up the interrupt
service
   routines.

   This functions initialises the ADC unit to:
   - Trigger a conversion sequence from timer 1 overflow
   - Convert the appropriate ADC channels

   Result registers as follows:
   - ADCRESULT0 = ADCINA0
   - ADCRESULT1 = ADCINB0
   - ADCRESULT2 = ADCINA1
   - ADCRESULT3 = ADCINB1
   - ADCRESULT4 = ADCINA2
   - ADCRESULT5 = ADCINB2
   - ADCRESULT6 = ADCINA3
   - ADCRESULT7 = ADCINB3
   - ADCRESULT8 = ADCINA4
   - ADCRESULT9 = ADCINB4
   - ADCRESULT10 = ADCINA5
   - ADCRESULT11 = ADCINB5
   - ADCRESULT12 = ADCINA6 yHA
   - ADCRESULT13 = ADCINB6     yHB
   - ADCRESULT14 = ADCINA7 yLA
   - ADCRESULT15 = ADCINB7 yLB

   It initialises the Evant Manager A unit to:
   - drive PWM1-4 as PWM pins not GPIO
   - a 0.48ns deadtime between the high and low side pins
   - Timer 1 as an up/down counter for the PWM carrier

   It initialises the PIE unit to:
   - Take PDPINTA as a power stage interrupt
```

```
  - Use the internal ADC completion interrupt to trigger the main
ISR

  \author A.McIver
  \par History:
  \li 12/10/07 AM - initial creation
  */
  void vsi_init(void)
  {
      EvaRegs.GPTCONA.all = 0x0000;
      EvaRegs.EVAIMRA.all = 0x0000;
      EvaRegs.EVAIFRA.all = BIT0;
      EvaRegs.COMCONA.all = 0x0000;
      EvaRegs.ACTRA.all = 0x0000;

  // Set up ISRs
      EALLOW;
      //PieVectTable.ADCINT = &isr_adc;
      PieVectTable.PDPINTA = &isr_gate_fault;
      PieVectTable.T1UFINT = &isr_pwm;
      PieVectTable.T1PINT =  &isr_pwm;
      PieVectTable.CAPINT1 = &isr_CAP1;
      PieVectTable.CAPINT2 = &isr_CAP2;
      PieVectTable.CAPINT4 = &isr_CAP4;
      PieVectTable.CAPINT5 = &isr_CAP5;
      PieVectTable.CAPINT6 = &isr_CAP6;
      PieVectTable.T1CINT = &isr_T1CINT;
            //PieVectTable.T2PINT = &isr_T2P;
      /* SPI interrupt test code */
      //PieVectTable.SPIRXINTA = &isr_SPIRX;
      EDIS;

  // Set up compare outputs
      EALLOW;
      GpioMuxRegs.GPDMUX.all = BIT0;
      GpioMuxRegs.GPDQUAL.bit.QUALPRD = 6; // 500ns qualification
period



      //Enable test pin on EVB
      GpioMuxRegs.GPBMUX.bit.T3PWM_GPIOB6 = 0;
      GpioMuxRegs.GPBMUX.bit.T4PWM_GPIOB7 = 0;

      GpioMuxRegs.GPBDIR.bit.GPIOB6 = 1;
      GpioMuxRegs.GPBDIR.bit.GPIOB7 = 1;

      //Enable first pin of EVB as digout pins for outputing sync
signal
      GpioMuxRegs.GPBMUX.bit.PWM7_GPIOB0 = 0;
      GpioMuxRegs.GPBDIR.bit.GPIOB0 = 1;

      //Enable second pin of EVB as digout pins for as a test point
      GpioMuxRegs.GPBMUX.bit.PWM8_GPIOB1 = 0;
      GpioMuxRegs.GPBDIR.bit.GPIOB1 = 1;

      //Enable second pin of EVB as digout pins for as a test point
      GpioMuxRegs.GPBMUX.bit.PWM9_GPIOB2 = 0;
      GpioMuxRegs.GPBDIR.bit.GPIOB2 = 1;
```

```
      //Allow SPISTE pin to be GPIO pin so it can be pull low to
enable SPI in slave mode
      GpioMuxRegs.GPFMUX.bit.SPISTEA_GPIOF3 = 0;
      GpioMuxRegs.GPFDIR.bit.GPIOF3 = 1;

      GpioMuxRegs.GPBMUX.bit.CAP4Q1_GPIOB8   = 1 ;    //Enabling
Capture port 4 pins
      GpioMuxRegs.GPBMUX.bit.CAP5Q2_GPIOB9   = 1 ;    //Enabling
Capture port 5 pins
      GpioMuxRegs.GPBMUX.bit.CAP6QI2_GPIOB10 = 1 ;    //Enabling
Capture port 6 pins

      EDIS;

      /*    DBT         DBTPS       time
            9           2                 0.48
            9           3                 0.96
            9           4                 1.92
            12          3                 1.28
      */

      //1.8us deadtime

      EvaRegs.DBTCONA.bit.DBT   = 8;
      EvaRegs.DBTCONA.bit.EDBT1 = 1;
      EvaRegs.DBTCONA.bit.EDBT2 = 1;
      EvaRegs.DBTCONA.bit.EDBT3 = 1;
      EvaRegs.DBTCONA.bit.DBTPS = 6;

      EvaRegs.CMPR1 = PERIOD_2/2.0;
      EvaRegs.CMPR2 = PERIOD_2;

      // Setup and load COMCONA
      EvaRegs.COMCONA.bit.ACTRLD = 2;           // reload ACTR
immediately
      EvaRegs.COMCONA.bit.SVENABLE = 0;   // disable space vector
PWM
      EvaRegs.COMCONA.bit.CLD = 1;        // reload on underflow or
period match
      EvaRegs.COMCONA.bit.FCOMPOE = 1;    // full compare enable
      EvaRegs.COMCONA.bit.CENABLE = 1;    // enable compare
operation

  // Set up Timer 1
      EvaRegs.T1CON.all = 0x0000;
      EvaRegs.T1PR = PERIOD_2;
      EvaRegs.T1CMPR = PERIOD_2/2;
      EvaRegs.T1CNT = 0x0000;

      EvaRegs.GPTCONA.bit.T1CMPOE = 1;
      // Setup and load GPTCONA
      //EvaRegs.GPTCONA.bit.T1TOADC = 2;  // period int flag starts
ADC
      EvaRegs.GPTCONA.bit.T1TOADC = 2;          // Timer 1 period
starts ADC
      //EvaRegs.GPTCONA.bit.T2TOADC = 2;        // Timer 2 period
starts ADC
      EvaRegs.GPTCONA.bit.TCMPOE  = 1;
      EvaRegs.GPTCONA.bit.T1PIN  = 1;

  // Set up ADC
```

```
      AdcRegs.ADCMAXCONV.all = 0x0007;              // 8 double
conv's (16 total)
      AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0x0;    // Setup ADCINA0/B0
as 1st SEQ1 conv.
      AdcRegs.ADCCHSELSEQ1.bit.CONV01 = 0x1;    // Setup ADCINA1/B1
as 2nd SEQ1 conv.
      AdcRegs.ADCCHSELSEQ1.bit.CONV02 = 0x2;    // Setup ADCINA2/B2
as 3rd SEQ1 conv.
      AdcRegs.ADCCHSELSEQ1.bit.CONV03 = 0x3;    // Setup ADCINA3/B3
as 4th SEQ1 conv.
      AdcRegs.ADCCHSELSEQ2.bit.CONV04 = 0x4;    // Setup ADCINA4/B4
as 5th SEQ2 conv.
      AdcRegs.ADCCHSELSEQ2.bit.CONV05 = 0x5;    // Setup ADCINA5/B5
as 6th SEQ2 conv.
      AdcRegs.ADCCHSELSEQ2.bit.CONV06 = 0x6;    // Setup ADCINA6/B6
as 7th SEQ2 conv.
      AdcRegs.ADCCHSELSEQ2.bit.CONV07 = 0x7;    // Setup ADCINA7/B7
as 8th SEQ2 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 = 0;       // disable
interrupt
      AdcRegs.ADCTRL2.bit.INT_MOD_SEQ1 = 1;       // int at end of
every SEQ1
      AdcRegs.ADCTRL2.bit.INT_MOD_SEQ2 = 1;
      AdcRegs.ADCTRL3.bit.SMODE_SEL = 1;          // simultaneous
sampling mode
      AdcRegs.ADCTRL3.bit.ADCCLKPS = 0x04;        // ADCLK =
HSPCLK/8 (9.375MHz)

      //GpioDataRegs.GPBCLEAR.all = SW_B ;
      //GpioDataRegs.GPBSET.all = SW_A ;

      /* ADC Registers:
  - ADCRESULT0    =     ADCINA0 - NC-A:I3 or NO-A:APOT1
  - ADCRESULT1    =     ADCINB0 - Vdc2
  - ADCRESULT2    =     ADCINA1 - NC-A:Vac3 or NO-A:Vdc3
  - ADCRESULT3    =     ADCINB1 - I5
  - ADCRESULT4    =     ADCINA2 - I1
  - ADCRESULT5    =     ADCINB2 - I4
  - ADCRESULT6    =     ADCINA3 - Vac1
  - ADCRESULT7    =     ADCINB3 - Vdc1
  - ADCRESULT8    =     ADCINA4 - I2
  - ADCRESULT9    =     ADCINB4 - NO-B:APOT2 or NC-B:I6
  - ADCRESULT10   =     ADCINA5 - Vac2
  - ADCRESULT11   =     ADCINB5 - NO-B:Vgen or NC-B:Vdc4
  - ADCRESULT12   =     ADCINA6 - yHA
  - ADCRESULT13   =     ADCINB6 - yHB
  - ADCRESULT14   =     ADCINA7 - yLA
  - ADCRESULT15   =     ADCINB7 - yLB
  */

      /* ADC Registers& IIB Mapping
  - ADCRESULT0    =     ADCINA0 - ADC0-VDC1
  - ADCRESULT1    =     ADCINB0 - ADC1-APOT1
  - ADCRESULT2    =     ADCINA1 - ADC2-I1
```

```
  - ADCRESULT3    =     ADCINB1 - ADC3-VAC1
  - ADCRESULT4    =     ADCINA2 - ADC4-I3
  - ADCRESULT5    =     ADCINB2 - ADC5-I4
  - ADCRESULT6    =     ADCINA3 - ADC6-VAC4
  - ADCRESULT7    =     ADCINB3 - ADC7-I6
  - ADCRESULT8    =     ADCINA4 - ADC8-VDC2
  - ADCRESULT9    =     ADCINB4 - ADC9-APOT2
  - ADCRESULT10   =     ADCINA5 - ADC10-I2
  - ADCRESULT11   =     ADCINB5 - ADC11-VAC2
  - ADCRESULT12   =     ADCINA6 - ADC12-VAC3
  - ADCRESULT13   =     ADCINB6 - ADC13-I5
  - ADCRESULT14   =     ADCINA7 - ADC14-VAC5
  - ADCRESULT15   =     ADCINB7 - ADC15-VAC6
  */

  // Enable interrupts
      DINT;
      EvaRegs.EVAIMRA.all = 0;                          // disable all
interrupts
      // Enable PDPINTA: clear PDPINT flag and T1PINT flag
      //EvaRegs.EVAIFRA.all = BIT0|BIT7;
      //EvaRegs.EVAIMRA.bit.PDPINTA = 1;              //Disable for
testing WYK 2009/05/20
      EvaRegs.EVAIFRA.all = BIT0|BIT7|BIT9|BIT8;
      //PDPINTA, T1UFINT, T1PINT, T1CINTenabled
      EvaRegs.EVAIMRA.bit.T1UFINT = 1;
      EvaRegs.EVAIMRA.bit.T1PINT = 1;
      EvaRegs.EVAIMRA.bit.T1CINT = 1;
      EvaRegs.EVAIMRB.bit.T2PINT = 1;




      PieCtrlRegs.PIEIER1.bit.INTx1 = 1;        // Enable PDPINTA in
PIE: Group 1 interrupt 1
      //PieCtrlRegs.PIEIER1.bit.INTx6 = 1;      // Enable ADC
interrupt in PIE: Group 1 interrupt 6
      PieCtrlRegs.PIEIER2.bit.INTx6 = 1;        // Enable T1UFINT in
PIE: Group 2 interrupt 6.  WYK 20090525
      PieCtrlRegs.PIEIER2.bit.INTx4 = 1;        // Enable T1PINT in
PIE: Group 2 interrupt 4.  WYK 20091013
      PieCtrlRegs.PIEIER2.bit.INTx5 = 1;        // Enable T1CINT in
PIE: Group 2 interrupt 5.  WYK 20091207
      PieCtrlRegs.PIEIER3.bit.INTx5 = 1;        // Enable CAPINT1 in
PIE: Group 3 interrupt 5
      PieCtrlRegs.PIEIER3.bit.INTx6 = 1;        // Enable CAPINT2 in
PIE: Group 3 interrupt 6
      PieCtrlRegs.PIEIER5.bit.INTx5 = 1;        // Enable CAPINT4 in
PIE: Group 5 interrupt 5
      PieCtrlRegs.PIEIER5.bit.INTx6 = 1;        // Enable CAPINT5 in
PIE: Group 5 interrupt 6
      PieCtrlRegs.PIEIER5.bit.INTx7 = 1;        // Enable CAPINT6 in
PIE: Group 5 interrupt 7
      //PieCtrlRegs.PIEIER3.bit.INTx1 = 1;      // Enable T2PINT in
PIE: Group 3 interrupt 1
      //PieCtrlRegs.PIEIER6.bit.INTx1 = 1;;   //SPI interrupt test

      IER |= M_INT1|M_INT2|M_INT3|M_INT5;     // Enable CPU
Interrupts 1,2,3,5,6
```

```
      //IER |= M_INT1|M_INT2|M_INT3|M_INT6;   // Enable CPU
Interrupts 1,2,3,6
      EINT;

      AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1; // clear interrupt flag
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;   // Acknowledge
interrupt to PIE
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP2;   // Acknowledge
interrupt to PIE
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;   // Acknowledge
interrupt to PIE
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP5;   // Acknowledge
interrupt to PIE

      /*Setup timer 2 for capture port operation */
      //Timer period -1 = switching period (to match the timer
period as period of
      //a up-down is 2* period, period of continuous up is period -
1).  This allow for easier decoding of capture results
      EvaRegs.T2PR = 65536 - 1;
      EvaRegs.T2CNT = 0;
      EvaRegs.T2CON.all = 0x0000;
      EvaRegs.T2CON.bit.TCLKS10 = 0x00;
      EvaRegs.T2CON.bit.SET1PR = 0x00;
      EvaRegs.T2CON.bit.TMODE = 2;                    //Continuous
up count mode
      EvaRegs.T2CON.bit.TPS = 7;                //prescalar 1/128
      EvaRegs.T2CON.bit.T2SWT1 =  1;            //Use own TENABLE
bit
      EvaRegs.T2CON.bit.TENABLE = 1;            // enable timer2

      /*Capture port setting */
       GpioMuxRegs.GPAMUX.bit.CAP1Q1_GPIOA8 = 1;
       GpioMuxRegs.GPAMUX.bit.CAP2Q2_GPIOA9 = 1;
      EvaRegs.CAPCONA.all = 0x0000;
      EvaRegs.CAPCONA.bit.CAPRES = 0;                       //Reset
capture unit
      EvaRegs.CAPCONA.bit.CAP12EN = 1;                //Enable
capture port 1 and 2
      EvaRegs.CAPCONA.bit.CAP1EDGE = 0x03;            //Detect both
edge in capture port 1
      EvaRegs.CAPCONA.bit.CAP2EDGE = 0x03;            //Detect both
edge in capture port 2
      //Make the capture unit believe it already has 1 result.  This
causes the capture port to generate an interrupt when
      //it set another capture come along.  (Capture port only
generate interrupt at second capture)
      EvaRegs.CAPFIFOA.bit.CAP1FIFO = 1;
      EvaRegs.CAPFIFOA.bit.CAP2FIFO = 1;

      /*Setting up capture port 1 interrupt */
      EvaRegs.EVAIMRC.all = 0;                        //Disable all
capture port interrupt
      EvaRegs.EVAIFRC.all = 0;                        //Clearing
interrupt flag for capture port
      EvaRegs.EVAIMRC.bit.CAP1INT = 1;          //Enabling capture
port 1 interrupt
      EvaRegs.EVAIMRC.bit.CAP2INT = 0;          //Enabling capture
port 2 interrupt

      /*Setup timer 3 for capture port 4 operation on EVB*/
```

```
      //Timer period -1 = switching period (to match the timer
period as period of
      //a up-down is 2* period, period of continuous up is period -
1).  This allow for easier decoding of capture results
      EvbRegs.T3PR = (PERIOD_2 * 2) - 1;
      EvbRegs.T3CNT = 0;
      EvbRegs.T3CON.all = 0x0000;
      EvbRegs.T3CON.bit.TCLKS10 = 0x00;
      EvbRegs.T3CON.bit.TMODE = 2;        //Continuous up count mode
      EvbRegs.T3CON.bit.TPS = 0;              //no prescalar
      EvbRegs.T3CON.bit.TENABLE = 1;         // enable timer3

            /*Setup timer 4 for capture port 4 operation on EVB*/
      //Timer period -1 = switching period (to match the timer
period as period of
      //a up-down is 2* period, period of continuous up is period -
1).  This allow for easier decoding of capture results
      EvbRegs.T4PR = (PERIOD_2 * 2) - 1;
      EvbRegs.T4CNT = 0;
      EvbRegs.T4CON.all = 0x0000;
      EvbRegs.T4CON.bit.TMODE   = 2;            //Continuous up
count mode
      EvbRegs.T4CON.bit.TPS     = 0;          //no prescalar
      EvbRegs.T4CON.bit.T4SWT3  = 0;         // Star timer 4
independently
      EvbRegs.T4CON.bit.TENABLE = 0;         // enable timer4

      //Capture port B Setting
      EvbRegs.CAPCONB.all = 0x0000;
      EvbRegs.CAPCONB.bit.CAP45EN   = 1;                    //
Enable captures 4 and 5
      EvbRegs.CAPCONB.bit.CAP6EN    = 0;                    //
Enable capture 6
      EvbRegs.CAPCONB.bit.CAPRES    = 0;                    //
Release from reset
      EvbRegs.CAPCONB.bit.CAP4EDGE  = 0x03;                 //
fallin edge on capture 4
      EvbRegs.CAPCONB.bit.CAP5EDGE  = 0x03;                 // both
edge on capture 5
      EvbRegs.CAPCONB.bit.CAP6EDGE  = 0x02;                 //
falling edge on capture 6
      EvbRegs.CAPCONB.bit.CAP45TSEL = 0;                    //
Capture 4 and 5 select timer 3
      EvbRegs.CAPCONB.bit.CAP6TSEL  = 1;                    //
Capture 6 select timer 3

     /*Setting up capture port 4,5,6 interrupt */
      EvbRegs.EVBIMRC.all = 0;                        // Disable all
capture port interrupt
      EvbRegs.EVBIFRC.all = 0;                        // Clearing
interrupt flag for capture port
      EvbRegs.EVBIMRC.bit.CAP4INT = 0;          // Enabling capture
port 4 interrupt
      EvbRegs.EVBIMRC.bit.CAP5INT = 0;          // Enabling capture
port 5 interrupt
      EvbRegs.EVBIMRC.bit.CAP6INT = 0;          // Enabling capture
port 6 interrupt

      //Make the capture unit believe it already has 1 result.  This
causes the capture port to generate an interrupt when
```

```
      //it set another capture come along.  (Capture port only
generate interrupt at second capture)
      //EvbRegs.CAPFIFOB.bit.CAP4FIFO = 1;
      //EvbRegs.CAPFIFOB.bit.CAP5FIFO = 1;
      //EvbRegs.CAPFIFOB.bit.CAP6FIFO = 1;


   /* Setup and load T1CON to start operation */
      EvaRegs.T1CON.bit.TMODE = 1;        // continous up/down count
mode
      EvaRegs.T1CON.bit.TPS = 0;                // input clock
prescaler
      EvaRegs.T1CON.bit.TECMPR = 1;       // enable time compare
      EvaRegs.T1CON.bit.TCLD10 = 1;             // timer compare
reload at underflow and period
      EvaRegs.T1CON.bit.TENABLE = 0;            // enable timer

      EvaRegs.T1CNT = 0 ;
      EvbRegs.T4CNT = 0 ;

      EvaRegs.T1CON.bit.TENABLE = 1;            // enable timer
timer 1
      EvbRegs.T4CON.bit.TENABLE = 1;            // enable timer
timer 4


      // Initialise state machine
      vsi_state.first = 1;
      vsi_state.f = &st_vsi_init;
                        GrabShow();
                        GrabClear();
                        GrabStart();
                        GrabRun();

   } /* end vsi_init */


   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
   /**
   This function is called from the main background loop once every
millisecond.
   It performs all low speed tasks associated with running the core
interrupt
   process, including:
   - checking for faults
   - calling the VSI state functions
   - calling internal analog scaling functions

   \author A.McIver
   \par History:
   \li 13/10/07 AM - derived from 25kVA:vsi:vsi.c
   */
   void vsi_state_machine(void)
   {
      SS_DO(vsi_state);
      if (adc_int.flag_rms != 0) // rms flag synched to VSI
fundamental
      {
            adc_int.flag_rms = 0;
            scale_adc_rms();
```

```
      }
      else if (adc_int.flag_dc != 0) // ADC_DC_TIME flag
      {
            adc_int.flag_dc = 0;
            scale_adc_dc();
      }
      else if (adc_int.flag_cal != 0)
      {
            adc_int.flag_cal = 0;
            calibrate_adc();
      }
  } /* end vsi_state_machine */


  /*
==========================================================================
=====
  __Exported_VSI_Functions()
  ==========================================================================
========== */

  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
  /**
  This function switches the VSI from the stopped state to a running
state.

  \author A.McIver
  \par History:
  \li 13/10/07 AM - derived from 25kVA:vsi:vsi.c
  */
  void vsi_enable(void)
  {
      if (detected_faults == 0)
            is_switching = 1;
  } /* end vsi_enable */


  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
  /**
  This function switches the VSI from the running state to a stop
state.

  The ramp down process has the side effect of resetting the
reference to zero.

  \author A.McIver
  \par History:
  \li 13/10/07 AM - derived from 25kVA:vsi:vsi.c
  */
  void vsi_disable(void)
  {
      is_switching = 0;
  } /* end vsi_disable */


  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
  /**
  This function sets the target output modulation depth.
```

```
  The target is passed in tenths of a percent, so a value of 1000
corresponds to
  100% modulation depth.

  \author A.McIver
  \par History:
  \li 24/04/09 AM - initial creation

  \param[in] m Target output modulation depth
  */
  void vsi_set_mod(Uint16 m)
  {
      int32
            temp;

      if (m > MOD_DEPTH_MAX)
      {
            m = MOD_DEPTH_MAX;
      }
      temp = (((int32)m) << MOD_SHIFT) / ((int32)MOD_DEPTH_MAX);

      mod_ref = (int16)temp;
  } /* end vsi_set_mod */


  void vsi_set_mod_immediate(Uint16 m)
  {
      int32
            temp;

      if (m > MOD_DEPTH_MAX)
      {
            m = MOD_DEPTH_MAX;
      }
      temp = (((int32)m) << MOD_SHIFT) / ((int32)MOD_DEPTH_MAX);

      mod_ref = (int16)temp;
      mod_targ = mod_ref;
      //Shift left of 1 introduce to deal with the fact that
mod_targ 200% at full range
      I_ref_Peak_AB = ((long)I_NOM*((long)mod_targ<<1))>>MOD_SHIFT;

  } /* end vsi_set_mod */


  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
  /**
  This function returns the target output modulation depth.

  \author A.McIver
  \par History:
  \li 24/04/09 AM - initial creation

  \returns The VSI target output modulation depth in tenths of a
percent
  */
  Uint16 vsi_get_mod(void)
  {
      int32
```

```
            temp;

      temp = ((int32)mod_ref * (int32)MOD_DEPTH_MAX +
(1L<<(MOD_SHIFT-1)) )
                        >> MOD_SHIFT;

      return (Uint16)temp;
  } /* end vsi_get_mod */


  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
  /**
  Set the target output frequency in Hz.

  \author A.McIver
  \par History:
  \li 12/10/07 AM - initial creation
  \li 04/03/08 AM - added return of new frequency

  \returns The new frequency in Hz

  \param[in] f Target fundamental frequency in Hz
  */
  double vsi_set_freq(double f)
  {
      if (f == 0.0||refMode == DC_REF) // DC output
      {
            phase_step = 0L;
            phase = 32768uL*65536uL;
      }
      else
      {
            if (f > F_FREQ_MAX)
            {
                  f = F_FREQ_MAX;
            }
            else if (f < F_FREQ_MIN)
            {
                  f = F_FREQ_MIN;
            }

            phase_step = (Uint32)(PHASE_STEP_SC * f + 0.5); //
atomic load
      }
      Ref_freq_float = f;
      return (double)phase_step/PHASE_STEP_SC;
  } /* end vsi_set_freq */


  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
  /**
  This function returns the VSI fundamental frequency.

  \author A.McIver
  \par History:
  \li 19/06/08 AM - initial creation

  \returns The VSI fundamental frequency
  */
```

```
  double vsi_get_freq(void)
  {
      return (double)phase_step/PHASE_STEP_SC;
  } /* end vsi_get_freq */


  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
  /**
  This function returns the status of the VSI output system. It
returns
  - stopped or running
  - fault code
  - ramping or settled

  \author A.McIver
  \par History:
  \li 13/10/07 AM - derived from 25kVA:vsi:vsi.c

  \retval VSI_RUNNING VSI system switching with output
  \retval VSI_SETTLED Output has reached target
  \retval VSI_FAULT VSI system has detected a fault
  */
  Uint16 vsi_get_status(void)
  {
      return vsi_status;
  } /* end vsi_get_status */


  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
  /**
  This function returns the fault word of the VSI module.

  \author A.McIver
  \par History:
  \li 04/03/08 AM - initial creation

  \returns The present fault word
  */
  /// Report what faults are present in the VSI
  Uint16 vsi_get_faults(void)
  {
      return detected_faults;
  } /* end vsi_get_faults */


  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
  /* void vsi_clear_faults(void)
  Parameters: none
  Returns: nothing
  Description: Clear the detected faults.
  Notes:
  History:
      13/10/05 AM - initial creation
  \li 28/04/08 AM - added event reporting
  */
  void vsi_clear_faults(void)
  {
      Uint16
```

```
            i;

      if (detected_faults & FAULT_VSI_PDPINT)
      {
            for (i=0; i<100; i++)
                  i++; // delay for fault to clear

            EvaRegs.COMCONA.all = 0;
            EvaRegs.COMCONA.all = 0xAA00;
      }
      detected_faults = 0;
  } /* end vsi_clear_faults */


  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
  /* Uint16 vsi_get_vdc(void)
  Parameters: none
  Returns: DC bus voltage in Volts
  Description: Retrieves filtered and scaled Vh measurements.
  Notes:
  History:
      13/10/05 AM - initial creation
  */
  /*
  Function is commented out until scaling is set up
   Uint16 vsi_get_vdc(void)
  {
      return (Uint16)(adc_int.vdc.real + 0.5);
  }*/ /* end vsi_get_vdc */


  /*
==========================================================================
===== */
  /* Interrupt Routines */
  /*
==========================================================================
===== */

  #ifndef BUILD_RAM
  #pragma CODE_SECTION(isr_adc, "ramfuncs");
  #endif
  interrupt void isr_adc(void)
  {

      EvaRegs.EVAIFRA.all = BIT7;                     // clear
interrupt flag
      AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1;       // clear interrupt
flag
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;   // Acknowledge
interrupt to PIE
      adc_ready_flag = 1;


  }
  #ifndef BUILD_RAM
  #pragma CODE_SECTION(isr_T1CINT, "ramfuncs");
  #endif
  interrupt void isr_T1CINT(void)
  {
```

```
  if (master_slave_mode == 1)

  {
      SIN_TABLE_UPDATE(phase_sin, index_sin);
      SIN_TABLE_UPDATE(phase_sin_4PI_on_3, index_sin_4PI_on_3);
      SIN_TABLE_UPDATE(phase_sin_2PI_on_3, index_sin_2PI_on_3);

      if(stepping)
      {
            if(stepping_counter < 1500)
            {
                  I_ref_Peak_AB_tr = I_ref_Peak_AB * 2 ;
            //    SET_TP10();
            }
            else if (stepping_counter < 3000)
            {
                  //CLEAR_TP10();
                  I_ref_Peak_AB_tr = I_ref_Peak_AB ;

            }
      else if(stepping_counter == 3000)
                  stepping_counter = 0;

            stepping_counter++;

      I_ref_A = ((((long)I_ref_Peak_AB_tr<<1)*(long)sin_val) >> 16);
      I_ref_B =
((((long)I_ref_Peak_AB_tr<<1)*(long)sin_4PI_on_3_val) >> 16);
      I_ref_C =
((((long)I_ref_Peak_AB_tr<<1)*(long)sin_2PI_on_3_val) >> 16);
      }
      else
      {
      I_ref_A = ((((long)I_ref_Peak_AB<<1)*(long)sin_val) >> 16);
      I_ref_B = ((((long)I_ref_Peak_AB<<1)*(long)sin_4PI_on_3_val)
>> 16);
      I_ref_C = ((((long)I_ref_Peak_AB<<1)*(long)sin_2PI_on_3_val)
>> 16);
      }
      SIN_TABLE_READ(index_sin, sin_val);
      SIN_TABLE_READ(index_sin_4PI_on_3, sin_4PI_on_3_val);
      SIN_TABLE_READ(index_sin_2PI_on_3, sin_2PI_on_3_val);

  /*  if(I_ref_B > 0)
            SET_TP10();
      else
            CLEAR_TP10();
  */


      fixed_hyst_band   = real_fixed_band_serial      ;
      bus_voltage                   = real_bus_voltage_serial     ;

      GpioDataRegs.GPDSET.all = nDAC1;
      GpioDataRegs.GPDSET.all = nDAC2;
      SET_TP11();
      SPI_Primary_dis_Secondary_en();   // s = 1
      SPI_Secondary_Slave_DSP_Master(); // oe = 0
      //MASTER_SEND_DATA(((i>>8) & 0x00FF));//
      //MASTER_SEND_DATA(i & 0x00FF);//
```

```
      for(i=0;i<20;i++);
      spi_putc(((I_ref_A>>8) & 0x00FF));
      spi_putc(I_ref_A & 0x00FF);

      spi_putc(((I_ref_C>>8) & 0x00FF));
      spi_putc(I_ref_C & 0x00FF);

      spi_putc(((fixed_hyst_band>>8) & 0x00FF));
      spi_putc(fixed_hyst_band & 0x00FF);

      spi_putc(((bus_voltage>>8) & 0x00FF));
      spi_putc(bus_voltage & 0x00FF);

      SPI_Primary_en_Secondary_dis();   // s = 0
      SPI_Secondary_Master_DSP_Slave() ;//oe = 1
      CLEAR_TP11();
  }
  else if (master_slave_mode == 0)
      {


      SPI_Primary_dis_Secondary_en();   // s =1
      SPI_Secondary_Master_DSP_Slave(); // oe = 1
      DSP_SET_SPI_SLAVE() ;

      }

      EvaRegs.EVAIFRA.all = BIT8;
      //Clear interrupt flag
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP2;   // Acknowledge
interrupt to PIE

  }

  /**
  \fn interrupt void isr_adc(void)
  \brief Updates VSI and stores ADC results

  This interrupt is triggered by the completion of the internal ADC
conversions.
  It then:
  - stores the internal ADC results
  - applies the internal 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 on internal
ADC quantities
  - updates phase angle
  - calculates switching times
  - loads compare registers with switching times
  - sets up analogs for next interrupt

  \author A.McIver
  \par History:
  \li 12/10/07 AM - initial creation
  */
  #ifndef BUILD_RAM
  #pragma CODE_SECTION(isr_pwm, "ramfuncs");
  #endif
  interrupt void isr_pwm(void)
```

```
  {

      //Find out the direction which the timer is going, used to
update timer 1 compare to

      int timer1_dir = EvaRegs.GPTCONA.bit.T1STAT;
      int timer4_dir = EvbRegs.GPTCONB.bit.T4STAT;


      interrupt_counter++;
      if(interrupt_counter ==2)
      {
            zero_crossing_clock_count++ ;
            interrupt_counter = 0 ;
      SET_TP10() ;
      //if(zero_crossing_clock_count == 2) zero_crossing_clock_count
= 0 ;
      if(zero_crossing_clock_count == 2)
      {
      zero_crossing_clock_count = 0 ;


      zero_crossing_clock = EvaRegs.T2CNT;

  /*  if((level_select==1) || (level_select_B==1) ||
(level_select_C==1))
      {
            zero_crossing_capture_A = zero_crossing_capture_off_A ;
      }
      else if((level_select==0) || (level_select_B==0) ||
(level_select_C==0))
      {
            zero_crossing_capture_A = zero_crossing_capture_on_A ;
      }*/

      if (master_slave_mode == 1)
      {
      if((zero_crossing_clock > 60000L) && (zero_crossing_capture_A
< 2000L))
      {
            zero_crossing_capture_A       = zero_crossing_capture_A
      + 65535L ;
      }
      else if((zero_crossing_capture_A > 60000L) &&
(zero_crossing_clock < 2000L) )
      {
            zero_crossing_clock = zero_crossing_clock + 65535L ;
      }

      capture_difference_A = zero_crossing_clock -
zero_crossing_capture_A - 234L;
      }
      else if (master_slave_mode == 0)
      {
      if((zero_crossing_clock > 60000L) && (zero_crossing_capture_A
< 2000L))
      {
            zero_crossing_capture_A       = zero_crossing_capture_A
      + 65535L ;
      }
```

```
      else if((zero_crossing_capture_A > 60000L) &&
(zero_crossing_clock < 2000L) )
      {
            zero_crossing_clock = zero_crossing_clock + 65535L ;
      }

      capture_difference_A = zero_crossing_clock -
zero_crossing_capture_A - 234L;
      }
      }
  }
            //duty_cycle_A_new = duty_cycle_A ;
            duty_cycle_A_new = ((512L * duty_cycle_A) - (256L *
duty_cycle_prev_A)) >> 8 ;

      if(duty_cycle_A_new < 20) //< average_offset)
      {
            duty_cycle_A_new = 20; //average_offset ;
            capture_difference_A = 0 ;
      }
      if(duty_cycle_A_new > 95L)
            duty_cycle_A_new = 95L ;
      if(duty_cycle_A_new < -95L)
            duty_cycle_A_new = -95L ;
  if (master_slave_mode == 1)

  {
      if (timer1_dir == 1)//When the the timer value reaches
underflow
      {
            // Send the sync pulse through B4
            SET_TP13();
            //GpioDataRegs.GPBDAT.bit.GPIOB4 = 1;
            GpioDataRegs.GPBDAT.bit.GPIOB5 = 1;
            for (i=0;i<3;i++)
                  wait++;
            CLEAR_TP13();
            //GpioDataRegs.GPBDAT.bit.GPIOB4 = 0;
            GpioDataRegs.GPBDAT.bit.GPIOB5 = 0;
      }

      //dac_fast_write(DAC_MODULE_D2,DAC_WRn_UPDn,DAC_ADDR_A,
fixed_hyst_band );
      Vband_B = (((duty_cycle_A_new) * (100L - duty_cycle_A_new )) *
fixed_hyst_band * 1) / 2500L ;
      //Vband_B = (((duty_cycle_A) * (100L - duty_cycle_A )) *
fixed_hyst_band * 1) / 2500L ;

  //  if(duty_cycle_A_new <= 20) capture_difference_A = 0 ;
      band_offset_A = ((((long)(capture_difference_A * 1118L) >>7) *
Vband_B )>>12);

      if (band_offset_A > 200L)
            band_offset_A = 200L ;
      else if (band_offset_A < -200L)
            band_offset_A = -200L ;

      dac_fast_write(DAC_MODULE_D2,DAC_WRn_UPDn,DAC_ADDR_A, Vband_B
+ band_offset_A );
      DAC1_FAST_WRITE(DAC_WRn_UPDn,DAC_ADDR_D,(I_ref_B - 5) +2047);
      //DAC1_FAST_WRITE(DAC_WRn_UPDn,DAC_ADDR_D,(I_ref_B) +2047);
```

```
      //DAC1_FAST_WRITE(DAC_WRn_UPDn,DAC_ADDR_A,(Vnp * V3rd_offset)
+2047);
      CLEAR_TP10();

  }
            msecond_count_on ++ ;
      if(start_flag_off == 1)
            msecond_count_off ++ ;

  predicted_on_time  = EvaRegs.T2CNT - first_capture_new_A ;
  predicted_off_time = EvaRegs.T2CNT - second_capture_A    ;

  if(predicted_on_time < -50000L)
      predicted_on_time  = predicted_on_time + 65535L ;

  if(predicted_off_time < -50000L)
      predicted_off_time  = predicted_off_time + 65535L ;

  if(stepping)
  {

      if(abs(predicted_on_time) > (period_HCC_prev + hyst_band_B ))
      {
            start_flag = 1 ;
            level_count_prev = level_count ;
            if(msecond_count_on > 70)
            {
                  msecond_count_on = 0 ;
                  start_flag_on = 0 ;
                  start_flag_off = 1 ;
                  level_count = 1 ;
                  detect_flag = 1 ;
            }

      }

      if(abs(predicted_off_time) > (period_HCC_prev + hyst_band_B))
      {
            start_flag = 1 ;
            level_count_prev = level_count ;
            if(msecond_count_off > 70)
            {
                  msecond_count_off = 0 ;
                  start_flag_on = 1 ;
                  start_flag_off = 0 ;
                  level_count = 2 ;
                  detect_flag = 1 ;
            }

      }

  if(msecond_count_on > 110)
  {
      start_flag_off = 1 ;
      start_flag_on  = 0 ;
      msecond_count_on = 0 ;
      detect_flag = 1 ;
  }
  if(msecond_count_off > 110)
  {
      start_flag_off = 0 ;
```

```
      start_flag_on  = 1 ;
      msecond_count_off = 0 ;
      detect_flag = 1 ;
  }

      if ((level_count == 1) && (detect_flag == 1))
      {
            //SET_TP11();
            detect_flag = 0 ;
            GpioDataRegs.GPASET.all       = BIT1;
      }

      else if ((level_count == 2) && (detect_flag == 1))
      {
            //CLEAR_TP11();
            detect_flag = 0 ;
            GpioDataRegs.GPACLEAR.all     = BIT1;
      }


  }
  if (master_slave_mode == 0)
      {
            //    SPI_Secondary_Master_DSP_Slave(); // oe = 1


      I_ref_A_1_up  = ((SpiaRegs.SPIRXBUF) & 0x00FF);
      I_ref_A_1_low = (SpiaRegs.SPIRXBUF & 0x00FF);
      I_ref_A_1     = (int)(I_ref_A_1_low | (I_ref_A_1_up << 8)) ;


      I_ref_C_1_up  = ((SpiaRegs.SPIRXBUF) & 0x00FF);
      I_ref_C_1_low = (SpiaRegs.SPIRXBUF & 0x00FF);
      I_ref_C_1     = (int)(I_ref_C_1_low | (I_ref_C_1_up << 8)) ;

      fixed_hyst_band_transmitted_up   = ((SpiaRegs.SPIRXBUF) &
0x00FF);
      fixed_hyst_band_transmitted_low  = ((SpiaRegs.SPIRXBUF) &
0x00FF);
      fixed_hyst_band_transmitted =
(fixed_hyst_band_transmitted_low) | (fixed_hyst_band_transmitted_up
<< 8) ;

      bus_voltage_up   = ((SpiaRegs.SPIRXBUF) & 0x00FF);
      bus_voltage_low  = ((SpiaRegs.SPIRXBUF) & 0x00FF);
      bus_voltage_trans = (bus_voltage_low) | (bus_voltage_up << 8)
;


      SPI_Primary_en_Secondary_dis();   // s = 0
      DSP_SET_SPI_MASTER() ;

            //SET_TP11();
            if (timer1_dir == 1)
            {
                  // Send the sync pulse through T1PWM
                  SET_TP13(); // GpioDataRegs.GPBDAT.bit.GPIOB0 = 1;
                  for (i=0;i<5;i++)
                        wait++;
                  CLEAR_TP13(); //GpioDataRegs.GPBDAT.bit.GPIOB0 =
0;
```

```
            }
            // Slave tries to sync to the value in the capture port
            if (timer1_dir == 0)
            {
                  /*
                  CAP1_read = EvaRegs.CAP1FIFO;
                  Line above commented out because DIGIN5_5 is not
going through the header correct
                  Could be a bus contention somewhere
                  */
                  CAP5_read = EvbRegs.CAP5FIFO;
                  if (CAP5_read > period_2)
                        carrier = CAP5_read - 2*period_2;
                  else
                        carrier= CAP5_read;

                  if(carrier < 60 )
                  {
                        // We are lagging the master
                        // Reduce the period to catch up
                        carrier_adjust = -1;
                  }
                  else if (carrier > 65 )
                  {
                        // We are leading the master
                        // Increase the period to catch up
                        carrier_adjust = 1;
                  }
                  else
                        carrier_adjust = 0;

                  // We want it to wobble around the original FSW
                  period_2 = PERIOD_2 + carrier_adjust;
                  period = period_2*2;
                  EvaRegs.T1PR = period_2;
                  EvbRegs.T4PR = period_2*2-1;
            }

      Vband_B = (((duty_cycle_A_new) * (100L - duty_cycle_A_new )) *
fixed_hyst_band_transmitted * 1) / 2500L ;
   //  Vband_B = (((duty_cycle_A) * (100L - duty_cycle_A )) *
fixed_hyst_band_transmitted * 1) / 2500L ;

      if(duty_cycle_A_new <= 20) capture_difference_A = 0 ;
      band_offset_A = ((((long)(capture_difference_A *
/*1118L*/bus_voltage_trans) >>7) * Vband_B)>>12);

      if (band_offset_A > 200L)
            band_offset_A = 200L ;
      else if (band_offset_A < -200L)
            band_offset_A = -200L ;

      dac_fast_write(DAC_MODULE_D2,DAC_WRn_UPDn,DAC_ADDR_A,
Vband_B);//+ band_offset_A );


      if(Unit_number == 1)
      {


   ////////////////////////////////////////////////////////////////
```

```
      if(level_select)

            DAC1_FAST_WRITE(DAC_WRn_UPDn,DAC_ADDR_D,(I_ref_A_1 - 20)
+2047);
            //DAC1_FAST_WRITE(DAC_WRn_UPDn,DAC_ADDR_D,(I_ref_A_1)
+2047);

      }
      else if(Unit_number == 2)
      {
            DAC1_FAST_WRITE(DAC_WRn_UPDn,DAC_ADDR_D, (I_ref_C_1 -
10) +2047);
            //DAC1_FAST_WRITE(DAC_WRn_UPDn,DAC_ADDR_D, (I_ref_C_1)
+2047);
      }

      }

  /*
=========================================================================
=====
  isr_pwm_1st_SPI()
  ========================================================================
========== */



      Vdc = (AdcRegs.ADCRESULT7>>4);
      Vdc = ((int16)((((int32)(Vdc- ADC_OFFSET_VDC-
cal_offsetA))*(int32)cal_gainA) >> 14));
      Vdc = (-1L * (Vdc * 561L)>>11) + 5 ;

      Vdc_2 = (AdcRegs.ADCRESULT2>>4);
      Vdc_2 = ((int16)((((int32)(Vdc_2- ADC_OFFSET_VDC-
cal_offsetA))*(int32)cal_gainA) >> 14));
      Vdc_2 = (-1L * (Vdc_2 * 561L)>>11) + 5 ;

       Vnp = (AdcRegs.ADCRESULT2>>4);
       Vnp = ((int16)((((int32)(Vnp- ADC_OFFSET_VDC-
cal_offsetA))*(int32)cal_gainA) >> 14));
       Vnp = (-1L * (Vdc_2 * 561L)>>11) + 5 ;


      // calibration from references
      adc_int.yHA.dc_sum += (Uint32)(AdcRegs.ADCRESULT12>>4);
      adc_int.yLA.dc_sum += (Uint32)(AdcRegs.ADCRESULT14>>4);
      adc_int.yHB.dc_sum += (Uint32)(AdcRegs.ADCRESULT13>>4);
      adc_int.yLB.dc_sum += (Uint32)(AdcRegs.ADCRESULT15>>4);
      adc_int.count_cal++;
      if (adc_int.count_cal > ADC_COUNT_CAL)
      {
            adc_int.count_cal = 0;
            adc_int.yHA.dc_sum_bak = adc_int.yHA.dc_sum;
            adc_int.yLA.dc_sum_bak = adc_int.yLA.dc_sum;
            adc_int.yHB.dc_sum_bak = adc_int.yHB.dc_sum;
            adc_int.yLB.dc_sum_bak = adc_int.yLB.dc_sum;
            adc_int.yHA.dc_sum = 0;
          adc_int.yLA.dc_sum = 0;
          adc_int.yHB.dc_sum = 0;
          adc_int.yLB.dc_sum = 0;
```

```
          adc_int.flag_cal = 1;
      }


  //  DSP_SET_SPI_MASTER();
  //  SET_SPI_MASTER();
  /*================================================================
===========
  Master to Slave SPI communications
  ================================================================
=========*/
      //DISABLE_CPLD();
      spibuf[0] = (status);                    //STATUS;
      spibuf[1] = (cmprbtop)>>8;    //SLAVE1_TOPC_HI;
      spibuf[2] = (cmprbtop);             //SLAVE1_TOPC_LO;
      spibuf[3] = (cmprbbot)>>8;    //SLAVE1_BOTC_HI;
      spibuf[4] = (cmprbbot);             //SLAVE1_BOTC_LO;
      spibuf[5] = (cmprctop)>>8;    //SLAVE2_TOPC_HI;
      spibuf[6] = (cmprctop);             //SLAVE2_TOPC_LO;
      spibuf[7] = (cmprcbot)>>8;    //SLAVE2_BOTC_HI;
      spibuf[8] = (cmprcbot);             //SLAVE2_BOTC_LO;
      checksum = 0;
      for (i=0;i<9;i++)
      checksum += spibuf[i] & 0x00FF;
      checksum = checksum & 0x00FF;
      spibuf[9] = checksum;               //CHECKSUM;

            // gtransmit == 1)
      if (loop_no > 30000) // 1.5 secs * 10kHz * 2 ISRs per cycle
      {
            // ZAKI NPC Comms: Enable the external buffers
      GpioDataRegs.GPBDAT.bit.GPIOB1 = 0;
      for (i=0;i<9;i++)
      {
            SpiaRegs.SPITXBUF = spibuf[i] << 8;
      }

      SpiaRegs.SPITXBUF = checksum << 8;
      }// END OF MASTER TO SLAVE SPI COMMS
  /*================================================================
===========
  Master to Slave SPI communications
  ================================================================
=========*/

      //hyst_band = ((10401L * (long)Vdc)/(long)real_sw_freq_serial)
;
      //hyst_band = hyst_band * 2048L / 10000L ;

      V3rd_count        = real_V3rd_serial                  ;
      V3rd_offset       = real_V3rd_offset_serial     ;
      average_offset    = average_offset_serial ;
      level_select_point = level_select_point_serial ;
      //CLEAR_TP10();

  //}
      /************************************************/
      //CLEAR_TP10();

  #ifdef GRAB_INCLUDE
      if (GrabRunning())
```

```
      {
            if ( (int_count&0x0007) == 0)
            {

                  GrabStore(0, capture_difference_A);
                  GrabStore(1,
V_half_bus_2);//zero_crossing_capture_on_A);
                  GrabStore(2, band_offset_A);
                  GrabStore(3, /*SpiaRegs.SPIRXBUF*/duty_cycle_A);
  //              GrabStore(4,
AdcRegs.ADCRESULT7);//EvbRegs.CAP4FIFO);
                  GrabStep();
            }
      }
  #endif

      // Reinitialize for next ADC interrupt
      EvaRegs.EVAIFRA.all = BIT9|BIT7;          // clear interrupt
flag
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP2;   // Acknowledge
interrupt to PIE

      prev_index_sin = index_sin;

  } /* end isr_adc */

  #ifndef BUILD_RAM
  #pragma CODE_SECTION(isr_CAP1, "ramfuncs");
  #endif
  interrupt void isr_CAP1(void)
  {
      //SET_TP11();
      temp_stack_A = EvaRegs.CAP1FBOT ;
      PWM_status_A = GpioDataRegs.GPADAT.bit.GPIOA8 ;
      duty_cycle_prev_A = duty_cycle_A ;

      if(PWM_status_A == 1)
      {
            //first_capture_new_A = EvaRegs.CAP1FIFO;
            first_capture_new_A = temp_stack_A;
            off_time_A = abs(first_capture_new_A - second_capture_A)
;

            if(off_time_A >= 65535L)
                  off_time_A = (long)off_time_A - 65535L    ;
            else if((off_time_A > 60000L) && (off_time_A < 65535L))
                  off_time_A = 65535L - (long)off_time_A ;


            duty_cycle_A = (int) (( ((100L * (on_time_A)) /
(on_time_A + off_time_A))- 0L)*1L);
            duty_cycle_A_virt = duty_cycle_A  ;


      //    if(abs(duty_cycle_A - duty_cycle_prev_A) > 20)
      //          duty_cycle_A = duty_cycle_prev_A ;

            duty_cycle_A_new_prev = duty_cycle_A_new ;
            //duty_cycle_A_new = duty_cycle_A ;
            //duty_cycle_A_new = ((512L * duty_cycle_A) - (256L *
duty_cycle_prev_A)) >> 8 ;
```

```
            //if(abs(duty_cycle_A_new - duty_cycle_A_new_prev) > 20)
duty_cycle_A_new = duty_cycle_A_new_prev ;
            //duty_cycle_A = ((duty_cycle_A * 512L) +
(duty_cycle_prev_A * 512L)) >> 10 ;

            period_a = first_capture_new_A - first_capture_old_A ;

            first_capture_old_A = first_capture_new_A ;




            if(first_capture_new_A < second_capture_A)
                  zero_crossing_capture_off_A =
((first_capture_new_A + 65535L) + second_capture_A) >> 1 ;
            else
                  zero_crossing_capture_off_A = (second_capture_A +
first_capture_new_A) >> 1 ;

            //zero_crossing_capture_A = zero_crossing_capture_off_A
;


      }
      else if (PWM_status_A == 0)
      {
            second_capture_A = temp_stack_A;
            //second_capture_A = EvaRegs.CAP1FIFO;
            on_time_A = abs(second_capture_A - first_capture_old_A)
;

            if(on_time_A >= 65535L)
                  on_time_A = on_time_A - 65535L      ;
            else if((on_time_A > 60000L) && (on_time_A < 65535L))
                  on_time_A = 65535L - on_time_A ;


            if(second_capture_A < first_capture_old_A)
                  zero_crossing_capture_on_A = ((second_capture_A +
65535L) + first_capture_old_A) >> 1 ;
            else
                  zero_crossing_capture_on_A = (second_capture_A +
first_capture_old_A) >> 1 ;

            zero_crossing_capture_A = zero_crossing_capture_on_A ;

      }
      //CLEAR_TP11();
      EvaRegs.CAPFIFOA.bit.CAP1FIFO = 1;
      EvaRegs.EVAIFRC.all = BIT0;
      //Clear interrupt flag
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;          //
Acknowledge interrupt to PIE
   }

   #ifndef BUILD_RAM
   #pragma CODE_SECTION(isr_CAP2, "ramfuncs");
   #endif
   interrupt void isr_CAP2(void)
   {
```

```
      EvaRegs.CAPFIFOA.bit.CAP2FIFO = 1;
      EvaRegs.EVAIFRC.all = BIT1;
      //Clear interrupt flag
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;          //
Acknowledge interrupt to PIE
   }

   #ifndef BUILD_RAM
   #pragma CODE_SECTION(isr_CAP4, "ramfuncs");
   #endif
   interrupt void isr_CAP4(void)
   {
    // SET_TP11();
      PWM_status_B = GpioDataRegs.GPBDAT.bit.GPIOB8 ;
      temp_stack_B = EvbRegs.CAP4FBOT ;
      duty_cycle_prev_B = duty_cycle_B ;
      if(PWM_status_B == 1)
      {
            first_capture_new_B = temp_stack_B ;
            //first_capture_new_B = EvbRegs.CAP4FIFO;
            if (first_capture_new_B < second_capture_B)
            {
                  first_capture_new_new_B = 65535L +
first_capture_new_B ;
                  zero_crossing_capture_off_B =
(first_capture_new_new_B + second_capture_B) >> 1 ;
                  duty_cycle_B = (int) (( ((100L * (second_capture_B
- first_capture_old_B)) / (first_capture_new_new_B -
first_capture_old_B))));
                  period_B = first_capture_new_new_B -
first_capture_old_B ;

            }
            else
            {
                  duty_cycle_B = (int) (( ((100L * (second_capture_B
- first_capture_old_B)) / (first_capture_new_B -
first_capture_old_B))));
                  period_B = first_capture_new_B -
first_capture_old_B ;
                  zero_crossing_capture_off_B = (
      first_capture_new_B + second_capture_B) >> 1 ;
            }
            //DAC1_FAST_WRITE(DAC_WRn_UPDn,DAC_ADDR_C, duty_cycle_B
* 5 +2047);
            first_capture_old_B = first_capture_new_B ;
      }
      else if (PWM_status_B == 0)
      {

            second_capture_B = temp_stack_B;
            //second_capture_B = EvbRegs.CAP4FIFO;
            if(second_capture_B < first_capture_old_B)
            {
                  second_capture_B = 65535L + second_capture_B ;
                  zero_crossing_capture_on_B =
(((long)(first_capture_old_B + second_capture_B)) * 512L) >> 10 ;
            }
            else
                  zero_crossing_capture_on_B =
(((long)(first_capture_old_B + second_capture_B)) * 512L) >> 10 ;
```

```
            on_time_B = second_capture_B - first_capture_old_B ;
            off_time_B_prev = off_time_B ;
            off_time_expected_B = ((off_time_B_prev>>1) + (((102L *
off_time_B  * real_V3rd_serial)>>10)))+(period_B_prev>>2) ;
      }
      //CLEAR_TP11();
      EvbRegs.CAPFIFOB.bit.CAP4FIFO = 1;
      EvbRegs.EVBIFRC.all = BIT0;
      //Clear interrupt flag
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP5;          //
Acknowledge interrupt to PIE
   }

   #ifndef BUILD_RAM
   #pragma CODE_SECTION(isr_CAP5, "ramfuncs");
   #endif
   interrupt void isr_CAP5(void)
   {
     //SET_TP11();
      PWM_status_C = GpioDataRegs.GPBDAT.bit.GPIOB9 ;
      //temp_stack_C = EvbRegs.CAP5FBOT ;
      duty_cycle_prev_C = duty_cycle_C ;
      if(PWM_status_C == 1)
      {
            first_capture_new_C = EvbRegs.CAP5FIFO;
            //first_capture_new_C = temp_stack_C;
            if (first_capture_new_C < second_capture_C)
            {
                  first_capture_new_new_C = 65535L +
first_capture_new_C ;
                  duty_cycle_C = (int) (( ((200L * (second_capture_C
- first_capture_old_C)) / (first_capture_new_new_C -
first_capture_old_C))-100L)*1L);
                  zero_crossing_capture_C =
(((long)(first_capture_new_new_C + second_capture_C)) * 512L) >>10 ;
            }

            else
            {
                  duty_cycle_C = (int) (( ((200L * (second_capture_C
- first_capture_old_C)) / (first_capture_new_C -
first_capture_old_C))-100L)*1L);
                  zero_crossing_capture_C =
(((long)(first_capture_new_C + second_capture_C)) * 512L) >>10 ;

            }

            first_capture_old_C = first_capture_new_C ;
            duty_cycle_C = (duty_cycle_C - average_offset) * 1L;

   /*       if (duty_cycle_C > 95L)
                  duty_cycle_C = 95L ;
            else if (duty_cycle_C < -95L)
                  duty_cycle_C = -95L ;
   */
            if (abs(duty_cycle_prev_C - duty_cycle_C) > 30L)
                  duty_cycle_C = duty_cycle_prev_C ;

            duty_cycle_C = ((duty_cycle_C * 512L) +
(duty_cycle_prev_C * 512L) >> 10);
```

```
            //DAC1_FAST_WRITE(DAC_WRn_UPDn,DAC_ADDR_B,(duty_cycle_C
* 10)+2047);

      }
      else if (PWM_status_C == 0)
      {
            //second_capture_C = temp_stack_C;
            second_capture_C = EvbRegs.CAP5FIFO;
            if(second_capture_C < first_capture_old_C)
            {
                  second_capture_C = 65535L + second_capture_C ;
                  zero_crossing_capture_C =
(((long)(first_capture_old_C + second_capture_C)) * 512L) >> 10 ;
            }
            else
                  zero_crossing_capture_C =
(((long)(first_capture_old_C + second_capture_C)) * 512L) >> 10 ;
      }
      //CLEAR_TP11();
      EvbRegs.CAPFIFOB.bit.CAP5FIFO = 1;
      EvbRegs.EVBIFRC.all = BIT1;
      //Clear interrupt flag
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP5;          //
Acknowledge interrupt to PIE
   }

   #ifndef BUILD_RAM
   #pragma CODE_SECTION(isr_CAP6, "ramfuncs");
   #endif
   interrupt void isr_CAP6(void)
   {

      EvbRegs.CAPFIFOB.bit.CAP6FIFO = 1;
      EvbRegs.EVBIFRC.all = BIT2;
      //Clear interrupt flag
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP5;          //
Acknowledge interrupt to PIE
   }

   #ifndef BUILD_RAM
   #pragma CODE_SECTION(isr_T2P, "ramfuncs");
   #endif
   interrupt void isr_T2P(void){
      /*
      if(master_slave_mode == 1){
            SET_SYNC_PIN();
      }
      */
      EvaRegs.EVAIFRB.all = BIT0;
      //Clear interrupt flag
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;   // Acknowledge
interrupt to PIE
   }



   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
   /**
   //AN:CAP
```

```
  The Fundamental Sine-Wave Extraction using capture port for
hysteresis variable band controller.
  \author R.Davoodnezhad
  \par History:
  \li 12/10/10 */

  /*
  #ifndef BUILD_RAM
  #pragma CODE_SECTION(isr_SPIRX, "ramfuncs");
  #endif
  interrupt void isr_SPIRX(void){
      static int i = 0;
      if(i == 1){
            i = 0;
      }
      else{
            i = 1;
      }
      SpiaRegs.SPIFFRX.bit.RXFFINTCLR = 1;      // interrupt on 3
bytes in fifo
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP6;   // Acknowledge
interrupt to PIE

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

  \author A.McIver
  \par History:
  \li 02/05/07 AM - initial creation
  */
  #ifndef BUILD_RAM
  #pragma CODE_SECTION(isr_gate_fault, "ramfuncs");
  #endif
  interrupt void isr_gate_fault(void)
  {

      PDPINT_Flag = 1 ;
      is_switching = 0;
      VSI_DISABLE();
      mod_targ = 0;
      detected_faults |= FAULT_VSI_PDPINT;
      // Acknowledge this interrupt to receive more interrupts from
group 1
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
      EvaRegs.EVAIFRA.all = BIT0;
  } /* end isr_gate_fault */


  /*
==========================================================================
=====
  __VSI_State_Functions()
  ==========================================================================
========== */
```

```
   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
   /**
   This function initialises the VSI system. It resets the target
modulation
   depth to zero.

   It is followed by the stop state.

   \author A.McIver
   \par History:
   \li 12/10/07 AM - initial creation
   */
   void st_vsi_init(void)
   {
       mod_ref = 0;
       mod_targ = 0;
       SetSwFreq(sw_freq);

       SS_NEXT(vsi_state,st_vsi_stop);
   } /* end st_vsi_init */


   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
   /**
   This is the state where the VSI is stopped. There is no switching.
It waits
   for a start trigger.

   \author A.McIver
   \par History:
   \li 12/10/07 AM - initial creation
   */
   void st_vsi_stop(void)
   {
       if (SS_IS_FIRST(vsi_state))
       {

             SS_DONE(vsi_state);
             VSI_DISABLE();
             mod_targ = 0;
             vsi_status &= ~(VSI_RUNNING|VSI_SETTLED);
       }

       if (detected_faults != 0)
       {
             SS_NEXT(vsi_state,st_vsi_fault);
             return;
       }

       if (is_switching != 0) // start trigger
       {
             SS_NEXT(vsi_state,st_vsi_gate_charge);
       }
   } /* end st_vsi_stop */


   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
   /**
```

```
  In this state the VSI gates are enabled and the low side gates
held on to
  charge the high side gate drivers. The next state is either the
ramp state.

  \author A.McIver
  \par History:
  \li 12/10/07 AM - initial creation
  */
  void st_vsi_gate_charge(void)
  {
      if (SS_IS_FIRST(vsi_state))
      {
            SS_DONE(vsi_state);
            vsi_counter = 0;
            //VSI_GATE_CHARGE();
            vsi_status |= VSI_RUNNING;
      }
      if (detected_faults != 0)
      {
            SS_NEXT(vsi_state,st_vsi_fault);
            return;
      }
      // check for stop signal
      if (is_switching == 0)
      {
            SS_NEXT(vsi_state,st_vsi_stop);
            return;
      }
      vsi_counter++;
      if (vsi_counter > 200)
      {
            SS_NEXT(vsi_state,st_vsi_ramp);
      }
  } /* end st_vsi_gate_charge */


  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
  /**
  This state ramps up the target modulation depth to match the
reference set by
  the background. It only changes the target every 100ms and
synchronises the
  change with a zero crossing to avoid step changes in the output.

  \author A.McIver
  \par History:
  \li 12/10/07 AM - initial creation
  \li 28/04/08 AM - added event reporting
  */
  void st_vsi_ramp(void)
  {
      if (SS_IS_FIRST(vsi_state))
      {
            SS_DONE(vsi_state);
            vsi_counter = 0;
            if(refMode == DC_REF|| refMode == SINGLE_AC || refMode
== SINGLE_AC_G || refMode == SINGLE_AC_PR || refMode ==
SINGLE_PI_HP){
                  VSI_ENABLE_1P();
```

```
            }
            else{
                  VSI_ENABLE();
            }
      }
      if (detected_faults != 0)
      {
            SS_NEXT(vsi_state,st_vsi_fault);
            return;
      }
      // check for stop signal
      if (is_switching == 0)
      {
            SS_NEXT(vsi_state,st_vsi_stop);
            return;
      }
      // check for target reached
      if (mod_targ == mod_ref)
      {
            SS_NEXT(vsi_state,st_vsi_run);
            return;
      }
      // ramp reference towards target
      if (mod_ref > mod_targ + 5)
      {
            mod_targ += 5;
            //Shift left of 1 introduce to deal with the fact that
mod_targ 200% at full range
            I_ref_Peak_AB =
((long)I_NOM*((long)mod_targ<<1))>>MOD_SHIFT;
      }
      else if (mod_ref < mod_targ - 5)
      {
            mod_targ -= 5;
            //Shift left of 1 introduce to deal with the fact that
mod_targ 200% at full range
            I_ref_Peak_AB =
((long)I_NOM*((long)mod_targ<<1))>>MOD_SHIFT;
      }
      else
      {
            mod_targ = mod_ref;
            //Shift left of 1 introduce to deal with the fact that
mod_targ 200% at full range
            I_ref_Peak_AB =
((long)I_NOM*((long)mod_targ<<1))>>MOD_SHIFT;
      }
   } /* end st_vsi_ramp */


   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
   /**
   This state has the VSI running with the target voltage constant.
The output is
   now ready for measurements to begin. If the reference is changed
then the
   operation moves back to the ramp state.

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

```
   \li 12/10/07 AM - initial creation
   */
   void st_vsi_run(void)
   {
       if (SS_IS_FIRST(vsi_state))
       {
             SS_DONE(vsi_state);
             vsi_status |= VSI_SETTLED;
       }
       if (detected_faults != 0)
       {
             SS_NEXT(vsi_state,st_vsi_fault);
             return;
       }
       // check for stop signal
       if (is_switching == 0)
       {
             SS_NEXT(vsi_state,st_vsi_stop);
       }
       // check for changes in reference
       if (mod_targ != mod_ref)
       {
             vsi_status &= ~VSI_SETTLED;
             SS_NEXT(vsi_state,st_vsi_ramp);
       }
   } /* end st_vsi_run */


   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
   /* void st_vsi_fault(void)
   Parameters: none
   Returns: nothing
   Description: Delays for a while after faults are cleared.
   Notes:
   History:
       03/11/05 AM - initial creation
   \li 04/03/08 AM - set vsi_status with fault bit
   \li 28/04/08 AM - added event reporting
   */
   void st_vsi_fault(void)
   {
       if (SS_IS_FIRST(vsi_state))
       {
             SS_DONE(vsi_state);
             VSI_DISABLE();
             vsi_counter = 0;
             vsi_status |= VSI_FAULT;
             vsi_status &= ~(VSI_RUNNING|VSI_SETTLED);
             putxx(detected_faults);
             put_str("->VSI faults\n");
       }
       if (detected_faults == 0)
             vsi_counter++;
       else
             vsi_counter = 0;
       if (vsi_counter > 100)
       {
             //PDPINT_Flag = 1 ;
             vsi_status &= ~VSI_FAULT;
             SS_NEXT(vsi_state,st_vsi_stop);
```

```
      }
   } /* end st_vsi_fault */


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

   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
   /**
   This function is called every fundamental period to perform the
RMS
   calculations and scale the analog quantities to Volts and Amps for
use in the
   background.

   \author A.McIver
   \par History:
   \li 12/10/07 AM - derived from IR25kVA:vsi:adc_scale
   \li 21/08/08 AM - added VSI DC offset compensation
   \li 12/09/08 AM - added stop_count and moved to floating point
data
   */
   void scale_adc_rms(void)
   {
       double
             val,
             temp;

       // calculate A0 RMS quantity
       temp =
(double)adc_int.A0.dc_sum_bak/(double)adc_int.count_rms_bak;
       val = (double)adc_int.A0.rms_sum_bak*(double)(1<<ADC_RMS_PS)
                                / (double)adc_int.count_rms_bak -
temp*temp;
       if (val < 0.0) val = 0.0;
       adc_int.A0.real = ADC_REAL_SC * sqrt(val);

   } /* end scale_adc_rms */


   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
   /**
   This function is called every ADC_DC_TIME to perform the DC
calculations and
   scale the analog quantities to Volts and Amps for use in the
background.

   \author A.McIver
   \par History:
   \li 12/10/07 AM - derived from IR25kVA:vsi:adc_scale
   */
   void scale_adc_dc(void)
   {
       double
             val;
```

```
      // calculate B0 DC quantity
      val = (double)adc_int.B0.dc_sum_bak/(double)ADC_COUNT_DC;
      adc_int.B0.real = ADC_REAL_SC * val;

  } /* end scale_adc_dc */


  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * */
  /**
  Calibrates the adc for gain and offset using the reference inputs.

  See spra989a.pdf for calibration details

  \author A.McIver
  \par History:
  \li 07/10/05 AM - initial creation
  */
  void calibrate_adc(void)
  {
  //  char
  //        str[60];
      double
            yHA = 0.0,
            yLA,
            yHB,
            yLB;

      yHA = (double)adc_int.yHA.dc_sum_bak/(double)ADC_COUNT_CAL;
      yLA = (double)adc_int.yLA.dc_sum_bak/(double)ADC_COUNT_CAL;
      yHB = (double)adc_int.yHB.dc_sum_bak/(double)ADC_COUNT_CAL;
      yLB = (double)adc_int.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 > 0.94) && (cal_gain_A < 1.05) )
            && ( (cal_gain_B > 0.94) && (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_offset_A,cal_gain_B,cal_offset_B);
  //  put_str(str);
  } /* end calibrate_adc */

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

  void get_state(void){
```

```
      if(vsi_state.f == st_vsi_init){
            put_str("INIT ");
      }
      else if(vsi_state.f == st_vsi_stop){
            put_str("STOP ");
      }
      else if(vsi_state.f == st_vsi_gate_charge){
            put_str("GATE ");
      }
      else if(vsi_state.f == st_vsi_ramp){
            put_str("RAMP ");
      }
      else if(vsi_state.f == st_vsi_run){
            put_str("RUN  ");
      }
      else if(vsi_state.f == st_vsi_fault){
            put_str("FAU  ");
      }

  }

  /* sets the switching frequency : returns switching frequency
achieved */
  /* fsw is in Hz */
  int SetSwFreq(int fsw)
  {
      unsigned int half_period;

      half_period = (unsigned int)((HSPCLK/2.0/fsw));
      //if (half_period > MAX_PER_2) half_period = MAX_PER_2;
      //else if (half_period < MIN_PER_2) half_period = MIN_PER_2;

      period_2 = half_period;
      period = period_2*2;
      sw_freq = fsw;                /* Write new switching freq to
global variable */
      MAX_TIME    =     (int16)(period_2-6) ;
      //Recalculate phase advance speed for sin table read
      PHASE_STEP_SC =   (65536.0*65536.0/(fsw*2.0));
      vsi_set_freq(Ref_freq_float);
      //If switching frequency is changed re-calculate controller
parameters
      set_KI();
      set_KP();
      return (int)((HSPCLK/2/(long)half_period + 1)/2);
  } /* end SetSwFreq */

  void step_toggle(int direction){
      if(direction == ON){
            step_tog = ON;
      }
      else if (direction == OFF){
            step_tog = OFF;
      }
  }



  void set_KP_var(double KP){
      real_KP = KP;
```

```
      set_KP();
  }

  void set_TINT_var(double TINT){
      real_TINT = TINT;
      set_KI();
  }


  void set_KP(void){

            //The division by 1<<PROP_DISCARD_BITS allow for the use
of higher proportional constant
            //then otherwise possible, there should not be
significant loss to accuracy provided
            //that the number for PROP_DISCARD_BITS is smallish, the
calculation for lost of accuracy
            //is listed in comment above

            //All together shifted left by 13 , which is then
multiple by error and shifted back right by 16 in the PI calculation
            //leaving net shift of right by 3 after that calculation
            //The period_2 multiplication with result of PI
calculation to get Command then also introduce a shift by 3 left, so
it balance out, hence
            //the need for multplication by 1<<13 in the PI constant
calculation
            //The division by I_NOM give kind of a effect of getting
the multiplication with error to become a per unit number(i.e. below
1), that is scaled by
            //power of 2, as this give a more useful no(bigger then
1)  The result then multiple by period_2 to give the required count
            //Multiplicaiton by 65536 is there so that the number
become 1 after shifted back by 16, otherwise the result of this
calculation become smaller
            //then 1, as the real_KP is a number that is rather
small (i.e. in real scale, not 2^16 == 1 scale used in DSP
calculations)
            Kp_i =
(real_KP*(double)1.0/(double)I_NOM)*(double)65536.0/(1l<<PROP_DISCAR
D_BITS)*(1l<<13);
            set_PResonant();
            set_high_pass_PI();
  }

  void set_KI(void){
      //Old controller form
      //Ki_i =
((double)1.0/((double)sw_freq*2.0)/real_TINT/(double)I_NOM)*(double)
65536.0*(double)(1l<<13)/((double)(1l<<INT_DISCARD_BITS));
      //Controller form in lecture
      Ki_i =
((double)real_KP/((double)sw_freq*2.0)/real_TINT/(double)I_NOM)*(dou
ble)65536.0*(double)(1l<<13)/((double)(1l<<INT_DISCARD_BITS));
      set_PResonant();
      set_high_pass_PI();
  //  Ki_i =
((float)period*2/(float)sw_freq/real_TINT/(float)I_NOM)*(float)I_SCA
LE;

  }
```

```
  void SetHystBand(double real_sw_freq_serial_in)
  {

  }
  void SetBusVolts(double real_sw_freq_serial_in)
  {

  }

  void set_ref_mode(unsigned int mode){
          refMode = mode;
  }

  int get_phase_step(void){
      return phase_step;
  }

  void step_ref_setup(unsigned long int phase, unsigned int
req_new_mag){
      count_from_zero_for_step = phase / phase_step;
      new_mod_targ = req_new_mag;
      //mod_ref = new_mod_targ;
      step_ref_request = 1;
  }

  void step_phase_setup(unsigned long int phase, unsigned int
step_size){
      count_from_zero_for_step = phase / phase_step;
      add_phase = step_size;
      step_phase_request = 1;
  }



  void set_Feedforward(int status){
      FFenable = status;
  }

  int get_Feedforward_status(void){
      return FFenable;
  }

  /*
  //Store in an array the inverse of bus voltage multiple by 2^13,
these numbers are used in feed forward calculations
  //and bus compensation calculations
  //Multiplication by 2^13 is because the actual calculation for
duty cycle contain a left shift by 3 which must
  //be taken into account.
  void inverse_bus_v_array_setup(void){
      int i = 0;
      for(i=0; i<= BUS_ARRAY_SIZE; i++){
          inverse_bus_v_array[i] =
(int)(((double)1/((double)(LOWER_BUS_V+i)))*(double)(1<<13));
      //    putd(inverse_bus_v_array[i]); puts(" ");

      }
  }
  */
```

```
  /*************************************************************
*****************************************************
  * Function: set_PResonant
  * Use:  Calculate the fix point coefficient used in P+Resoant
controller.  These coefficients are competitable with the
  *                     function DELTA_FILTER_2ND_ORDER.
  *   Note:
  *   The P+Resonant transfer function implemented here is of the
form:
  *   H{s} = Kp*(1 + (1/Tr)*(2*w_c*s)/(s^2 + 2*w_c*s + w_0^2))
  * This transfer function is transformed into the following form:
  * H{s} = (bs_0*s^2 + bs_1*s + bs_0)/(as_0*s^2 + as_1*s + as_2)
  * with the varibles bs_0, bs_1, bs_0, as_0, as_1, as_2 being
different coefficient of the transfer function, as define in code
  * This S domain transfer function is then transformed into a Z
domain form in floating point using Tustin transfrom
  * H{z} = (bz_0 + bz_1*z^-2 + bz_2*z^-2)/(1 + az_1*z^-1 + az_2*z^-
2)
  * Using conversion formula outline in P135 of Michael's thesis,
the Z domain function is first transformed to delta domain form,
  * then converted to fix point.
  * Delta domain transfer function is of the form:
  * H{d}=(beta_0 + beta_1*d^-1 + beta_2*d^-2)/(1 + alpha_1*d^-1 +
alpha_2*d^-2)
  *************************************************************
*****************************************************/
  void set_PResonant(void){

            Kp_i_f = real_KP;
            Ki_i_f = 1.0/real_TINT;

            //Variables for P+Resonant controller, used in both S to
Z domain transform, and Z to Delta domain transform
            delta = 1.0/((1<<LOG2_1_ON_DELTA));
            one_on_delta = 1/delta;
            w_c = w_c_f * 2.0*PI;
            w_0 = ((double)50.0)* 2.0*PI;
            Ts = 1.0/(sw_freq*2.0);

            //Defining S domain transfer function of P+Resonant
controller, the form of controller is:
            //H(s) = (bs_0*s^2 + bs_1*s + bs_0)/(as_0*s^2 + as_1*s +
as_2)
            //Transfer function define in floating point form
            bs_0 = Kp_i_f;
            bs_1 = 2.0*(Ki_i_f*Kp_i_f+w_c*Kp_i_f);
            bs_2 = Kp_i_f * w_0*w_0;

            as_0 = 1.0;
            as_1 = 2.0*w_c;
            as_2 = w_0*w_0;

            //Defining Z domain transfer function of P+Resonant
controller, the form of controller is:
            //H(z) = (bz_0 + bz_1*z^-2 + bz_2*z^-2)/(1 + az_1*z^-1 +
az_2*z^-2)
            //The S to Z transform is done using Tustin transform
            //Transfer function define in floating point form
            az_0 = (4.0/(Ts*Ts)*as_0 + as_1 *2.0/Ts + as_2);

            bz_0 = (4.0/(Ts*Ts)*bs_0 + bs_1 * 2.0/Ts + bs_2)/az_0;
```

```
            bz_1 = (2.0*bs_2 - bs_0 * 8.0/(Ts*Ts))/az_0;
            bz_2 = (4.0/(Ts*Ts)*bs_0 - bs_1 * 2.0/Ts + bs_2)/az_0;

            az_1 = (2*as_2 - as_0 * 8.0/(Ts*Ts))/az_0;
            az_2 = (4.0/(Ts*Ts)*as_0 - as_1 *2.0/Ts + as_2)/az_0;

            //Z to Delta domain transformation
            beta_0_f = bz_0;
            beta_1_f = (2.0*bz_0 + bz_1)/delta;
            beta_2_f = (bz_0 + bz_1 + bz_2)/(delta * delta);

            alpha_1_f = (2.0 + az_1)/delta;
            alpha_2_f = (1.0 + az_1 + az_2)/(delta*delta);

            //Delta transform from floating point to fix point
            alpha_0 = 1<<LOG2_ALPHA_0;

            beta_0 = beta_0_f * (double)alpha_0+0.5;
            beta_1 = beta_1_f * (double)alpha_0+0.5;
            beta_2 = beta_2_f * (double)alpha_0+0.5;

            alpha_1 = alpha_1_f *  (double)alpha_0+0.5;
            alpha_2 = alpha_2_f *  (double)alpha_0+0.5;

  }

  void set_compensation_filter(void){

            double K = 1.0, fc = 20.0, zeta = 30.0;

            //Variables for P+Resonant controller, used in both S to
Z domain transform, and Z to Delta domain transform
            delta = 1.0/((1<<LOG2_1_ON_DELTA));
            one_on_delta = 1/delta;
            w_c = fc * 2.0*PI;
            Ts = 1.0/(sw_freq*2.0);

            //Defining S domain transfer function of P+Resonant
controller, the form of controller is:
            //H(s) = (bs_0*s^2 + bs_1*s + bs_0)/(as_0*s^2 + as_1*s +
as_2)
            //Transfer function define in floating point form
            bs_0 = 0.0;
            bs_1 = 0.0;
            bs_2 = K * w_c*w_c;

            as_0 = 1.0;
            as_1 = 2.0*zeta*w_c;
            as_2 = w_c*w_c;

            //Defining Z domain transfer function of P+Resonant
controller, the form of controller is:
            //H(z) = (bz_0 + bz_1*z^-2 + bz_2*z^-2)/(1 + az_1*z^-1 +
az_2*z^-2)
            //The S to Z transform is done using Tustin transform
            //Transfer function define in floating point form
            az_0 = (4.0/(Ts*Ts)*as_0 + as_1 *2.0/Ts + as_2);

            bz_0 = (4.0/(Ts*Ts)*bs_0 + bs_1 * 2.0/Ts + bs_2)/az_0;
            bz_1 = (2.0*bs_2 - bs_0 * 8.0/(Ts*Ts))/az_0;
            bz_2 = (4.0/(Ts*Ts)*bs_0 - bs_1 * 2.0/Ts + bs_2)/az_0;
```

```
            az_1 = (2*as_2 - as_0 * 8.0/(Ts*Ts))/az_0;
            az_2 = (4.0/(Ts*Ts)*as_0 - as_1 *2.0/Ts + as_2)/az_0;

            //Z to Delta domain transformation
            beta_0_f = bz_0;
            beta_1_f = (2.0*bz_0 + bz_1)/delta;
            beta_2_f = (bz_0 + bz_1 + bz_2)/(delta * delta);

            alpha_1_f = (2.0 + az_1)/delta;
            alpha_2_f = (1.0 + az_1 + az_2)/(delta*delta);

            //Delta transform from floating point to fix point
            alpha_com_0 = 1<<LOG2_ALPHA_0_COM;

            beta_com_0 = (beta_0_f * (double)alpha_com_0+0.5);
            beta_com_1 = (beta_1_f * (double)alpha_com_0+0.5);
            beta_com_2 = (beta_2_f * (double)alpha_com_0+0.5);

            alpha_com_1 = (alpha_1_f *  (double)alpha_com_0+0.5);
            alpha_com_2 = (alpha_2_f *  (double)alpha_com_0+0.5);

    }

    void set_high_pass_PI (void){
            double P = 20*PI;
            double one = 65536.0;
            Ts = 1.0/(5000.0*2.0);

            Kp_i_f = real_KP;
            Ki_i_f = 1.0/real_TINT;


            /*a0 = Ts*Kp_i_f*Ki_i_f / (P*Ts+2.0) / (double)I_NOM;
            a1 = Ts*Kp_i_f*Ki_i_f / (P*Ts+2.0) / (double)I_NOM;
            b1 = -(P*Ts-2.0)/(P*Ts+2.0) / (double)I_NOM;
            Kp_double = Kp_i_f / (double)I_NOM;*/

            a0 = Ts*Kp_i_f*Ki_i_f / (P*Ts+2.0) ;
            a1 = Ts*Kp_i_f*Ki_i_f / (P*Ts+2.0) ;
            b1 = -(P*Ts-2.0)/(P*Ts+2.0) ;
            Kp_double = Kp_i_f ;

            a0_fp = (int32)(a0 * one);
            a1_fp = (int32)(a1 * one);
            b1_fp = (int32)(b1 * one);
            Kp_fp = (int32)(Kp_double * one);
            one_on_INOM = one / (double)I_NOM;

    }


    void set_RefMode(int mode){
        refMode = mode;
    }

    int get_Ref_mode(void){
        return refMode;
    }

    void display_ref_mode(void){
```

```
    if(refMode == DC_REF){
          put_str("DC PI ");
    }
    else if(refMode == SINGLE_AC){
          put_str("1P PI ");
    }
    else if(refMode == THREE_PHASE_PI){
          put_str("3P PI ");
    }
    else if(refMode == SINGLE_AC_G){
          put_str("1 PI G ");
    }
    else if(refMode == SINGLE_AC_OL){
          put_str("1P OL ");
    }
    else if(refMode == THREE_PHASE_DQ){
          put_str("3P DQ ");
    }
    else if(refMode == THREE_PHASE_PI_G){
          put_str("3P G  ");
    }
    else if(refMode == THREE_PHASE_HS){
          put_str("3P_HYS ");
    }
    else if(refMode == SINGLE_PHASE_HS){
          put_str("1P_HYS ");
    }
    else if(refMode == SINGLE_AC_PR){
          put_str("1P PR ");
    }
    else if(refMode == THREE_PHASE_PR){
          put_str("3P PR ");
    }
    else if(refMode == SQUARE_WAVE){
          put_str("SQ WV ");
    }
    else if(refMode == THREE_PHASE_OL){
          put_str("3P OL ");
    }
    else if(refMode == PHASE_A){
          put_str("P_A ");
    }
    else if(refMode == PHASE_B){
          put_str("P_B ");
    }
    else if(refMode == PHASE_C){
          put_str("P_C ");
    }
    else if(refMode == SINGLE_PI_HP){
          put_str("HP PI ");
    }
    else if(refMode == THREE_PH_2CONT_HS){
          put_str("3PH_2CNT ");
    }
    else if(refMode == SPH_3L_HCC){
          put_str("SPH_3L ");
    }
}
```

Bibliography

[1] N.A. Moguilnaia, K.V. Vershinin, M.R. Sweet, O.I. Spulber, M.M. De Souza, E.M.S. Narayanan, "Innovation in power semiconductor industry: past and future, *"Engineering Management, IEEE Transactions on*, vol.52, no.4, pp. 429- 439, Nov. 2005.

[2] E.K. Sato, M. Kinoshita, Y. Yamamoto, T. Amboh, "High efficiency multilevel uninterruptible power supply, "*Energy Conversion Congress and Exposition, 2009. ECCE 2009. IEEE*, vol., no., pp.3143-3150, 20-24 Sept. 2009.

[3] M. P. Kazmierkowski and L. Malesani, "Current control techniques for three-phase voltage-source PWM converters: a survey", *IEEE Trans. on Ind. Electron*, Vol. 45, No. 5, 1998, pp.691-703.

[4] T. M. Rowan and R. J. Kerkman, "A new synchronous current regulator and an analysis of current-regulated PWM inverters", *IEEE Trans. Ind. Applicat*., Vol. IA-22, No. 4, Jul./Aug. 1986, pp. 678 – 690.

[5] N. Mohan, T. M. Undeland, and W. P. Robbins, *Power Electronics: Converters, Applications, and Design*, 3rd ed. Hoboken: John Wiley & Sons, INC., 2002.

[6] Lai Jih-Sheng, Fang Zheng Peng, "Multilevel converters-a new breed of power converters, "*Industry Applications, IEEE Transactions on*, vol.32, no.3, pp.509-517, May/Jun 1996.

[7] J. Rodriguez, Lai Jih-Sheng, Fang Zheng Peng, "Multilevel inverters: a survey of topologies, controls, and applications," *Industrial Electronics, IEEE Transactions on*, vol.49, no.4, pp. 724- 738, Aug 2002.

[8] J. Rodriguez, L.G. Franquelo, S. Kouro, J.I. Leon, R.C. Portillo, M.A.M. Prats, M.A. Perez, "Multilevel Converters: An Enabling Technology for High-Power Applications" *Proceedings of the IEEE*, vol.97, no.11, pp.1786-1817, Nov. 2009.

[9] T. Kerekes, M. Liserre, R. Teodorescu, C. Klumpner, M. Sumner, "Evaluation of Three-Phase Transformerless Photovoltaic Inverter Topologies," *Power Electronics, IEEE Transactions on*, vol.24, no.9, pp.2202-2211, Sept. 2009.

[10] Wu Libo, Zhao Zhengming, Liu Jianzheng, "A Single-Stage Three-Phase Grid-Connected Photovoltaic System With Modified MPPT Method and Reactive Power Compensation," *Energy Conversion, IEEE Transactions on*, vol.22, no.4, pp.881-886, Dec. 2007.

[11] D. Holmes and T. Lipo, Pulse Width Modulation for Power Converters: Principles and Practice. 2003.

[12] J. Holtz, "Pulsewidth modulation-a survey, "*Power Electronics Specialists Conference, 1992. PESC '92 Record., 23rd Annual IEEE*, vol., no., pp.11-18 vol.1, 29 Jun-3 Jul 1992.

[13] H.W. van der Broeck, H.-C. Skudelny, G.V. Stanke, "Analysis and realization of a pulsewidth modulator based on voltage space vectors," *Industry Applications, IEEE Transactions on*, vol.24, no.1, pp.142-150, Jan/Feb 1988.

[14] D. W. Novotny and T. A. Lipo, Vector Control and Dynamics of AC Drives, Oxford Science Publications, 1996.

[15] B. P. McGrath, D. G. Holmes, and T. Lipo, "Optimized space vector switching sequences for multilevel inverters," *Power Electronics, IEEE Transactions on*, vol. 18, no. 6, pp. 1293-1301, 2003.

[16] A.M. Hava, R.J. Kerkman, T.A. Lipo, "A high-performance generalized discontinuous PWM algorithm," *Industry Applications, IEEE Transactions on*, vol.34, no.5, pp.1059-1071, Sep/Oct 1998.

[17] Dae-Wook Kang, Yo-Han Lee, Bum-Seok Suh, Chang-Ho Choi, and Dong-Seok Hyun, "An improved carrier-based SVPWM method using leg voltage redundancies in generalized cascaded multilevel inverter topology," *Power Electronics, IEEE Transactions on*, vol. 18, no. 1, pp. 180-187, 2003.

[18] R. Teodorescu, F. Blaabjerg, J. K. Pedersen,E. Cengelci, S. Sulistijo, B. Woo, and P. Enjeti, "Multilevel converters a survey," *in Proc. Eur. Power Electron. Conf.*, Lausanne, Switzerland, 1999.

[19] A. Nabae, I. Takahashi, H. Akagi, "A New Neutral-Point-Clamped PWM Inverter, "*Industry Applications, IEEE Transactions on*, vol.IA-17, no.5, pp.518-523, Sept. 1981.

[20] T.A. Meynard, H. Foch, "Multi-level conversion: high voltage choppers and voltage-source inverters," *Power Electronics Specialists Conference, 1992. PESC '92 Record., 23rd Annual IEEE*, vol., no., pp.397-403.

[21] M. Marchesoni, M. Mazzucchelli, S. Tenconi, "A nonconventional power converter for plasma stabilization, "*Power Electronics, IEEE Transactions on*, vol.5, no.2, pp.212-219, Apr 1990.

[22] R.H. Wilkinson, T.A. Meynard, H. du Toit Mouton, "Natural Balance of Multicell Converters: The General Case," *Power Electronics, IEEE Transactions on*, vol.21, no.6, pp.1658-1666, Nov. 2006.

[23] T.A. Meynard, H. Foch, P. Thomas, J. Courault, R. Jakob, M. Nahrstaedt, M. "Multicell converters: basic concepts and industry applications, "*Industrial Electronics, IEEE Transactions on*, vol.49, no.5, pp. 955- 964, Oct 2002.

[24] B.P. McGrath, "Topologically independent modulation of multilevel inverters," Ph.D. Thesis, Monash University, Australia, 2002.

[25] T. A. Meynard and H. Foch, "Electronic device for electrical energy conversion between a voltage source and a current source by means of controllable switching cells", *IEEE Trans. Ind. Electron*., vol. 49, pp. 955–964, Oct. 2002.

[26] P.W. Hammond, "A new approach to enhance power quality for medium voltage AC drives," *Industry Applications, IEEE Transactions on*, vol.33, no.1, pp.202-208, Jan/Feb 1997.

[27] G. Carrara, S. Gardella, M. Marchesoni, R. Salutari, and G. Sciutto, "A new multilevel PWM method: a theoretical analysis," *Power Electronics, IEEE Transactions on*, vol. 7, no. 3, pp. 497-505, 1992.

[28] Z. Mohzani, B.P. McGrath, D.G. Holmes, "Natural balancing of the Neutral Point voltage for a three-phase NPC multilevel converter," *IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society*, vol., no., pp.4445-4450, 7-10 Nov. 2011.

[29] C. Newton and M. Sumner, "Neutral point control for multi-level inverters: theory, design and operational limitations," in *Industry Applications Conference, 1997. Thirty-Second IAS Annual Meeting, IAS '97., Conference Record of the 1997 IEEE*, 1997, vol. 2, pp. 1336-1343 vol.2.

[30] J. Rodriguez, S. Bernet, P. K. Steimer, and I. E. Lizama, "A Survey on Neutral-Point-Clamped Inverters," *Industrial Electronics, IEEE Transactions on*, vol. 57, no. 7, pp. 2219-2230, 2010.

[31] H. du Toit Mouton, "Natural balancing of three-level neutral-point clamped PWM inverters," *Industrial Electronics, IEEE Transactions on*, vol. 49, no. 5, pp. 1017-1025, 2002.

[32] Z. Mohzani, B.P. McGrath, D.G. Holmes, "The balancing properties of DC link compensation for 3-phase Neutral Point Clamped (NPC) Converter," *Power Electronics and Motion Control Conference (IPEMC), 2012 7th International*, vol.1, no., pp.574-579, 2-5 June 2012.

[33] Poh Chiang Loh, D. G. Holmes, Y. Fukuta, and T. A. Lipo, "Reduced common-mode modulation strategies for cascaded multilevel inverters," *Industry Applications, IEEE Transactions on*, vol. 39, no. 5, pp. 1386-1395, 2003.

[34] B.P. McGrath and D.G. Holmes, "A comparison of multicarrier PWM strategies for cascaded and neutral point clamped multilevel inverters," in *Power Electronics Specialists Conference, 2000. PESC 00. 2000 IEEE 31st Annual*, 2000, vol. 2, pp. 674-679 vol.2.

[35] B.P McGrath, T. Meynard, G. Gateau, D.G. Holmes, "Optimal Modulation of Flying Capacitor and Stacked Multicell Converters Using a State Machine Decoder," *Power Electronics, IEEE Transactions on*, vol.22, no.2, pp.508-516, March 2007.

[36] B.P. McGrath and D.G. Holmes, "Enhanced Voltage Balancing of a Flying Capacitor Multilevel Converter Using Phase Disposition (PD) Modulation," *Power Electronics, IEEE Transactions on*, vol.26, no.7, pp.1933-1942, July 2011.

[37] C. Seo, D. Choi and D. Hyun, "A new simplified space-vector PWM method for three-level inverters," *IEEE Trans. on Power Electronics*, vol. 16, July 2001,pp. 545-550.

[38] A novel SVM algorithm for multilevel three-phase converters," in Conf. Rec. IEEE 33rd Annual Power Electronics Specialists Conference, Cairns, 2002, vol. 2, pp. 509-513.

[39] N. Celanovic and D. Boroyevich, "A fast space vector modulation algorithm for multilevel three-phase converters," *IEEE Trans. on Industry Applications*, vol. 37, no. 2,2001,pp. 637-641.

[40] P. Mattavelli, "Synchronous-frame harmonic control for high-performance AC power supplies," *Industry Applications, IEEE Transactions on*, vol. 37, no. 3, pp. 864-872, 2001.

[41] P. Mattavelli, F. Polo, S. Sattin, and F. DaI Lago, "Dynamic improvement in UPS by means of control delay minimization," in *Industry Applications Conference, 2004. 39th IAS Annual Meeting. Conference Record of the 2004 IEEE*, 2004, vol. 2, pp. 843-849 vol.2.

[42] P. Mattavelli, S. Fasolo, "Implementation of synchronous frame harmonic control for high-performance AC power supplies," *Industry Applications Conference, 2000. Conference Record of the 2000 IEEE*, vol.3, no., pp.1988-1995 vol.3, 2000.

[43] S. Buso and P. Mattavelli, *Digital Control in Power Electronics*, 1st ed. United States of America: Morgan and Claypool Publishers, 2006.

[44] G. F. Franklin, J. D. Powell, and M. L. Workman, *Digital control of dynamic systems, 3rd ed*, 3rd ed. Menlo Park, Calif.: Addison-Wesley, 1998.

[45] T. Yoshii, S. Inoue, and H. Akagi, "Control and Performance of a Medium-Voltage Transformerless Cascade PWM STATCOM with Star-Configuration," in *Industry Applications Conference, 2006. 41st IAS Annual Meeting. Conference Record of the 2006 IEEE*, 2006, vol. 4, pp. 1716-1723.

[46] R. Teodorescu, F. Blaabjerg, U. Borup, and M. Liserre, "A new control structure for grid-connected LCL PV inverters with zero steady-state error and selective harmonic compensation," in *Applied Power Electronics Conference and Exposition, 2004. APEC '04. Nineteenth Annual IEEE*, 2004, vol. 1, pp. 580-586 Vol.1.

[47] B.K. Bose, "An adaptive hysteresis-band current control technique of a voltage-fed PWM inverter for machine drive system," *Industrial Electronics, IEEE Transactions on*, vol.37, no.5, pp.402-408, Oct 1990.

[48] D. Casadei, F. Profumo, G. Serra, A. Tani, "FOC and DTC: two viable schemes for induction motors torque control," *Power Electronics, IEEE Transactions on*, vol.17, no.5, pp. 779- 787, Sep 2002.

[49] C. French, P. Acarnley, "Direct torque control of permanent magnet drives," *Industry Applications, IEEE Transactions on*, vol.32, no.5, pp.1080-1088, Sep/Oct 1996.

[50] M-W. Naouar, E. Monmasson, A.A. Naassani, I. Slama-Belkhodja, N. Patin, "FPGA-Based Current Controllers for AC Machine Drives—A Review," *Industrial Electronics, IEEE Transactions on*, vol.54, no.4, pp.1907-1925, Aug. 2007.

[51] M. Brod and D. W. Novotny, "Current Control of VSI-PWM Inverters," *Industry Applications, IEEE Transactions on*, vol. 21, no. 3, pp. 562-570, 1985.

[52] D. G. Holmes, T. A. Lipo, B. P. McGrath, and W. Y. Kong, "Optimized Design of Stationary Frame Three Phase AC Current Regulators," *Power Electronics, IEEE Transactions on*, vol. 24, no. 11, pp. 2417-2426, 2009.

[53] D. N. Zmood and D. G. Holmes, "Stationary frame current regulation of PWM inverters with zero steady-state error," *Power Electronics, IEEE Transactions on*, vol. 18, no. 3, pp. 814-822, 2003.

[54] D. N. Zmood, D. G. Holmes, and G. H. Bode, "Frequency-domain analysis of three-phase linear current regulators," *Industry Applications, IEEE Transactions on*, vol. 37, no. 2, pp. 601-610, 2001.

[55] R. F. de Camargo and H. Pinheiro, "Comparison of Six Digital Current Control Techniques for Three-Phase Voltage-Fed PWM Converters Connected to the Utility Grid," in *Power Electronics Specialists Conference, 2005. PESC '05. IEEE 36th*, 2005, pp. 1422-1428.

[56] R. D. Lorenz and D. B. Lawson, "Performance of feedforward current regulators for field oriented induction machine controllers," *IEEE Trans. Ind. Applicat.*, vol. IA-23, pp. 597–602, July/Aug. 1987.

[57] C. Andrieux and M. Lajoie-Mazenc, "Analysis of different current control systems for inverter-fed synchronous machine," in *Proc. EPE Conf.*, Brussels, Belgium, 1985, pp. 2.159–2.165.

[58] W.Y.Kong, D. G. Holmes, and B. P. McGrath, "Improved stationary frame AC current regulation using feedforward compensation of the load EMF," in *Proc. IEEE APEC-09, IEEE Appl. Power Electron. Conf.*, Washington, DC, CD record, 2009, pp. 145–151.

[59] D.G. Holmes, D.A Martin, "Implementation of a direct digital predictive current controller for single and three phase voltage source inverters,"Industry App. Conference, 1996. Thirty-First IAS Annual Meeting, IAS '96., Conference Record of the 1996 IEEE, vol.2, no., pp.906-913 vol.2, 6-10 Oct 1996.

[60] C. D. Schauder and R. Caddy, "Current Control of Voltage-Source Inverters for Fast Four-Quadrant Drive Performance," *Industry Applications, IEEE Transactions on*, vol. 18, no. 2, pp. 163-171, 1982.

[61] L. Norum, W. Sulkowski, and L. A. Aga, "Compact realization of PWM-VSI current controller for PMSM drive application using low cost standard microcontroller," in *Conf. Rec. IEEE PESC'92*, Toledo, Spain, 1992, pp. 680–685.

[62] C. T. Rim, N. S. Choi, G. C. Cho, and G. H. Cho, "A complete DC and AC analiysis of three-phase controlled-current PWM rectifer using circuit D-Q transformation," *IEEE Trans. Power Electron.*, vol. 9, pp. 390–396, July 1994.

[63] R. B. Sepe and J. H. Lang, "Inverter nonlinearities and discrete-time vector current control," *IEEE Trans. Ind. Applicat.*, vol. 30, pp. 62–70, Jan./Feb. 1994.

[64] L. Ben-Brahim and A. Kawamura, "Digital current regulation of fieldoriented controlled induction motor based on predictive flux observer," in *Conf. Rec. IEEE-IAS Annu. Meeting*, 1990, pp. 607–612.

[65] J. W. Kolar, H. Ertl, and F. C. Zach, "Analysis of on- and off-line optimized predictive current controllers for PWM converter system," *IEEE Trans. Power Electron.*, vol. 6, pp. 454–462, May 1991.

[66] L. Malesani, P. Tenti, "A novel hysteresis control method for current-controlled voltage-source PWM inverters with constant modulation frequency," *Industry Applications, IEEE Transactions on*, vol.26, no.1, pp.88-92, Jan/Feb 1990.

[67] Q. Yao and D.G. Holmes, "A simple, novel method for variable-hysteresis-band current control of a three phase inverter with constant switching frequency," *Industry Applications Society Annual Meeting, 1993., Conference Record of the 1993 IEEE*, vol., no., pp.1122-1129 vol.2, 2-8 Oct 1993.

[68] R.B. Sepe, Jr., "A unified approach to hysteretic and ramp-comparison current controllers," *Industry Applications Society Annual Meeting, 1993., Conference Record of the 1993 IEEE*, vol., no., pp.724-731 vol.1, 2-8 Oct 1993.

[69] M. A. Rahman, T. S. Radwan, A. M. Osheiba, and A. E. Lashine, "Analysis of current controllers for voltage-source inverter," *IEEE Trans. Ind. Electron.*, vol. 44, No. 4, pp. 477-485, Aug. 1997.

[70] S. Srikanthan, M.K. Mishra, "Constant frequency current control using a ramp comparison method for a DSTATCOM application," *TENCON 2008 - 2008 IEEE Region 10 Conference*, vol., no., pp.1-6, 19-21 Nov. 2008.

[71] B.K. Bose, "An adaptive hysteresis-band current control technique of a voltage-fed PWM inverter for machine drive system," *Industrial Electronics, IEEE Transactions on*, vol.37, no.5, pp.402-408, Oct 1990.

[72] Kawamura, Atsuo, Hoft, Richard, "Instantaneous Feedback Controlled PWM Inverter with Adaptive Hysteresis," *Industry Applications, IEEE Transactions on*, vol.IA-20, no.4, pp.769-775, July 1984.

[73] L. Malesani, L. Rossetto, L. Sonaglioni, P. Tomasin, and A. Zuccato, "Digital, adaptive hysteresis current control with clocked commutations and wide operating range," *IEEE Trans. Ind. Applicat.*, vol. 32, pp. 1115–1121, Mar./Apr. 1996.

[74] L. Malesani, P. Tenti, E. Gaio, and R. Piovan, "Improved current control technique of VSI PWM inverters with constant Modulation frequency and extended voltage range," *IEEE Trans. Ind. Applicat.*, vol. 27, pp. 365–369, Mar./Apr. 1991.

[75] Nagy, I., "Novel adaptive tolerance band based PWM for field-oriented control of induction machines," *Industrial Electronics, IEEE Transactions on*, vol.41, no.4, pp.406-417, Aug 1994.

[76] A. Tripathi, P.C. Sen, "Comparative analysis of fixed and sinusoidal band hysteresis current controllers for voltage source inverters," *Industrial Electronics, IEEE Transactions on*, vol.39, no.1, pp.63-73, Feb 1992.

[77] L. Malesani, P. Mattavelli, P. Tomasin, "Improved constant-frequency hysteresis current control of VSI inverters with simple feedforward bandwidth prediction," *Industry Applications, IEEE Transactions on*, vol.33, no.5, pp.1194-1202, Sep/Oct 1997.

[78] L. Malesani, P. Mattavelli, P. Tomasin, "High-performance hysteresis modulation technique for active filters ,"*Power Electronics, IEEE Transactions on*, vol.12, no.5, pp.876-884, Sep 1997.

[79] L.J. Borle, C.V. Nayar, "ZACE current controlled power flow for AC-DC power converters," *Power Electronics Specialists Conference, PESC '94 Record., 25th Annual IEEE*, vol., no., pp.539-545 vol.1, 20-25 Jun 1994.

[80] L.J. Borle, C.V. Nayar, "Zero average current error controlled power flow for AC-DC power converters, "*Power Electronics, IEEE Transactions on*, vol.10, no.6, pp.725-732, Nov 1995.

[81] G.H. Bode, D.G. Holmes, "Improved current regulation for voltage source inverters using zero crossings of the compensated current errors," *Industry Applications Conference, 2001. Thirty-Sixth IAS Annual Meeting. Conference Record of the 2001 IEEE*, vol.2, no., pp.1007-1014 vol.2, Sept. 30 2001-Oct. 4 2001.

[82] G.H. Bode, D.G. Holmes, "Load independent hysteresis current control of a three level single phase inverter with constant switching frequency," *Power Electronics Specialists Conference, 2001. PESC. 2001 IEEE 32nd Annual*, vol.1, no., pp.14-19 vol. 1, 2001.

[83] L. Sonaglioni, "Predictive digital hysteresis current control," *Industry Applications Conference, 1995. Thirtieth IAS Annual Meeting, IAS '95., Conference Record of the 1995 IEEE*, vol.3, no., pp.1879-1886 vol.3, 8-12 Oct 1995.

[84] S. Buso, S. Fasolo, L. Malesani and P. Mattavelli, "A Dead-Beat Adaptive Hysteresis Current Control," *IEEE Transactions on Industry Applications,* Vol. 36 No. 4, July/Aug 2000, pp. 1174-1180.

[85] W. Stefanutti, P. Mattavelli, "Fully digital hysteresis modulation with switching-time prediction," *Industry Applications, IEEE Transactions on*, vol.42, no.3, pp. 763- 769, May-June 2006.

[86] A. Nabae, S. Ogasawara, H. Akagi, "A Novel Control Scheme for Current-Controlled PWM Inverters," *Industry Applications, IEEE Transactions on*, vol.IA-22, no.4, pp.697-701, July 1986.

[87] M.P. Kazmierkowski, M.A. Dzieniakowski, W. Sulkowski, "Novel space vector based current controllers for PWM-inverters," *Power Electronics Specialists Conference, 1989. PESC '89 Record., 20th Annual IEEE*, vol., no., pp.657-664 vol.2, 26-29 Jun 1989.

[88] Ching-Tsai Pan, Ting-Yu Chang, "An improved hysteresis current controller for reducing switching frequency," *Power Electronics, IEEE Transactions on*, vol.9, no.1, pp.97-104, Jan 1994.

[89] A. Tilli, A. Tonielli, "Sequential design of hysteresis current controller for three-phase inverter," *Industrial Electronics, IEEE Transactions on*, vol.45, no.5, pp.771-781, Oct 1998.

[90] E. Aldabas, L. Romeral, A. Arias, M.G. Jayne, "Software-based digital hysteresis-band current controller," *Electric Power Applications, IEE Proceedings* -, vol.153, no.2, pp. 184- 190, 2 March 2006.

[91] B-H Kwon, T-W Kim, J-H Youm, "A novel SVM-based hysteresis current controller," *Power Electronics, IEEE Transactions on*, vol.13, no.2, pp.297-307, Mar 1998.

[92] F. Zare, G. Ledwich, "A New Hysteresis Current Control for Three-phase Inverters Based on Adjacent Voltage Vectors and Time Error," *Power Electronics Specialists Conference, 2007. PESC 2007. IEEE*, vol., no., pp.431-436, 17-21 June 2007.

[93] M. Mohseni, S.M. Islam, "A New Vector-Based Hysteresis Current Control Scheme for Three-Phase PWM Voltage-Source Inverters," *Power Electronics, IEEE Transactions on*, vol.25, no.9, pp.2299-2309, Sept. 2010.

[94] A. Veltman, P.P.J. van den Bosch and R.J.A. Gorter, "On-line Optimal Switching Patterns for 2-level and 3-level Inverters using the Fish Method", in *Conf. Rec. IEEE Power Electronics Specialists Conf*, pp.1063-1069, 1993.

[95] D.T.W. Liang, Jiang Li, "Flux vector modulation strategy for a four-switch three-phase inverter for motor drive applications," *Power Electronics Specialists Conference, 1997. PESC '97 Record., 28th Annual IEEE*, vol.1, no., pp.612-617 vol.1, 22-27 Jun 1997.

[96] P. C. Loh and D. G. Holmes, "A multidimensional variable band flux modulator for four-phase-leg voltage source inverters," *IEEE Trans. Power Electron.*, vol. 18, no. 2, pp. 628–635, Mar. 2003.

[97] D.C. Patel, R.R. Sawant, M.C. Chandorkar, "Three-Dimensional Flux Vector Modulation of Four-Leg Sine-Wave Output Inverters," *Industrial Electronics, IEEE Transactions on*, vol.57, no.4, pp.1261-1269, April 2010.

[98] P. C. Loh and D. G. Holmes, "A variable band universal flux/charge modulator for VSI and CSI modulation," *IEEE Trans. Ind. Appl.*, vol. 38, no. 3, pp. 695–705, May/Jun. 2002.

[99] Xie Hailian, L. Angquist, H.-P. Nee, "Novel flux modulated positive and negative sequence deadbeat current control of voltage source converters," *Power Engineering Society General Meeting, 2006. IEEE*, vol., no., pp.8 pp., 0-0 0

[100] M. Marchesoni, "High performance current control techniques for applications to multilevel high-power voltage source inverters," *IEEE Trans. Power Electron.*, pp. 189–204, vol. 7, no. 1, Jan. 1992.

[101] G.H. Bode, D.N. Zmood, P.C. Loh, D.G. Holmes, "A novel hysteresis current controller for multilevel single phase voltage source inverters," *Power Electronics Specialists Conference, 2001. PESC. 2001 IEEE 32nd Annual*, vol.4, no., pp.1845-1850 vol. 4, 2001.

[102] F. Zare and G. Ledwich, "A hysteresis current control for single phase multilevel voltage source inverters: PLD implementation," *IEEE Trans. Power Electron.*, vol. 17, no. 5, pp. 731–738, Sep. 2002.

[103] P. C. Loh, G. H. Bode, and P. C. Tan, "Modular hysteresis current control of hybrid multilevel inverters," in *Proc. IEE Electric. Power Appl.*, vol. 152, Jan. 2005, pp. 1–8.

[104] R. Gupta, A. Ghosh, and A. Joshi, "Cascaded multilevel control of DSTATCOM using multi-band hysteresis modulation," in *Proc. IEEE Power Eng. Soc. General Meet.*, Jun. 2006, pp. 1–7.

[105] F. Zare, S. Zabihi, G. Ledwich, "An adaptive hysteresis current control for a multilevel inverter used in an active power filter," *Power Electronics and Applications, 2007 European Conference on*, vol., no., pp.1-8, 2-5 Sept. 2007.

[106] R. Gupta, A. Ghosh, and A. Joshi, "Switching characterization of cascaded multilevel-inverter-controlled systems," *IEEE Trans. Ind. Electron.*, vol. 55, no. 3, pp. 1047–1058, Mar. 2008.

[107] G. H. Bode and D. G. Holmes, "Implementation of three level hysteresis current control for a single phase voltage source inverter," in *Proc. IEEE PESC*, 2000, pp. 33–38.

[108] G.H. Bode, D.G. Holmes, "Load independent hysteresis current control of a three level single phase inverter with constant switching frequency," *Power Electronics Specialists Conference, 2001. PESC. 2001 IEEE 32nd Annual*, vol.1, no., pp.14-19 vol. 1, 2001.

[109] P. C. Loh, G. H. Bode, D.G.Holmes, and T.A. Lipo, "A time-based double band hysteresis current regulation strategy for single phase multilevel inverters," *IEEE Trans. Ind. Appl.*, vol. 39, no. 3, pp. 883–892, May/Jun. 2003.

[110] Shukla, A., Ghosh, A., Joshi, A., "Hysteresis Modulation of Multilevel Inverters," *Power Electronics, IEEE Transactions on*, vol.26, no.5, pp.1396-1409, May 2011.

[111] A. Shukla, A. Ghosh, and A. Joshi, "Improved multilevel hysteresis current regulation and capacitor voltage balancing schemes for flying capacitor multilevel inverter,," *IEEE Trans. Power Electron.*, vol. 23, no. 2, pp. 518–529, Mar. 2008.

[112] M. R. Baiju, K. Gopakumar, L. Umanand, and A. Pittet, "Multi axis space phasor based multilevel current hysteresis controller for an open-end winding induction motor fed from dual inverters," in *Proc. 29 European Conf. on Intelligent Motion, PCIM'02*, Nurnberg, Germany, 2002, pp. 405-410.

[113] P.N. Tekwani, R.S. Kanchan, K. Gopakumar, "A self-adaptive space phasor based hysteresis current controller with switching of adjacent inverter voltage vectors for three-level voltage source inverter fed induction motor drive," *Electric Machines and Drives, 2005 IEEE International Conference on*, vol., no., pp.1765-1772, 15-15 May 2005.

[114] K. Gopakumar, A. Dey, R. Ramchand, P. Rajeevan, K. Mathew, "A Space Vector based Hysteresis Current Controller for a General n-level Inverter Fed Drive with Nearly Constant Switching Frequency Control," *Industrial Electronics, IEEE Transactions on* , vol.PP, no.99, pp.1, 0, Mar. 2012.

[115] T. Ghennam, E.M. Berkouk, B. Francois, "Three-level Inverter Controlled by means of Vector Hysteresis Current Control. Application to Back to Back

Structure," *Industrial Electronics, 2007. ISIE 2007. IEEE International Symposium on*, vol., no., pp.998-1003, 4-7 June 2007.

[116] J.D. Barros, J.F. Silva, "Optimal Predictive Control of Three-Phase NPC Multilevel Converter for Power Quality Applications," *Industrial Electronics, IEEE Transactions on*, vol.55, no.10, pp.3670-3681, Oct. 2008.

[117] T. Ghennam, E.M. Berkouk, B. Francois, "A Novel Space-Vector Current Control Based on Circular Hysteresis Areas of a Three-Phase Neutral-Point-Clamped Inverter," *Industrial Electronics, IEEE Transactions on* , vol.57, no.8, pp.2669-2678, Aug. 2010.

[118] R. Ramchand, K. Sivakumar, A. Das, C. Patel, K. Gopakumar, "Improved switching frequency variation control of hysteresis controlled voltage source inverter-fed IM drives using current error space vector," *Power Electronics, IET* , vol.3, no.2, pp.219-231, March 2010.

[119] P.C. Loh, D.G. Holmes, "A new flux modulation technique for multilevel inverters," *Power Electronics and Drive Systems, 2001. Proceedings., 2001 4th IEEE International Conference on*, vol.1, no., pp. 396- 402 vol.1, 22-25 Oct. 2001.

[120] CPT-DA2810 Technical Manual, Revision 1.4. Creative Power Technology, 2009.

[121] CPT-Mini2810 Card, Technical Manual, Revision 1.4. Creative Power Technology, 2010.

[122] CS_IIB Integrated Inverter Technical Manual, Revision 1.2. Creative Power Technology, 2008.

[123] CS_GIIB Integrated Inverter Technical Manual, Revision 1.0. Creative Power Technology, 2008.

[124] W.Y Kong, "Decentralised Control of a Cascaded Modular Multilevel Converter", Ph.D. Thesis, Monash University, Australia, 2011.

[125] Davoodnezhad, R.; Holmes, D. G.; McGrath, B. P.; " A Three-Level Self-Synchronizing Hysteresis Current Regulator with Constant Switching Frequency " Energy Conversion Congress and Conference (ECCEAsia DownUnder 2013), 2013 5th International, vol.1, no., pp.38-44, 4-6 June. 2013