

Multi-objective Direct Quasi-Sliding Mode Control of a Two Phase Buck Converter

by

Christopher Wilson

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama

August 4, 2012

Keywords: multiphase buck converter, sliding mode control, variable structure systems,
nonlinear control systems, MIMO

Copyright 2012 by Christopher Wilson

Approved by

Robert N. Dean, Co-Chair, Associate Professor of Electrical and Computer Engineering
John Y. Hung, Co-Chair, Professor of Electrical and Computer Engineering
George T. Flowers, Professor of Mechanical Engineering

Abstract

Multiphase converters are the state of the art for high efficiency point-of-load conversion for modern processors and FPGAs with wide operating load conditions. However, there are a variety of control issues that must be addressed, such as dynamically adjusting the number of active phases, robustness to disturbances, and phase current regulation. To achieve the highest efficiency conversion, the phase currents should not necessarily be split evenly between phases. Unfortunately, most of the literature concerning multiphase buck converters implements equal current sharing between phases.

Sliding mode controllers are known for their robustness and insensitivity to disturbances and plant parameter variations. Sliding mode control is particularly suited for power electronics due to the inherent switching between two system structures. The control effort of the sliding mode controller depends solely on which side of the surface the system state currently resides. This effort can be implemented in two ways: direct and indirect.

This work designs, simulates, and implements a direct sliding mode controller for a two-phase buck converter. The technical approach extends a single-phase sliding mode controller to control a two-phase buck converter. Since there are two phases, two switching surfaces are necessary. Thus, suitable surfaces must be developed to address the challenges of phase delay in addition to the overall goal of output voltage regulation. This work is tested in simulation with good results and is implemented with some modifications on a hardware test platform consisting of a TI TMS320F28335 microcontroller and two buck converter phases. The proposed controller can be extended to support regulating to desired phase's currents and dynamically change the numbers of active phases based on load conditions. This work confirms that the sliding mode controller is a suitable controller design for multiphase buck

converters and is the basis for future work improving current sharing, implementing phase shedding, and expanding the maximum number of active phases.

Acknowledgments

I would like to first thank God for the breath that He gave me and the ability to accomplish His goals. I would never have gotten this far without the encouragement of my loving wife and very supportive family. From parents who taught me to think, to a sister who always challenges me, and to a wife who's love, support, and humor I could never do without, I love you all.

I would also like to extend my gratitude to my committee members, Prof. John Y. Hung, Dr. Robert N. Dean, and Dr. George T. Flowers. Without your support, I would never have expanded my views on control systems and their applications. I also would like to thank Dr. Kelli Watts, who provided valuable comments during the writing process. I would also like to thank Dr. Thaddeus Roppel, who's demonstrated faith and encouragement was always refreshing. I was always encouraged and motivated by all the conversations, lunches, and Ultimate Frisbee I had with the late Dr. A. Scottedward Hodel. I will always be in his debt as his knowledge, example, and legacy led me to pursue the doctorate and to be an example to those around me.

I would also like to thank my group mates on the power project for their valuable assistance in getting parts built and bearing with the stress of the writing process. From late nights with Stephan Henning working on boards to repeated requests to replace parts with Justin Moses to conferences with Luke Jenkins, your help has been invaluable. Finally I would like to thank my former office mate, sounding board, and friend for far too many years at Auburn, Dave Broderick: "So long and thanks for all the chicken."

Table of Contents

Abstract	ii
Acknowledgments	iv
List of Figures	vii
List of Tables	ix
1 Introduction	1
1.1 Prior Work	1
1.1.1 Buck Converters	1
1.1.2 Control Issues for Multiphase Buck Converters	3
1.1.3 Sliding Mode Control	4
1.2 Contributions	5
1.3 Organization	5
2 Background	7
2.1 Buck Converters	7
2.1.1 Architecture improvements	7
2.1.2 Modeling	10
2.1.3 Control Improvements	11
2.2 Sliding Mode Control	13
2.3 Summary	17
3 Sliding Mode Control of a Single-phase Buck Converter	19
4 Sliding Mode Control of a Multiphase Buck Converter	22
4.1 Model of a Multiphase Buck Converter	22
4.2 SMC of a Two-Phase SR Buck Converter	23
4.2.1 Output Voltage Ripple	24

4.2.2	Control Input Balance	25
4.2.3	Inductor Current Monitoring	26
4.2.4	Load Current Sharing	27
4.2.5	Complete Controller	28
5	Simulation Results	29
6	Experimental Setup	34
6.1	Implementation Hardware and Test Setup	34
6.2	Experimental Algorithm	38
7	Validations through Experiments	42
7.1	Voltage Responses	42
7.2	Phase Current Responses	44
7.3	Experimental Analysis	48
8	Conclusions	49
8.1	Future Work	50
	Bibliography	51
	Appendices	61
A	Source Code for Two Phase Controller:	62
B	Source Code for Single Phase Controller:	114

List of Figures

3.1	Single-phase buck converter.	19
4.1	Two-phase buck converter.	22
5.1	Simulated voltage response for load increase.	30
5.2	Simulated current response for load increase.	31
5.3	Simulated voltage response for load decrease.	32
5.4	Simulated current response for load decrease.	33
6.1	Single-phase buck test board.	35
6.2	Controller Card	36
6.3	Carrier board for testing.	37
6.4	Load board used for testing.	37
7.1	Output voltage response for load increase.	43
7.2	Output voltage response for load decrease.	43
7.3	Output voltage response for load decrease.	44
7.4	Phase current response for load increase.	45
7.5	Total phase current response for load increase.	45

7.6	Output current response for load increase.	46
7.7	Output current response for load decrease.	47
7.8	Phase current response for load decrease.	47
7.9	Total phase current response for load decrease.	48

List of Tables

5.1	Parameter values used in the simulations.	29
6.1	Parameter values for the hardware test setup.	34
6.2	Parameter values used in the hardware controller.	41

Chapter 1

Introduction

Multiphase buck converters are utilized for high efficiency over wide operating loads [1]. Additionally, the multiphase buck converter can possess a smaller output voltage ripple than single-phase buck converters. Both of these advantages are accomplished by interleaving the active phases with a suitable phase delay, properly sharing current between active phases, and dynamically adjusting the number of active phases (phase shedding). The goal of this work is to extend an existing sliding mode (SM) controller for a single-phase buck converter to include proper phase shifting for a two-phase buck converter while maintaining and improving transient responses and steady-state voltage ripple.

1.1 Prior Work

This section will present some relevant motivations and background material for multiphase buck converters and sliding mode controllers. It is broken up into three subsections. The first section will present motivations for using a multiphase buck converter and the typical point-of-load application requirements the multiphase buck converter is designed to solve. The second section will present some control issues associated with a multiphase buck converter. The third section will present some background material on sliding mode control theory.

1.1.1 Buck Converters

With the rapid ascension of the internet and cloud computing, large data centers have been built to satisfy the ever increasing demands for information and processing [2]. Due to the increased power density afforded by modern processors, the power density of data

centers has also been steadily increasing. Correspondingly, the energy required to power and cool data centers has been growing as well [3]. In many cases, the cost of energy for one year exceeds the price of the servers themselves. This high energy usage can be reduced by using more efficient power supplies and topologies, which will also have the added benefit of reducing cooling requirements [4].

While there are various power distribution architectures and converters, one of the most challenging converters from an efficiency standpoint is the point-of-load (POL) converter. This converter takes in an intermediate DC voltage, typically 12V, from upstream supplies and converts it to a high-current, low voltage suitable for modern processors [5]. From an energy efficiency standpoint, this converter is very challenging due to the high current outputs required by modern processors. In many cases, the output of the POL is 1V at 100A. Since power is proportional to current squared ($P \propto I^2$), small parasitic resistances in the converter can cause dramatic power losses at higher currents.

In addition, the microprocessors that the POL powers have very stringent voltage tolerances. In the case of the Intel Core i7 processor, the processor requires 1.5% tolerance, or at 1V, 15mV at 65A [6]. Some FPGA vendors use similar tolerances. The Altera Stratix V requires 3% steady-state tolerance at 0.85V and 130A [7]. These tight tolerances and wide load range lead to a multiphase buck converter typically being used for POL applications [8].

For some customers, the end-to-end energy efficiency of the power distribution system is the primary concern. This is driven by the continually increasing cost of energy. In some cases, the yearly cost of energy is more than the cost of the computer system. Since the efficiency of the converters is load dependent and peaks at a certain load conditions, multiphase buck converters are a good fit for energy efficiency. Subsequent phases can be turned on as the load increases to keep the currently operating phases at their peak operating efficiencies. As the load decreases, phase shedding can occur, turning off phases to maintain a high overall efficiency.

1.1.2 Control Issues for Multiphase Buck Converters

In order to achieve the maximum efficiency across a wide range of loads, the controller must regulate the output voltage while ensuring optimal distribution of the load currents. As Costabeber investigated, efficiency of the overall converter is dependent on the load and the number of operating phases [1]. Thus, the controller must be able to enable and disable phases depending on the load conditions.

However, in addition to adjusting the number of phases, the controller must regulate the current in the phases. Typically multiphase buck converter controllers simply enforce equal current distribution between the phases [9]. This has the advantage of reducing per-phase currents and therefore the corresponding I^2R losses. However, as Vilchis et al. point out, maximum efficiency operating conditions are highly load dependent and can be optimized by adjusting the currents in each phase independently [10].

The controller should also be robust to widely-varying load, line, and component conditions. As the Intel Core i7 specification states, the idle current for the processor is 2 A and full load is up to 65 A [6]. Many computers schedulers are scheduling the processor for maximum throughput and then idling to improve battery life in mobile applications [11]. Thus, these wide current transients will become standard to improve mobile battery life. Additionally, the controller should also be immune to line voltage changes. As servers continue to be consolidated into multi-processor systems, the end POL converter will be connected to other POL converters and the upstream power supply, either an ac-dc converter or another dc-dc converter [4]. Since each processor's power draw influences the equivalent impedance the POL presents to the upstream supply, the input voltage on each POL converter will vary due to processor transients. Thus, regardless of what the other processors are doing, each POL should maintain output voltage regulation. The controller should also be robust against component parameter variations due to aging [12]. The POL should output the same

voltage regardless of component age until component failures occur. Multiphase buck converters are also capable of tolerating phase failures should the switches fail open. Therefore, the controller should be able to adjust and continue operating.

1.1.3 Sliding Mode Control

SM controllers are a family of nonlinear controllers that are particularly suited for variable structure systems, such as switching power converters. SM controllers are designed to switch between multiple system structures. The resulting closed loop dynamics are often geometrically described in state-space as trajectories along a sliding “surface” [13].

There has been significant work in designing SM controllers for power switching systems. There are two methods to implement sliding mode in power converters: direct and indirect [14, 15]. Direct control utilizes the sliding surface’s sign to directly feed the switches. Indirect control, however, utilizes the sliding surface to vary the duty cycle in a traditional pulse-width modulation (PWM) scheme. Li and Ye utilize SM controllers for a single phase buck converter [16]. Kordkheili and Mehrasa develop a SM controller for a three-phase ac-dc boost converter [17]. Biel et al design two SM controllers, one for a boost converter and one for a buck converter [18]. The boost converter is then connected to the buck converter. A sinusoidal reference to the buck converter generates the desired AC signal.

There has been some work to implement a SM controller for multiphase buck converters. Lee et al. utilizes a cascaded approach where the control effort of one phase is added to the surface of the next phase and integrated to obtain the second phase’s control effort [19]. However, they only consider fixed numbers of active phases for the converter. Additionally, there are a variety of scenarios where a designer might not want to have similar phase-shifted responses, such as in the cascaded approach. For the mutliphase buck converter to maintain the highest efficiency over a wide load range, it may be better to run most of the phases at a peak efficiency operating point and only one or two phases at lower efficiency operating points, instead of running all at identical operating points at an overall lower efficiency [1].

This also has the advantage of shutting down unused phases, eliminating their corresponding switching and conduction losses.

Therefore, a similar but unique surface for each phase of a multiphase buck converter is desired. A similar surface is desired such that all phases will respond to any load transients. However, the surface should be unique so that the steady state voltage ripple is reduced and each of the phase currents is controlled to attain optimal converter efficiency. In the development of these surfaces, it is beneficial to examine the single-phase buck converter model and corresponding SMC, examine the multiphase buck converter model, and expand the single-phase SMC with the necessary additional terms to control the multiphase converter.

1.2 Contributions

This work develops a design methodology for sliding mode control of a two phase buck converter. In contrast to prior works in the area, the new method enables design of different control strategies for each of the two phases. The design methodology enables the application of the two-phase converter to other control objectives, beyond mere voltage regulation. For example, the added objective of efficiency control can be addressed. To demonstrate the efficacy of the contribution, a design to meet the objectives of voltage regulation and proper time delay between phases is presented. The author also shows how the controller is implemented on a Texas Instruments (TI) C2000x family processor. The design methodology also forms a basis for extensions to additional phases.

1.3 Organization

Up until now, only highlights of the engineering problems, control issues, and start-of-the-art solutions have been presented. In the remaining chapters, the author gives in-depth descriptions about buck converters, sliding mode control theory, application of sliding mode control theory to buck converters, extension of the single-phase sliding mode controller to

two-phase, and implementation of the two phase sliding mode controller. Specifically, Chapter 2 presents an in-depth review of the literature on single and multiphase buck converters and sliding mode control theory. Chapter 3 presents the highlights of the single phase converter and its sliding mode controller which forms the basis of the multiphase controller presented in Chapter 4. Readers who are familiar with the general field may wish to skip to Chapter 4 where the new design methodology is detailed. Simulation results of the proposed controller are presented in Chapter 5. Experimental evaluations are presented in Chapter 6 and 7. Contributions are summarized in Chapter 8, with an eye to future directions.

Chapter 2

Background

2.1 Buck Converters

Buck converters are conceptually fairly simple power converters. However, given their applicability to step-down voltage and allow for high currents, much work is still being performed on them to improve their efficiency and transient responses through various additions and techniques. These techniques can generally be thought of as improvements to the buck architecture, the converter model, and the control system.

2.1.1 Architecture improvements

The architectural improvements to the buck converter mainly focus on various designs which should espouse improved efficiency, controller performance, or volume reductions.

Some work has been done on the system level which involves separating a monolithic centralized controller and putting controllers on each phase [20]. Huang et al. then link the individual per-phase controllers with an overall controller which manages phase syncing and multiple phase transient responses. Huang et al. claim that the controller bandwidth is improved due to reduced phase delay from the phase output voltage to the controller.

Other works take the traditional multiphase concept and tweak it slightly to enable better transient responses. Wen et al. propose using an auxiliary supply to provide supplemental current support during load transients, essentially creating a second phase with a small inductor and high switching frequency which is driven from the gate supply voltage [21]. Similarly, Du et al. investigate using two buck converters with different switching frequencies [22]. The slow phase is designed to support most of the load and is optimized

for steady-state performance. The fast phase is designed to support load transients and therefore possesses a smaller inductance and faster switching speeds.

Other authors tend to augment a multiphase buck converter with additional converter types to improve efficiency or density. MahdaviKhah et al. focus specifically on increasing the power density of the converter and add an additional capacitive divider stage before the two-phase interleaved buck converter to achieve the increased power density [23]. The capacitive divider allows a lower input voltage which allows for an increased duty cycle of the buck converter. Panguloori et al. provide an interesting approach by using an isolated full bridge current doubling rectifier in parallel with the standard buck converter [24]. The buck converter operates to supply load transients and improve the transient voltage response while the full bridge current doubler tries to meet the new load current demands. Upon the current doubler matching the load current demand, the buck converter turns back off. This approach is similar to the one provided by Du and Wen.

Still other works propose adding additional components and changing the overall architecture of the individual buck converters. Abe et al. develop a cascaded buck converter where all but the last cascaded converter introduces a capacitor between the high-side switch and the inductor node [25]. Additionally, the next phase is fed from the node between the high-side switch of the previous phase and the added capacitor. The goal of this technique is to reduce the effective voltage across any switch to at most $2/3 V_{in}$. However, one disadvantage of this technique is that now the full converter input current passes through the first phase's high-side switch. Additionally, if that switch fails open, the entire converter fails. Thus, such a technique succeeds in its initial goal, but is only as reliable as the first phase's high-side switch.

Similarly, Lee et al. suggest a new architecture for two-phase buck converters keeping the same number of active switches but introducing an additional series capacitor into one of the phases [26]. This reduces the voltage stress on one of the switches to $1/2 V_{in}$. Additionally, the switching voltage on the switching MOSFETs is reduced, thereby reducing the switching

losses. The author's full loss calculations show greatly improved efficiencies in the new design; however, the authors neglect to include the additional losses in the added capacitor in these calculations. Regardless, the measured efficiency numbers do show improvement at light load conditions.

Garcia et al. utilize work done for zero-voltage switching (ZVS) for buck converters to utilize current balance between the multiple phases [27]. Their technique, however, requires ripple currents that are higher than twice the average current in each phase's inductor, increasing RMS losses. To overcome those shortcomings, additional output capacitance is required. Additionally, Garcia utilizes the negative current flowing in the inductor to develop ZVS in the high side switch. The negative current also requires extra energy from the other phases, reducing the overall converter's efficiency.

Rahimi et al propose adding an additional 3 diodes, 2 capacitors, and an inductor on the inductor node of a synchronous buck converter [28]. This allows the main switches to experience a resonant condition and allows for zero current turn-on and zero voltage turn-off of the main and synchronous switches.

Hwu and Chen develop a modified buck converter utilizing a differential mode transformer in between the phase inductors and the switches [29]. The advantage of their approach is that current sharing between inductors is automatically enforced by the transformers. Interestingly, the authors do not discuss the desired phase delays between phases to reduce the output voltage ripple.

Jang et al examine the issue of small-duty cycles on multiphase converter efficiencies and develop an architecture utilizing two additional MOSFETs and capacitors to provide reduced switching voltages and to increase the effective duty cycle [30].

Xu et al look at the same issue and propose a tapped coupled inductor approach where the number of turns in the inductor are determined by the desired transient response [31]. They shown higher efficiency numbers due to the addition of a capacitor which provides reduced switching voltages across the main switching MOSFETs.

Indeed, many authors analyze the coupled inductors approach to improving buck converter performance, size, and efficiency. Xu et al provide evidence that for some situations, coupled inductors provide better energy efficiency with similar transient responses [32]. Nagaraja et al work to further refine the design techniques for utilizing coupled inductors in multiphase buck converters [33]. Wu et al develop averaged models for the coupled inductor design and illustrate the design ideas in a prototype converter [34]. Wibowo et al also analyze coupled inductor buck converters and show a roughly 10% reduction in voltage transients due to load changes over non-coupled inductors [35].

Garinto and later his collaborator, Taufik, discuss the advantages of multi-interleaved buck converters which can reduce the output filter requirements and the overall volume, but increases the number of inductors [36, 37, 38] per phase. Additionally, since multiple inductors are now carrying the full output current load, the I^2R losses are increased as well. Taufik et al eventually reduce the number of necessary inductors but still require $n+2$, where n is the number of phases [39].

2.1.2 Modeling

Various improvements have been made in the literature concerning the modeling of the buck converter and the various converter improvements.

Qiu et al utilize harmonic balance to generate nonlinear models of single and multiphase buck converters which show the sidebands generated by the square-wave pulsing, which are not shown in linear models [40]. Kondrath and Kazimierczuk develop an analytical model for wide-frequency and high-frequency small-signal models for the buck converter [41]. Their work shows that at low frequencies, the converter appears as a second order system. However, for high frequencies, the inductor and parasitic inductance of the capacitor dominate, making the system appear like a first order inductor. Similarly, Qiu introduces multifrequency small-signal models for multiphase buck converters to show control-loop bandwidth limitations [42].

Eirea and Sanders develop a method for estimating the phase current unbalance between the various phases [43]. Their method depends on similar high-side switch resistances between phases and makes use of the input voltage drop when the phases turn on. With unbalanced phases, the input voltage droops further with the higher current phase than with the lower current phase. This droop imbalance then allows estimation of the imbalance between phase currents.

Bouhalli et al discuss magnetic coupling issues to buck converters and present a model used for their analysis of various coupling configurations [44]. Their work shows that some configurations of inductor coupling are better suited to higher phase counts.

Shao et al look at the use of identifying return current paths to determine optimal thermal placement of inductors and capacitors [45]. Their work reduces the generation of electromagnetic interference at high frequencies by reducing the parasitic inductance of the high current loops.

2.1.3 Control Improvements

In addition to the architecture additions and new modeling techniques, buck converters are also experiencing an increase in the number of proposed controller architectures. Such controllers can be categorized based on controllers which adjust phase currents and phase shedding, adaptive voltage positioning, hysteretic controllers, and hybrid controllers.

There are a variety of control approaches to performing phase shedding and phase current balance. Costabeber et al. look at time-optimal phase shedding for multiphase buck converters which only utilizes a feed-forward approach for control [1]. Unfortunately, the system only looks at time-optimality for changing the phase currents. The output voltage during the phase shedding transients isn't considered. Additionally, Costabeber examines the efficiency of the overall converter based on load and the number of active phases. Pascual et al look at improving light load efficiency by shedding phases and propose a controller to

adjust a linear controller to enable the phase-shedding based on load [46]. The controller is verified in simulations.

In addition to phase shedding, phase current balance is an often stated goal of multiphase controllers in order to improve efficiency by reducing per-phase currents and improve reliability by not loading one particular phase with more losses over time. Retegui look at the multiphase buck converter as a high-power current source and provide control with minimized current ripple by anticipating the average crossings of the phase currents [9]. Chiu et al develop a modified high-gain peak current controller to combat phase current oscillations that are present in typical high gain peak current control schemes for multiphase buck converters [47].

Other control approaches ignore phase shedding and focus solely on adaptive voltage positioning (AVP) for modern computer processors with both nonlinear [48] and linear [49] controllers. Another use for adaptive voltage control includes envelope modulation for RF amplifiers [50].

Other authors propose using analog hysteretic controllers which can possess AVP and modular design [51], nonlinear and constant frequency [52], and predictive [53] control.

Vilchis et al. propose non-uniform current sharing between phases as well as a method of load dependant operating modes as a method to increase efficiency in battery powered applications [54, 10].

Some interesting work has also been done merging linear and nonlinear controllers for various modes of operation. Boscaino et al. utilize a primary linear controller during steady-state operations with a nonlinear controller to modify those parameters during transient events [55]. Barroado et al follow a similar approach, but utilize only the nonlinear controller outside of certain threshold levels of the output voltage [56]. Du et al., however, propose using linear control loops, but keep competing loops from saturating during various loop competitions [57].

Interestingly, Zaman et al seek to control the thermal issues for multiphase current mode buck converters and present a peak-current digital controller to reduce peak temperatures in the switching MOSFETs [58].

2.2 Sliding Mode Control

Since being developed by Russian scientists and engineers in the 1950's and 1960's, sliding mode control has been steadily developed for various applications.

Several authors have looked at discrete sliding mode control. Orosco and Vazquez present the necessary reaching and existence conditions for discrete sliding mode controllers [59]. Janardhanan et al present a method of discrete output feedback for quasi-sliding mode control with numerical simulations for uncertain systems [60]. Bandal and Vernekar develop a discrete-time sliding mode controller utilizing multirate output feedback for a magnetic levitation system [61]. They also develop a new formulation on discrete-time sliding mode control which guarantees a discrete system response similar to a continuous time sliding mode controller. Matas et al present a discrete sliding mode controller with output tracking for a boost converter [62].

Hybrid control techniques have also been developed which utilize linear and nonlinear controllers together, or utilize structures similar to linear controllers. Li et al develop a hybrid PD-sliding mode controller which uses the PD controller for transient performance and the sliding mode controller for steady-state disturbance rejection [63]. Tan et al develop a fixed-frequency PWM-based quasi-sliding mode controller which exhibits similar structure to a PD linear controller with additional terms for input and output voltage [64]. Further, they develop an error voltage based sliding mode controller with a form very similar to a PID [65] and then further refine the idea and offer general design considerations for such a controller [66]. Li and Ye also develop a sliding mode controller based on the PID approach [16].

However, Sira-Ramirez shows equivalence between sliding regimes of variable-structure feedback control and PWM responses in nonlinear dynamical systems [67]. Thus the hybrid techniques for nonlinear plants, such as dc-dc converters, actually just utilize two different sliding surfaces: one explicit and one implicit. Similarly, Iqbal et al propose a combined sliding mode controller utilizing both first order sliding modes and dynamic sliding modes [68]. Ackermann and Utkin develop an explicit form for the discontinuity plane for the SM controllers and then show a two controller approach, where static controllers force the system into sliding mode and dynamic controllers enforce the desired dynamic properties [69].

One issue often seen with sliding mode controllers is the requirement for infinite switching speed to slide across the sliding surface [13]. However, since real circuits cannot switch infinitely fast, a phenomenon known as chattering occurs. Therefore to minimize the chattering effect, Mohanty uses a sliding mode controller with a boundary layer when controlling an induction motor drive [70]. The boundary layer provides a hysteresis zone around the sliding mode where the controller will not switch.

Adjusting the hysteresis zone, or boundary layer width, is also a method for controlling the frequency of the converters, as discussed by Cardoso et al [71]. Nguyen and Lee propose using an adaptive hysteresis band to control the switching speed to adjust for variations in components [72]. They then discuss some practical methods for implementing indirect sliding mode control utilizing fixed-frequency PWM methods [73].

However, many other authors utilize adaptive sliding modes to dynamically adjust the sliding surfaces. Fei presents an adaptive sliding mode tracking controller based on a PI switching surface using an integral sliding term and an adaptive sliding mode controller [74]. Yu et al develop a new sliding mode controller design, termed adaptive seeking sliding mode control, to adaptively estimate and compensate the chattering effect due to continuous and slowly varying disturbances [75]. Comanescu and Xu present two robust sliding mode model reference adaptive system observers for speed estimation in a sensorless vector controlled induction machine drive [76]. Wai uses an adaptive sliding mode controller to control the

position of an induction servomotor drive [77]. The interesting thing about Wai's approach is that the surface proposed has no reaching phase; the entire trajectory is in sliding motion within disturbance bounds. Additionally, Wai proposes an adaptive approach which relaxes the requirements on the disturbance bounds to stay in the sliding mode.

In addition to adaptive work, sliding mode control has been used as the basis for fuzzy and neural-based controllers. Kung and Lai utilize sliding mode control techniques to design fuzzy control rules [78]. Next, region-wise linear fuzzy controller is designed based on the fuzzy sliding mode controller which is then tuned to find the optimal linear fuzzy parameters with a genetic algorithm. Betin et al use sliding mode theory to define optimal scaling values for a fuzzy logic controller [79]. Zhang et al utilize a second order time-varying system to model higher order time-invariant systems and propose a fuzzy sliding mode control scheme for the reduced order time-variant system [80]. Perry et al create a fuzzy sliding mode-like controller for buck converters which ensures zero steady-state error [15]. Raviraj and Sen compare PI, SM, and fuzzy controllers and outline the mismatch between characteristics which lead to varying performance between the controllers [81].

In addition to the fuzzy sliding mode controllers, sliding mode has also been applied to neural networks. Xiaojiang et al utilize a neural sliding mode controller to perform trajectory tracking for multi-link robots with uncertainty in model parameters and disturbances [82]. Interestingly, Topalov et al use a sliding mode controller to perform online training of a neural network controller without requiring a priori knowledge of plant dynamics [83].

With multi-input sliding mode controllers, it becomes necessary to define multiple surfaces. This is due to each system input requiring a control output, and consequently, the defined surface upon which to switch. Qian et al develop an hierarchical SM controller where two sliding surfaces define the system controller [84]. First, the system is designed to slide along one surface and then slides along the second. Yu et al demonstrate a cascaded sliding mode controller applied to a nonlinear bicycle robot [85]. The system is divided into two

different subsystems and each sub-sliding surface is designed. The ultimate control system slides along each sub-sliding surface controlling the robot effectively in simulation.

Sliding mode control also has an extensive history with power converters. Mahdavi et al analyze small and large signal disturbances around the operating points for DC-DC converters controlled by sliding mode controllers [86]. Then they utilize state space averaging and a neural network to implement a sliding mode controller on a Cuk converter [87]. Fossas et al use SMC to control the output voltage of a Cuk converter and transform the constant sliding mode surface into a time-varying surface once sliding mode is obtained [88].

In addition to the Cuk converter, SMC has been applied to the boost converters. Giral et al look at the use of SMC for boost converters for voltage doubling and quadrupling [89]. Lopez et al utilize sliding mode control based on the equivalent control method to parallel multiple boost converters and implement current sharing between the various phases [90]. Shtessel et al study the nonminimum phase tracking control for boost and buck-boost power converters under SMC [91].

Single phase buck converters have also been a popular topic for investigation of sliding mode techniques. Ahmed et al focus on the robustness of the sliding mode controller for a buck converter for three areas of operation: start up, line/load variation, and component variation. Their work shows the SMC has good immunity to component variations [92] and load and line disturbances [14]. Vazquez et al propose a technique for voltage-only SMC for buck converters [93]. Sira-Ramirez relates SMC of a buck converter to classical Δ modulators and $\Delta - \Sigma$ modulators [94]. Alarcon et al design and implement a CMOS ASIC which implements versatile sliding mode control laws for DC-DC converters [95]. Fossas and Ras discuss a second-order sliding mode controller which greatly reduces the chattering effect in a buck converter [96].

Extension of the single phase buck converter to the multiphase converter has also been a topic of particular interest. Mazumder et al propose a SMC for parallel buck converters using integral-variable structure and multiple sliding surface control and examine the existence

criteria and stability of the sliding manifolds for the paralleled converters [97]. Mazumder and Kamisetty provide an analog implementation of a sliding mode controller for multiphase buck converters [98]. Their experimental results show good performance for the multiphase converter at step changes but show no ability to implement phase shedding. Additionally, they utilize the indirect method of control via a PWM. Utkin designs a SMC which dictates the number of phases necessary for the desired amplitude of chattering in a multiphase buck converter [99]. Unfortunately, his approach does not provide for dynamically variable numbers of phases or non-balanced current between phases. Biel et al utilize Lee and Utkin's approach in [19] and adjust Lee's sliding parameter K to achieve voltage control and minimize chattering [100].

Shtessel et al study the system of modular power converters and how multiple systems which share the same input bus can effect each other if the controller is not specifically designed to account for these changes [101]. They present a SMC for each of the end-stage modules to include the effect of voltage input droop due to increased current demands from other paralleled stages. Lopez et al create a sliding surface which can successfully interleave phases and share current equally [102]. However, their approach requires measuring the currents for use in the surfaces. Additionally, it does not allow independent control of phase currents.

In addition to controllers, sliding mode can also be used for observers. Lienhardt et al use sliding mode observers to determine the capacitor voltages in stacked multicell converters [103]. Zhou et al discuss sliding mode observer-based sliding mode control for nonlinear systems with distributed delays [104].

2.3 Summary

However, in all the literature searched, there has not been a digital sliding mode controller which has been specifically designed to support various independent current levels

which will enable optimization of the efficiency of the multiphase buck converter across various load conditions. Mazumder and Kamisetty provide a SMC for multiphase converters, but it does not support load shedding or phase-current setting [98]. Vilchis et al utilize various non-uniform buck converter phases to support high efficiencies at various load conditions, but never actually experimentally implement the work [54, 10]. Thus, the goal of this work, a discrete sliding mode controller which can serve as the foundation for independent phase current setting and supports phase shedding, has not been addressed in prior works.

Chapter 3

Sliding Mode Control of a Single-phase Buck Converter

The single-phase controller is based on the controller by Tan et al [66]. To help define the notation used later in the paper, the key elements of Tan's work are repeated here. Familiar readers should skip to Section 4.1. A single-phase buck converter is shown in Fig. 3.1. The buck converter consists of an output capacitor, C , an inductor, L_1 , and two switches: the main switch, SW1, and a synchronous rectifier, SR1. Unlike the traditional buck converter, the synchronous buck converter replaces the diodes with controlled synchronous rectifier (SR) switches and is able to achieve a higher efficiency [105].

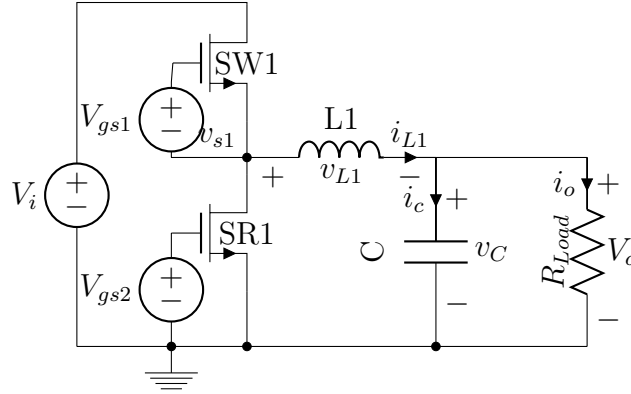


Figure 3.1: Single-phase buck converter.

The states of this converter are defined as:

$$x_1 = V_{ref} - V_o \tag{3.1}$$

$$\begin{aligned} x_2 = \dot{x}_1 &= -\frac{dV_o}{dt} \\ &= -\frac{1}{C} \left(\int \left(\frac{V_i u_1 - V_o}{L_1} \right) dt - \frac{V_o}{R_{Load}} \right) \end{aligned} \tag{3.2}$$

where u_1 is the control input and has the values of $[0, 1]$. This is a voltage formulation of the states, where the integral term in x_2 represents the inductor current.

The single-phase buck converter controller is derived from [65], and will be summarized here. Augmenting the two states given by Eqns (3.1) and (3.2) with:

$$x_3 = \int x_1 dt \quad (3.3)$$

The sliding mode surface can be considered as:

$$\begin{aligned} S &= \alpha \cdot x \\ &= \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 \end{aligned} \quad (3.4)$$

The control input is formulated to be:

$$u_1 = \begin{cases} 0 & \text{for } S < 0 \\ 1 & \text{for } S > 0 \end{cases} \quad (3.5)$$

The existence of the sliding mode can be determined by examining the stability of the surface. State trajectories will move to the switching surface if they satisfy the so called “reaching condition” given by:

$$S \cdot \dot{S} < 0 \quad (3.6)$$

For the case $S > 0$:

$$\dot{S} < 0 \quad (3.7)$$

$$\begin{aligned} \dot{S} = & \left(\frac{\alpha_1}{\alpha_2} - \frac{1}{CR_{Load}} \right) x_2 + \frac{V_{ref}}{CL_1} \\ & + \left(\frac{\alpha_3}{\alpha_2} - \frac{1}{CL_1} \right) x_1 - \frac{V_i}{CL_1} < 0 \end{aligned} \quad (3.8)$$

Similarly, for $S < 0$:

$$\dot{S} > 0 \quad (3.9)$$

$$\dot{S} = \left(\frac{\alpha_1}{\alpha_2} - \frac{1}{CR_{Load}} \right) x_2 + \frac{V_{ref}}{CL_1} + \left(\frac{\alpha_3}{\alpha_2} - \frac{1}{CL} \right) x_1 > 0 \quad (3.10)$$

For the sliding mode to exist, the coefficients, α , must satisfy:

$$0 < \left(\frac{\alpha_1}{\alpha_2} - \frac{1}{CR_{Load}} \right) x_2 + \frac{V_{ref}}{CL_1} + \left(\frac{\alpha_3}{\alpha_2} - \frac{1}{CL_1} \right) x_1 < \frac{V_i}{CL_1} \quad (3.11)$$

Now that the single-phase case has been presented, the model of the multiphase buck converter will be presented.

Chapter 4

Sliding Mode Control of a Multiphase Buck Converter

In order to present the proposed multiphase sliding mode controller, the model of the two phase buck converter will be derived. Next, the proposed two phase sliding mode controller will be developed. Chapter 5 will present the simulation results of the proposed controller. Chapter 6 will present the hardware test bench. Chapter 7 will present the experimental results of the controller.

4.1 Model of a Multiphase Buck Converter

A multiphase buck converter consists of parallel buck converters sharing a single output capacitor. An example two-phase synchronous buck converter is shown in Fig. 4.1. The converter has two control inputs, u_1 and u_2 , which are represented by switch pairs (V_{gs1}, V_{gs2}) and (V_{gs3}, V_{gs4}) , respectively. When u_1 is on, V_{gs1} is high and V_{gs2} is off.

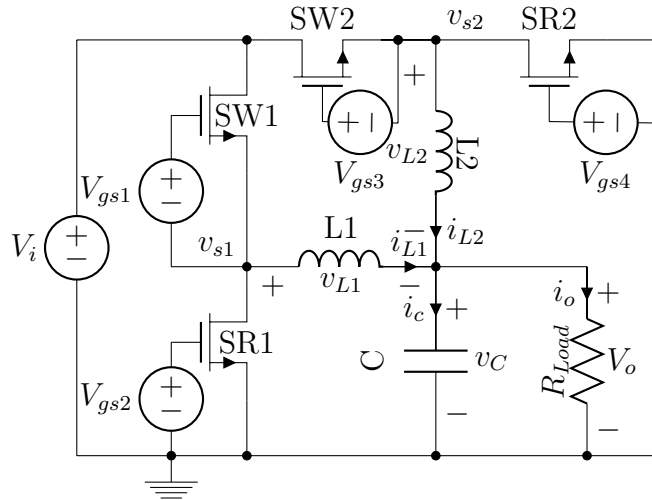


Figure 4.1: Two-phase buck converter.

The state variables for the multiphase converter are defined slightly differently than the single-phase case:

$$x_1 = V_{ref} - V_o \quad (4.1)$$

$$\begin{aligned} x_2 = \dot{x}_1 &= -\frac{dV_o}{dt} \\ &= -\frac{1}{C} \left(\int \frac{V_i u_1 - V_o}{L_1} dt + \int \frac{V_i u_2 - V_o}{L_2} dt - \frac{V_o}{R_{Load}} \right) \end{aligned} \quad (4.2)$$

$$x_3 = \int x_1 dt \quad (4.3)$$

where u_1 and u_2 are the control inputs for phase 1 and 2, respectively, and have values of $[0, 1]$. Notice the addition of the second integral term in Eqn (4.2) to account for the second phase.

Similarly, extension of the two-phase case to higher phase counts is easily seen by including additional integral terms for each phase. For example, a n phase converter will have the same x_1 state, with x_2 being defined as:

$$\begin{aligned} x_2 = -\frac{1}{C} &\left(\int \frac{V_i u_1 - V_o}{L_1} dt + \int \frac{V_i u_2 - V_o}{L_2} dt + \dots \right. \\ &\left. + \int \frac{V_i u_{n-1} - V_o}{L_{n-1}} dt + \int \frac{V_i u_n - V_o}{L_n} dt - \frac{V_o}{R_{Load}} \right) \end{aligned} \quad (4.4)$$

Now that the model of a two-phase SR buck converter has been developed, the two-phase SM controller will be developed.

4.2 SMC of a Two-Phase SR Buck Converter

Extension of the single-phase SMC to the multiphase case poses some additional challenges in exchange for some increased performance opportunities. The additional input source u_2 adds an extra degree of freedom that must be accounted for in an additional switching surface. However, this extra input can allow improved transient response and reduce output voltage ripple. The goal of this controller design is to take the single-phase

controller and adapt it as necessary for multiple phases. The equivalent multiphase sliding surfaces will begin expansion as:

$$S_1 = \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 \quad (4.5)$$

$$S_2 = \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 \quad (4.6)$$

where S_1, S_2 are the sliding surfaces for phase 1 and 2, respectively. The α gains are developed according to the design rules in [66]. The design process will be described below and motivations for adding additional terms to Eqns (4.5) and (4.6) will be given.

4.2.1 Output Voltage Ripple

In order to maintain minimum voltage ripple, the phase delay of phase 2 should be 180° [106]. However, there is currently nothing in Eqns (4.14) and (4.15) to enforce this. Therefore, an additional term is added to each surface to inject a resettable ramp in the sliding surface:

$$S_1 = \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 + \alpha_4 x_4 \quad (4.7)$$

$$S_2 = \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 + \alpha_4 x_5 \quad (4.8)$$

where

$$x_4 = \begin{cases} \int 1 \, dt & \text{for } u_1 = 0 \\ 0 & \text{for } u_1 = 1 \end{cases} \quad (4.9)$$

$$x_5 = \begin{cases} \int 1 \, dt & \text{for } u_2 = 0 \\ 0 & \text{for } u_2 = 1 \end{cases} \quad (4.10)$$

The ramps x_4 and x_5 are reset when $S_1 > 0$ and $S_2 > 0$, respectively. When $S_1 < 0$, then x_4 will start incrementing until S_1 becomes greater than 0. Similarly, x_5 will start incrementing when $S_2 < 0$ and will reset when S_2 becomes greater than 0. Unlike indirect SMC methods [107, 108], this ramp is not for frequency control. Instead, it is designed to ensure proper phase delay between the phases to minimize steady state voltage ripple.

The gain term, α_4 is a positive value shared between the phases which serves to adjust the impact of the ramp, relative to the voltage dynamics. Since the ramp is reset whenever the input for that phase goes high, the other phase will possess a higher value for the sliding surface during the next switching interval. This enables alternation between the phases as desired. The value for α_4 should be chosen such that $\alpha_4 x_4$ and $\alpha_4 x_5$ are slightly smaller than the steady-state dynamics right before the corresponding input turns on and resets the integrators. Too large of an α_4 results in oscillations and poor voltage regulation. Too low of an α_4 results in phase current oscillations between the phases and increased ripple.

4.2.2 Control Input Balance

Since the only time both inputs should be on is during a large voltage transient, each surface also includes a term to subtract the input of the other surface. This ensures both phases do not turn on at the same time under steady-state operating conditions. Additionally, this serves as the initial impulse to enable the previously discussed ramps to perform proper phase delay control. If this term is neglected, then both inputs can switch on at the same time during steady-state operation, resulting in a larger ripple than the single-phase case. However, both phases can still turn on during voltage transient effects as the surfaces in (4.5) and (4.6) will dominate the effect on the input term. Phrased another way, during initial transient effects:

$$\alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 \gg \text{sign}(u_n) \quad (4.11)$$

The newly modified surfaces are given by:

$$S_1 = \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 + \alpha_4 x_4 - \text{sign}(u_2) \quad (4.12)$$

$$S_2 = \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 + \alpha_4 x_5 - \text{sign}(u_1) \quad (4.13)$$

4.2.3 Inductor Current Monitoring

Since only the voltage dynamics are monitored in the original single-phase surface, there are no additional constraints on the phase currents. In buck converters with synchronous rectification (SR), such as the case in Fig 4.1, care must be taken to ensure the inductor current does not become negative. In converters without SR, the SR switches (SR1 and SR2) are replaced with diodes, which will prevent negative inductor current.

One possible response of the SR multiphase converter in (4.5) and (4.6) is for one phase to dramatically increase its current to positive infinity while the other phase performs the opposite response and decreases its current to negative infinity. So long as the net difference in phase currents keeps the voltage dynamics requirements, this is allowable. However, of course, no practical designer will desire this action as the unchecked current will cause the plant to fail. Thus, an additional term is added to each surface and is designed to constrain the phase currents from becoming negative:

$$S_1 = \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 + \alpha_4 x_4 - \text{sign}(u_2) + \alpha_{Neg} (1 - \text{sign}(i_{L2})) \quad (4.14)$$

$$S_2 = \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 + \alpha_4 x_5 - \text{sign}(u_1) + \alpha_{Neg1} (1 - \text{sign}(i_{L1})) \quad (4.15)$$

Consequently, α_{Neg} and α_{Neg1} are very large negative gains used to ensure that phase turns off when the opposite phase's current becomes negative. Thus, the other phase is forced to maintain the desired voltage dynamics and increase its current. If too small of an α_{Neg}

value is selected, then the phase currents may oscillate around zero, resulting in drastically decreased efficiency of the controller.

4.2.4 Load Current Sharing

One observed effect of the controller equations given in (4.7) and (4.8) is an alternation of the load current between the phases. For example, one phase will provide nearly the full output load current while the other phase hovers above zero. This is due to one phase staying on longer than another phase during transient events.

In order to facilitate current sharing, an additional term is added to each surface. This term is normally on while $\dot{S}_i$ is positive. However, when $\dot{S}_i$ is negative, this term becomes smaller. The effect of this term adjusts current sharing during transients. The terms x_6 and x_7 with their corresponding gains α_6, α_7 are introduced to S_1 and S_2 , respectively. These terms form a positive offset to the surface when $\dot{S}$ is positive, and a decreasing resettable ramp when the $\dot{S}$ is negative.

$$\begin{aligned} S_1 = & \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 + \alpha_4 x_4 - \text{sign}(u_2) \\ & + \alpha_{Neg} (1 - \text{sign}(i_{L2})) + \alpha_6 x_6 \end{aligned} \quad (4.16)$$

$$\begin{aligned} S_2 = & \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 + \alpha_4 x_5 - \text{sign}(u_1) \\ & + \alpha_{Neg1} (1 - \text{sign}(i_{L1})) + \alpha_7 x_7 \end{aligned} \quad (4.17)$$

where

$$x_6 = \begin{cases} \tau_{D1} - \int 1 \, dt & \text{for } \text{sign}(\dot{S}_1) < 0 \\ \tau_{D1} & \text{for } \text{sign}(\dot{S}_1) > 0 \end{cases} \quad (4.18)$$

$$x_7 = \begin{cases} \tau_{D2} - \int 1 \, dt & \text{for } \text{sign}(\dot{S}_2) < 0 \\ \tau_{D2} & \text{for } \text{sign}(\dot{S}_2) > 0 \end{cases} \quad (4.19)$$

The gain terms α_6 and α_7 should also be chosen such that $\alpha_6 x_6$ and $\alpha_7 x_7$ are small compared to the steady-state dynamics. Typical values chosen for this application are:

$$\alpha_6 = \frac{1}{2\tau_{D1}} \quad (4.20)$$

$$\alpha_7 = \frac{1}{2\tau_{D2}} \quad (4.21)$$

where τ_{D1} and τ_{D2} are the desired phase offsets. Too low or high of α_6 or α_7 results in unbalanced load sharing.

4.2.5 Complete Controller

The complete sliding surfaces are given in (4.16) and (4.17). The controller outputs are given by:

$$u_i = \begin{cases} 0 & \text{for } S_i < 0 \\ 1 & \text{for } S_i > 0 \end{cases} \quad (4.22)$$

Since the SM controller for the two-phase buck converter has been developed, a comparison is made to the single-phase converter to ensure similar or improved voltage dynamic performance while improving the transient response and decreasing steady-state voltage ripple.

Chapter 5

Simulation Results

Simulations were performed for a single-phase and two-phase buck converters. Hysteresis modulation was performed as described by [66] to maintain a constant switching frequency of 250kHz. The parameters used in the simulation can be found in Table ?? . For both the simulations, the same α_1 , α_2 , and α_3 were used to maintain the same error voltage dynamics. The values from Table 5.1 come from a prototype converter which will be utilized for future experimental validation.

Parameter	Value
C	121.1 μ F
L_1	1 μ H
L_2	1 μ H
V_i	12V
V_{ref}	1V
R_{Load}	0.1 Ω
α_1	10
α_2	968.8
α_3	582,892
α_4	0.002
α_6	1.25x10 ⁻³
α_7	2.5x10 ⁻³
α_{Neg}	-1x10 ⁶
α_{Neg1}	-1x10 ⁶
τ_{D1}	400
τ_{D2}	200

Table 5.1: Parameter values used in the simulations.

A 100% load increase was performed from 0.1 Ω to 0.05 Ω , or 10 A to 20 A. The voltage response is shown in Fig. 5.1. The current response is shown in Fig. 5.2. In the two-phase buck converter current response figures, phase A represents the first phase and phase B

represents the second phase. For the single-phase buck, the output voltage drops 65 mV and recovers in 60 μ s. For the two-phase buck, the voltage drops 45 mV and settles in the 60 μ s. However, the two-phase buck is within the steady-state output voltage ripple of the single-phase buck in only 50 μ s. Interestingly for the two-phase buck during the transient effect, the phase shift varies slightly until the error dynamics have started to settle. At which point the phase shift resumes its correct 180° shift.

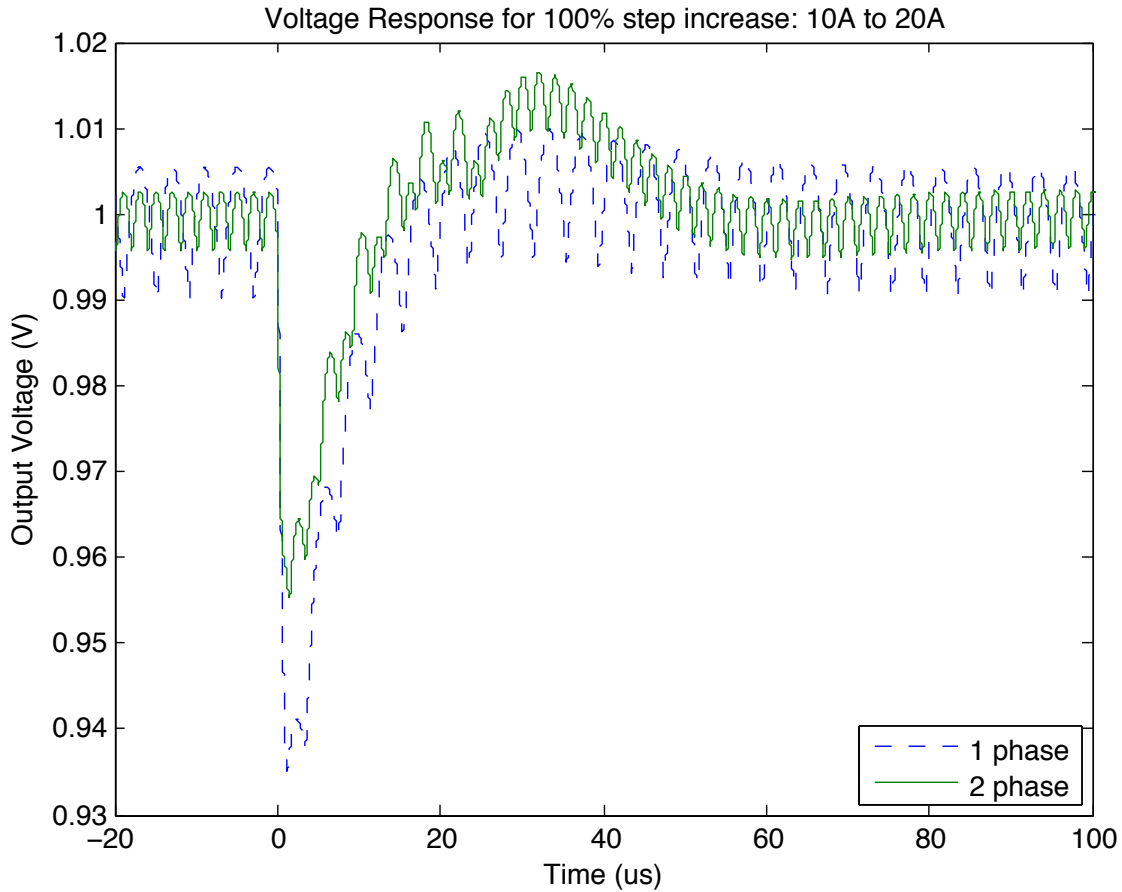


Figure 5.1: Single (dashed) and two-phase (solid) converter output voltage under 100% load step ($10\text{A} \Rightarrow 20\text{A}$).

A 50% load decrease was performed from $0.05\ \Omega$ to $0.1\ \Omega$, or 20 S to 10 S. The voltage response is shown in Fig. 5.3. The current response is shown in Fig 5.4. For the single-phase buck, the output voltage rises 270 mV and settles in 80 μ s. For the two-phase buck, the output voltage rises to 212 mV and settles in 50 μ s.

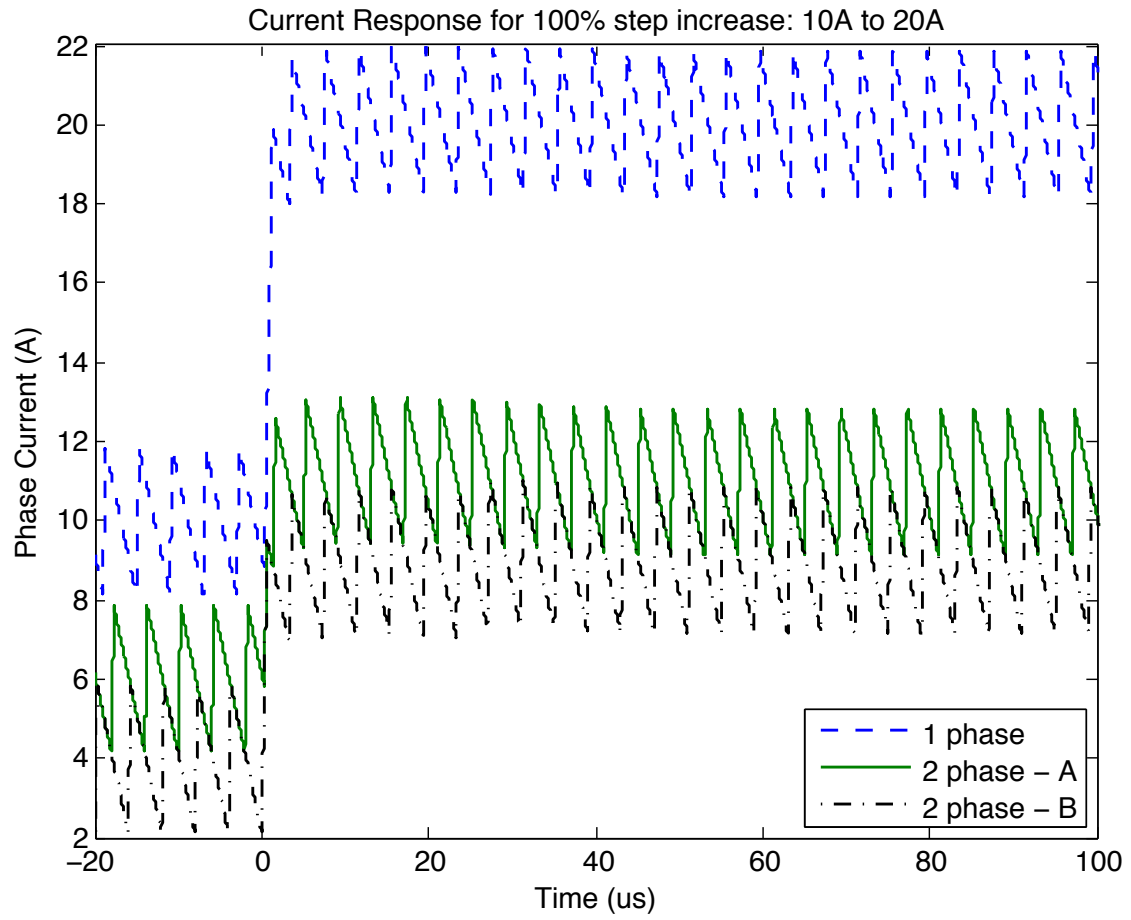


Figure 5.2: Single (dashed) and two-phase (dash-dot, solid) converter phase current under 100% load step ($10\text{A} \Rightarrow 20\text{A}$).

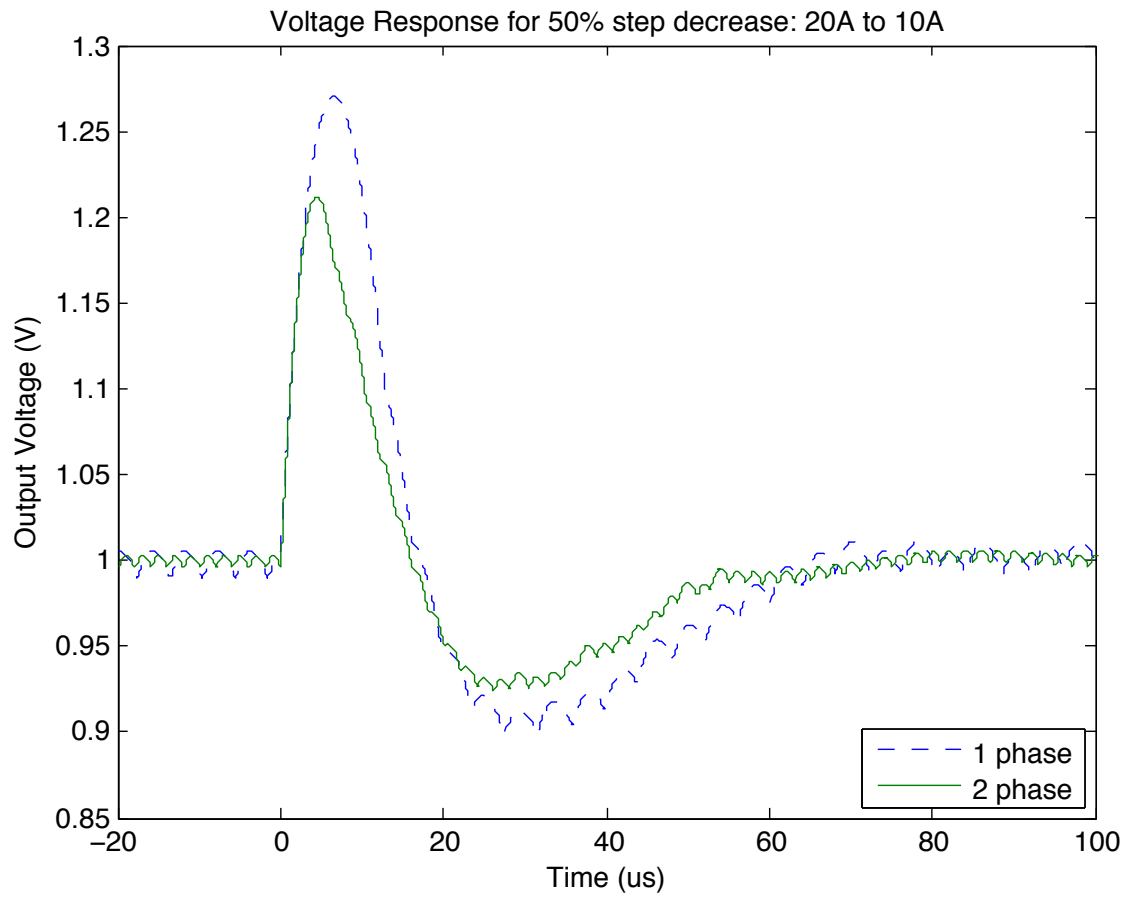


Figure 5.3: Single (dashed) and two-phase (solid) converter output voltage under 50% load drop ($20S \Rightarrow 10S$).

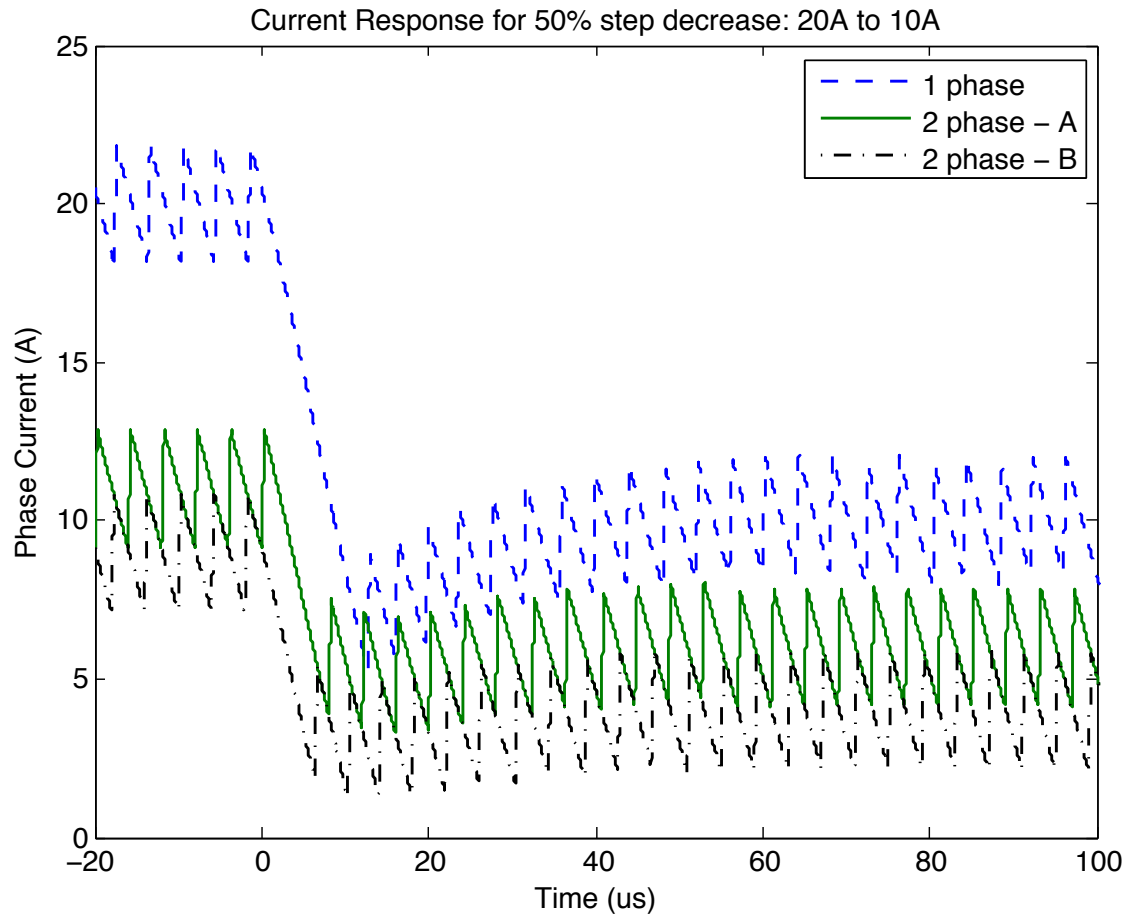


Figure 5.4: Single (dashed) and two-phase (dash-dot, solid) converter phase current under 50% load drop ($20S \Rightarrow 10S$).

Chapter 6

Experimental Setup

The simulated controller is now implemented on hardware. This chapter is broken into two sections. Section one will discuss the hardware used and the test setup. Section two will describe the experimental algorithm as implemented in the controller.

6.1 Implementation Hardware and Test Setup

An example phase is shown in Figure 6.1. Each phase utilizes an EPC 2015 eGaN MOSFET for the primary switch, which is driven by the TI LM5113 eGaN MOSFET driver. Phase currents are sensed by the CQ-2065e high speed current sensor, which has a typical bandwidth from DC to 1MHz and a range of ± 50 A. Each phase also utilizes a MBRB4030T4GOSCT-ND 30 V 40 A Schottky diode for inductor commutation currents. The current version of the buck converter is designed to be modular for testability. Thus, each phase is on its own separate board which plugs into a master carrier board. The connector is a 2 row, 50-pin header connector. Three signals are brought back to the controller's analog to digital converter (ADC): the switching node voltage, labeled as v_{s1} in Fig 4.1, the output voltage, and the output of the current sensor. The inductance and capacitance for each phase is given in Table 6.1.

Parameter	Nominal Value
C_1	121.1 μ F
C_2	121.1 μ F
L_1	100 μ H
L_2	100 μ H
V_i	12V

Table 6.1: Parameter values for the hardware test setup.

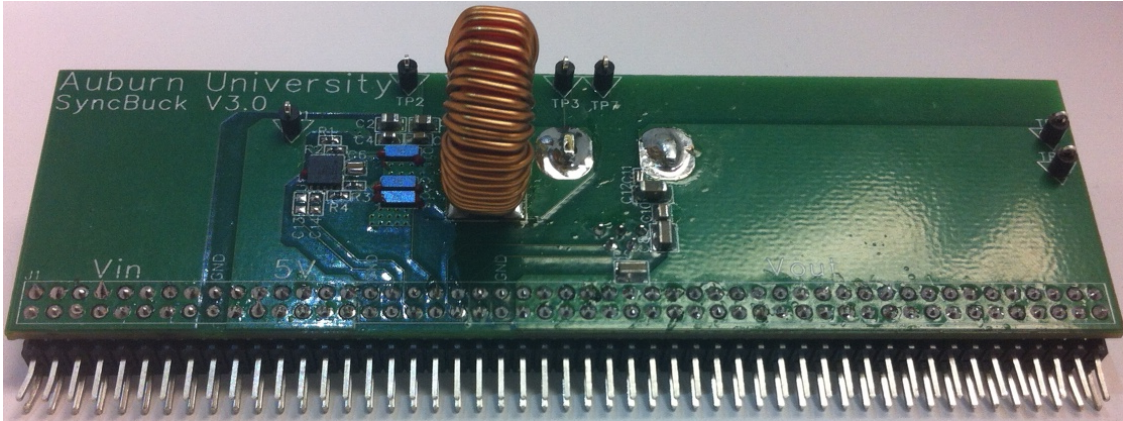


Figure 6.1: Single-phase buck test board.

The controller chip is a TI TMS320F28335 C2000x series processor on a TI ControlCard, shown on the red PCB in Fig. 6.2. This particular chip supports up to 150MHz clock rate with a 8-stage pipeline. It also possesses a 12MHz pipelined ADC broken into two groups of 8. Each group has its own sample and hold circuit which feeds a shared ADC stage. The chip also possess 6 PWM channels capable of duty cycles with high resolution up to 180 picoseconds (ps). Most importantly, the chip supports a 32-bit x 32-bit multiply and accumulate into a 64-bit result register. This allows high numerical precision and a wide range of supported values.

The carrier board, shown in Fig. 6.3, is responsible for connecting up to five buck converter phases and sending signals between the control card and the phases. It is also responsible for combining the output of the individual stages and sending it to the load. The carrier also has 121.1 μ F of capacitance to filter the output currents before sending to the load. The board also contains the 5V TTL-level FTDI compatible serial port and the JTAG programmer port. In addition, there are status LEDs and various input buttons to allow user input without a computer. The board also sports three power inputs across the top. The left-most connection is the digital 5V supply. The center connection is the buck converter's V_{in} . The right most connection is the auxiliary 5V gate-driver supply for the buck converters.

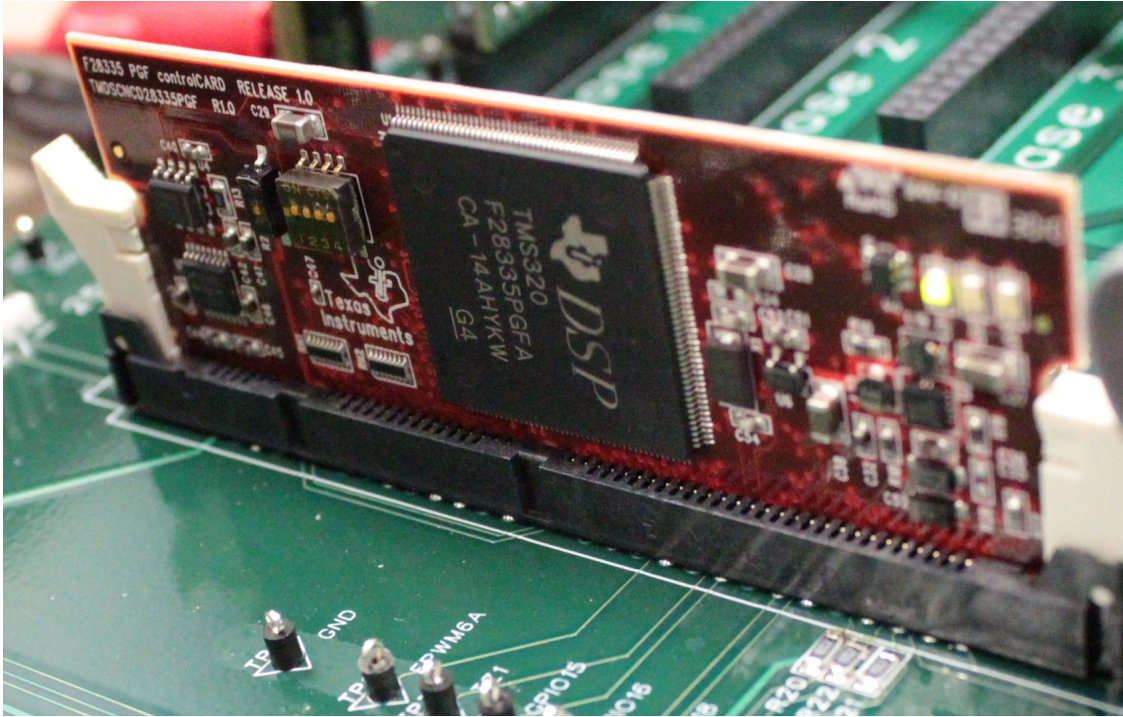


Figure 6.2: TMS320F28335 microcontroller on the ControlCard.

The load for the tests is shown in Fig 6.4. It is comprised of a heavy-copper weight PCB containing surface mount resistors of various values and two voltage inputs. One voltage input is the output voltage of the carrier boards. The other input is a gate voltage for switching MOSFETs. The configuration for the test is two groups of 4 resistors each of 1Ω , 0.5Ω , 0.2Ω , and 0.1Ω . One group is on the west side of the board and the other group is on the east side of the board. The resistors are connected to the output voltage of the carrier board and to the drains of MOSFET transistors. Each transistor has its source connected to ground and its gate connected to a jumper. This allows each resistor to be enabled or disabled based on the position of the jumper, either 5V or ground, respectively. The load board's minimum output resistance is $6.9\text{ m}\Omega$ or, in terms of admittance, 144 siemens.

For load increase tests, the trigger was connected to the gate of the transistor and the jumper was moved from ground to 5V. For load decrease tests, the trigger was also connected

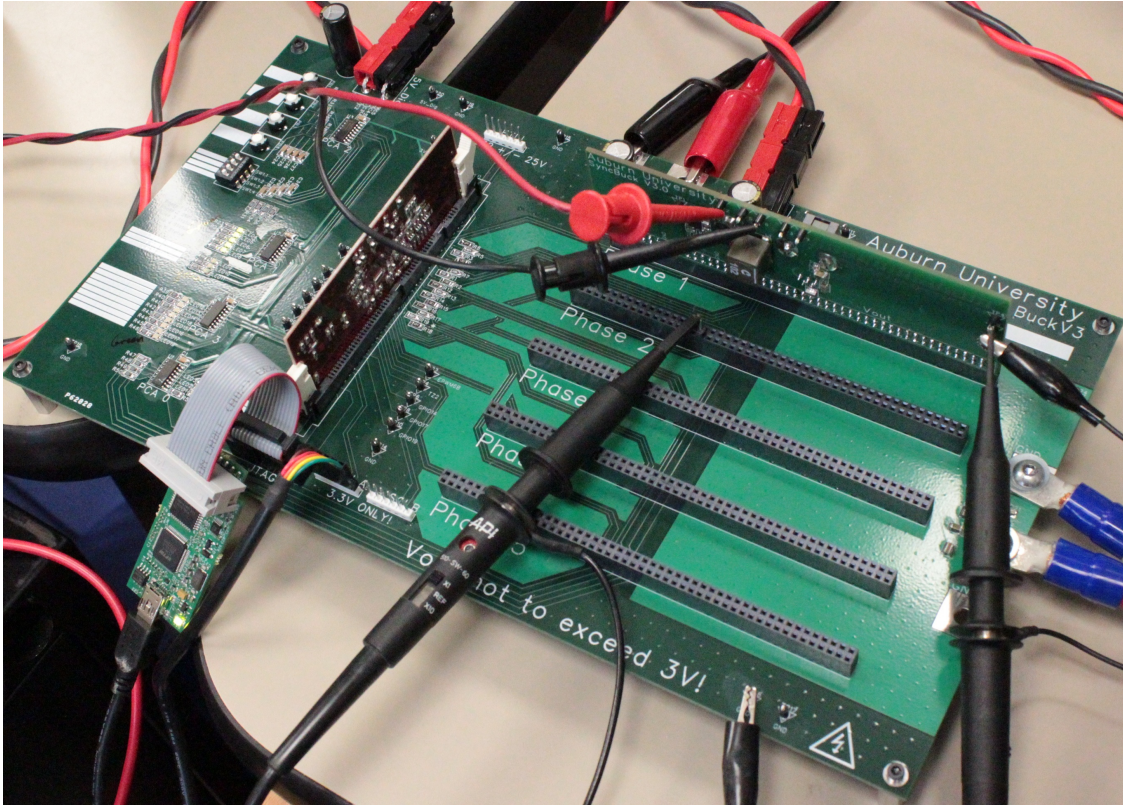


Figure 6.3: Carrier board for testing.

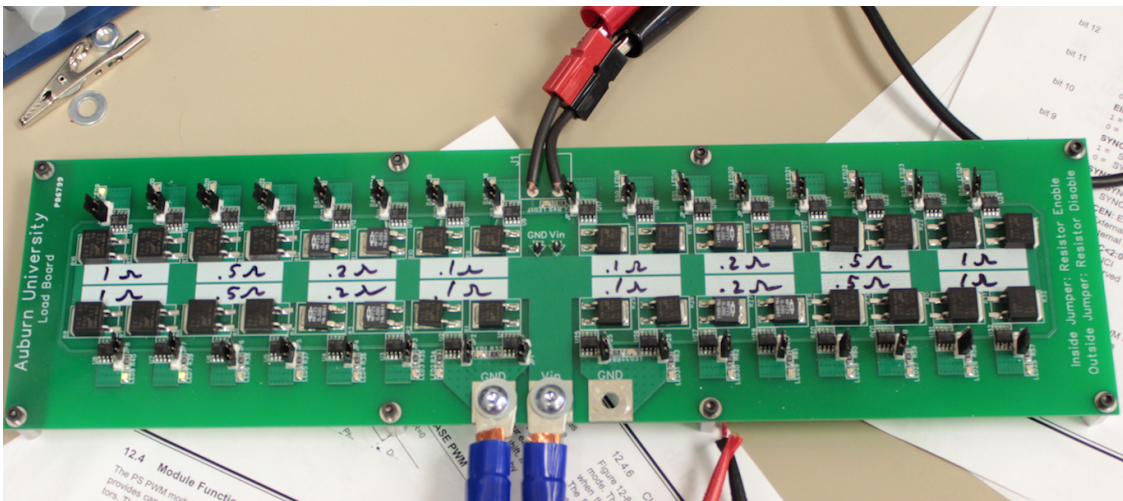


Figure 6.4: Load board used for testing.

to the gate and a pull-up resistor was added between 5V and the gate. The jumper was then connected between the gate and ground, turning off the transistor and disabling the resistor.

The original simulated system has too fast of a time-constant for the direct mode controller to compute the sliding surfaces and maintain decent regulation of the converter at an input voltage of twelve volts. Thus, the decision was made to utilize a larger inductor to slow down the time constant. This problem could be rectified using a faster processor or implementing the logic in a fast FPGA where many calculations could be performed simultaneously.

6.2 Experimental Algorithm

The experimental algorithm is slightly changed from the simulated algorithm due to a few key issues. The controller does not perform as quickly on floating point operations as it does on integer operations. Since the MATLAB/Simulink simulation utilizes floating point at its core, the algorithm should be adapted slightly to utilize 32-bit integer math. Additionally, the MATLAB/Simulink simulation performs multiple convergence loops for each simulation time-step, essentially performing all computations simultaneously. Since the microcontroller can only perform calculations consecutively, the $\text{sign}(u_n)$ terms where $n = 1, 2$ are particularly problematic. This is problematic because the calculation of S_1 requires foreknowledge of S_2 and the calculation of S_2 requires foreknowledge of S_1 .

Additionally, the integration term is allowed to saturate so as not to overflow its integer bounds and to reduce the integrator wind-up effect. Also, the synchronous rectifier is not utilized in this test. Therefore, the phase current term in the simulated system, $\alpha_{Neg}(1 - \text{sign}(i_{Ln})), n = 1, 2$ cannot be negative and consequently the term will always be 0 and can be ignored. The control equations implemented on the TMS320F28335 are given below:

$$S = \alpha_1 x_1 + \alpha_2 z^{-1} x_1 + \alpha_3 z^{-1} x_2 \quad (6.1)$$

$$S_1 = S + \alpha_4 Ramp_1 + \alpha_6 x_6 + \alpha_{C_2} z^{-1} c_2 \quad (6.2)$$

$$S_2 = S + \alpha_4 Ramp_2 + \alpha_7 x_7 + \alpha_{C_1} z^{-1} c_1 \quad (6.3)$$

where x_1 is the error voltage in ADC sample values given by:

$$x_1 = (V_{ref} - V_{ADC}) \quad (6.4)$$

and x_2 is a saturating integrator:

$$x_2 = \max_{x_{2min}} \left(\min_{x_{2max}} \left(\sum \frac{x_1}{k_1} \right) \right) \quad (6.5)$$

and the ramps are defined as resettable ramps:

$$Ramp_1 = \begin{cases} 0 & \text{for } S_1 > \kappa \\ Ramp_1 + 1 & \text{for } S_1 < \kappa \end{cases} \quad (6.6)$$

$$Ramp_2 = \begin{cases} 0 & \text{for } S_2 > \kappa \\ Ramp_2 + 1 & \text{for } S_2 < \kappa \end{cases} \quad (6.7)$$

x_6 and x_7 are resettable decreasing ramps:

$$x_6 = \begin{cases} \tau_{D1} & \text{for } (S_1 - z^{-1}S_1) > 0 \\ \max_{x_{2max}} (x_6 - 1) & \text{for } (S_1 - z^{-1}S_1) \leq 0 \end{cases} \quad (6.8)$$

$$x_7 = \begin{cases} \tau_{D2} & \text{for } (S_2 - z^{-1}S_2) > 0 \\ \max_{x_{2max}} (x_7 - 1) & \text{for } (S_2 - z^{-1}S_2) \leq 0 \end{cases} \quad (6.9)$$

c_1 and c_2 are defined as the output of the controller:

$$c_1 = \begin{cases} 1 & \text{for } S_1 > \kappa \\ 0 & \text{for } S_1 < -\kappa \end{cases} \quad (6.10)$$

$$c_2 = \begin{cases} 1 & \text{for } S_2 > \kappa \\ 0 & \text{for } S_2 < -\kappa \end{cases} \quad (6.11)$$

The coefficients for the hardware algorithm can be found in Table 6.2.

	Single Phase Values	Two Phase Values
Parameter	Value	
V_{ref}	1160 ADC Counts	
V_{ref}	0.85 V	
α_1	10	10
α_2	-2	-2
α_3	1	1
α_4	-	10
α_6	-	1
α_7	-	2
κ	80	80
$neg\kappa$	-80	-80
α_{C1}	-	-200
α_{C2}	-	-200
τ_{D1}	-	10
τ_{D2}	-	5
x_{2max}	0x4000	
x_{2min}	0x4000	

Table 6.2: Parameter values used in the hardware controller.

Chapter 7

Validations through Experiments

The controller implemented in the previous chapter is tested under multiple conditions. Two tests are performed: load increase and load decrease. This chapter is split into three sections. The first section will examine the voltage responses to the load variations. The second section will examine the phase current responses to the same load variations. Finally, the third section will develop some analysis on the outcomes of the data presented in the two previous sections.

7.1 Voltage Responses

The load step is performed from 4 S to 9 S, where $S=\Omega^{-1}$. Fig. 7.1 shows the voltage response with a 5 S step increase applied at $t = 0$. The two phase controller shows less droop and overshoot and recovers slightly faster than the single phase.

The next test performed on the system was a load decrease from 9 S to 4 S. As noted above, the response is very consistent between the single phase and two phase controllers. Fig 7.2 shows the voltage response for a load decrease from 9S to 4S. The single phase voltage response peaks at 1.5 V and then drops to 0.46 V. The two phase voltage response peaks at 1.25 V and then drops to 0.625 V before recovering in 0.7 ms. Fig 7.3 shows the single phase settling time as 1.2 ms.

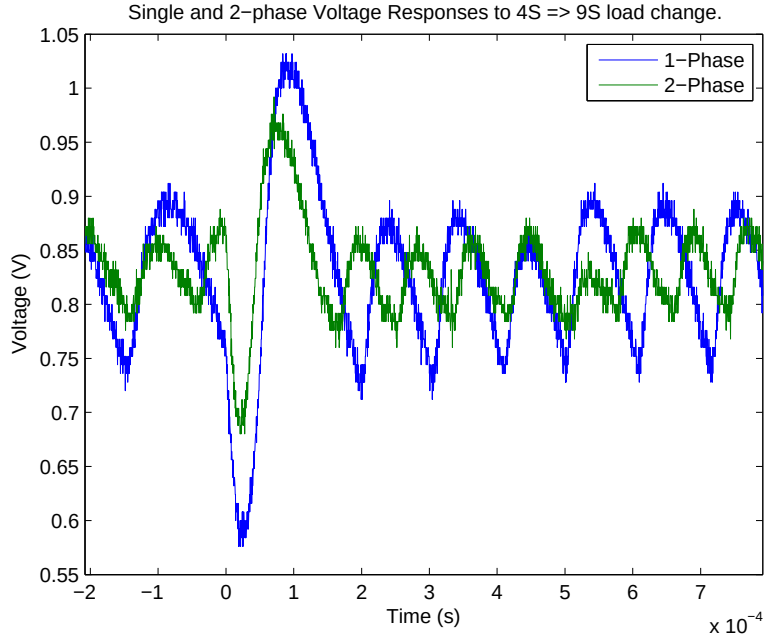


Figure 7.1: Output voltage response of the single and two-phase converter response with 125% load increase ($4S \Rightarrow 9S$) at $t = 0$.

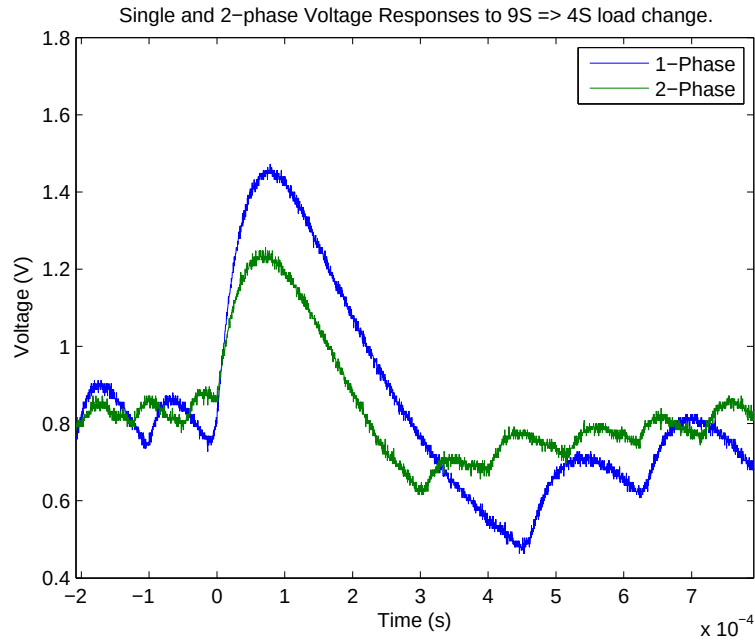


Figure 7.2: Output voltage response of the single and two-phase converter response with 44% load decrease ($9S \Rightarrow 4S$) at $t = 0$.

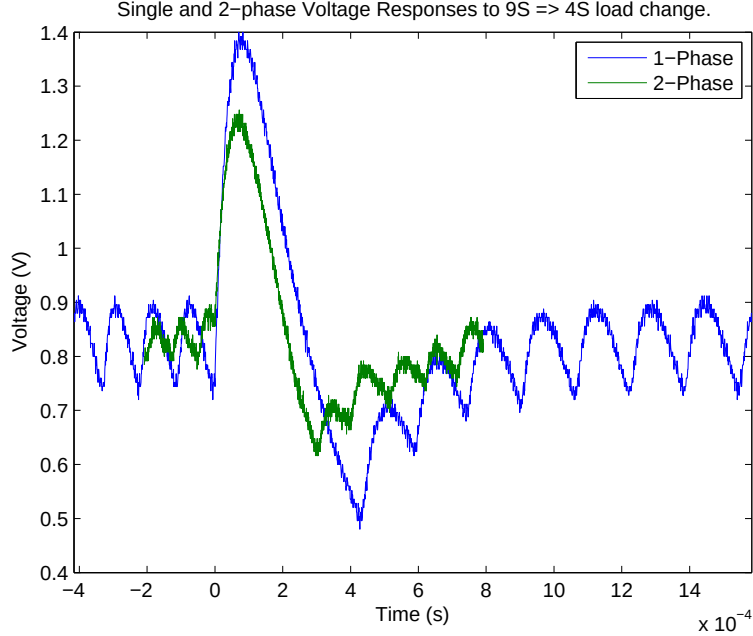


Figure 7.3: Output voltage response of the single and two-phase converter response with 44% load decrease ($9S \Rightarrow 4S$) at $t=0$.

7.2 Phase Current Responses

This section will examine the individual phase current responses and the overall converter current response to the two load changes. The first load change examined is the load increase from 4 S to 9 S. Fig. 7.4 shows the individual phase currents during the load step at $t = 0$. Notice at $t = 0$, all phases turn on and increase the inductor currents at a rate of $(V_{in} - V_o)/L$. Since both phases in the two phase system turn on, the system experiences twice the rise rate as in the single phase case. Fig. 7.5 shows the total phase currents for each system. For the two phase system, the sum of both phases at each time instant is plotted.

Fig. 7.6 shows the output current response for the same 5 S load step. The output capacitor provides the instantaneous current shortly after the load step. The phases recover after the initial surge of current provided by the capacitor as shown by the output current droop around $25 \mu s$. Since both phases in the two phase system turn on during the load step,

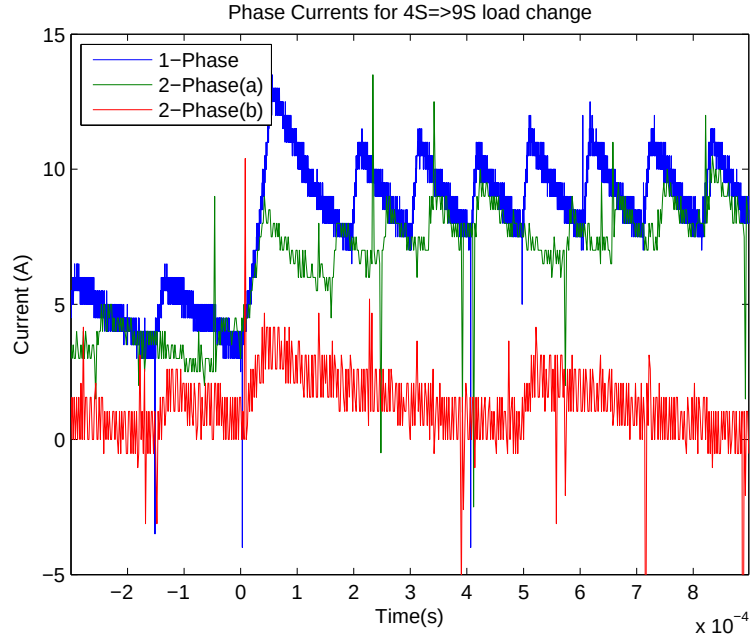


Figure 7.4: Phase current responses of the single and two-phase converter response with 125% load increase ($4S \Rightarrow 9S$) at $t=0$.

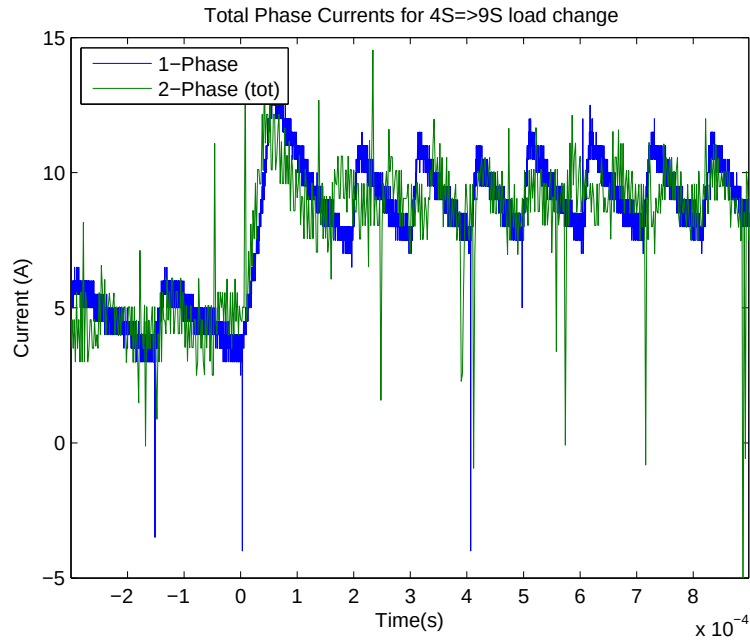


Figure 7.5: Total phase current responses of the single and two-phase converter response with 125% load increase ($4S \Rightarrow 9S$) at $t=0$.

the two phase system recovers closer to the steady-state output current of 7.25 A, giving it a smaller droop and overshoot compared to the single phase system.

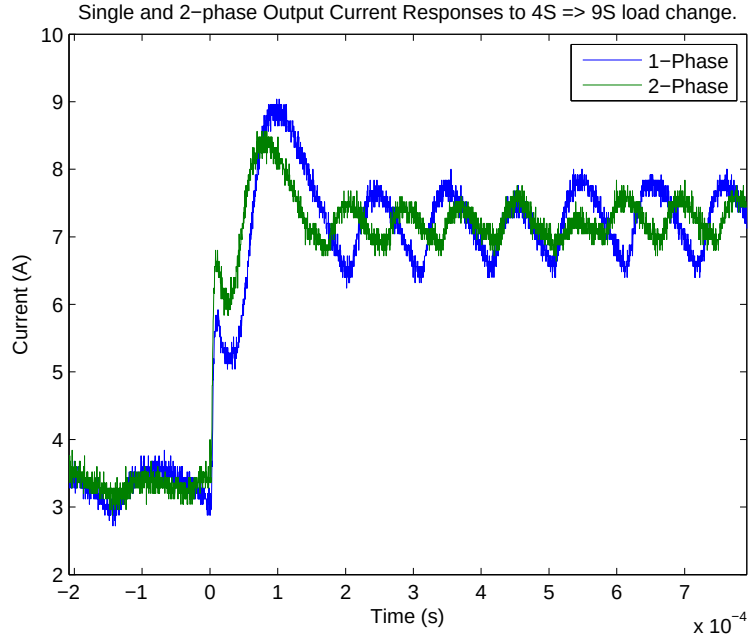


Figure 7.6: Output current response of the single and two-phase converter response with 125% load increase ($4S \Rightarrow 9S$) at $t=0$.

The next test performed on the system was a load decrease from 9 S to 4 S. Fig 7.7 shows the output current response of the converters. At $t = 0$, the output current drops due to the load change from 9 S to 4 S. Since the inductor currents cannot change instantaneously, the output capacitor absorbs the excess current, which results in the voltage peak as shown in Fig. 7.2.

Fig. 7.8 shows the current responses for each phase of the converters. The two phase system can respond faster due to the two inductors both decreasing the current. Fig. 7.9 shows the total phase currents for each converter. Notice both converters produce roughly the same currents indicating that both converters are responding in a similar fashion. This fulfills one of the desired design goals for the two phase controller.

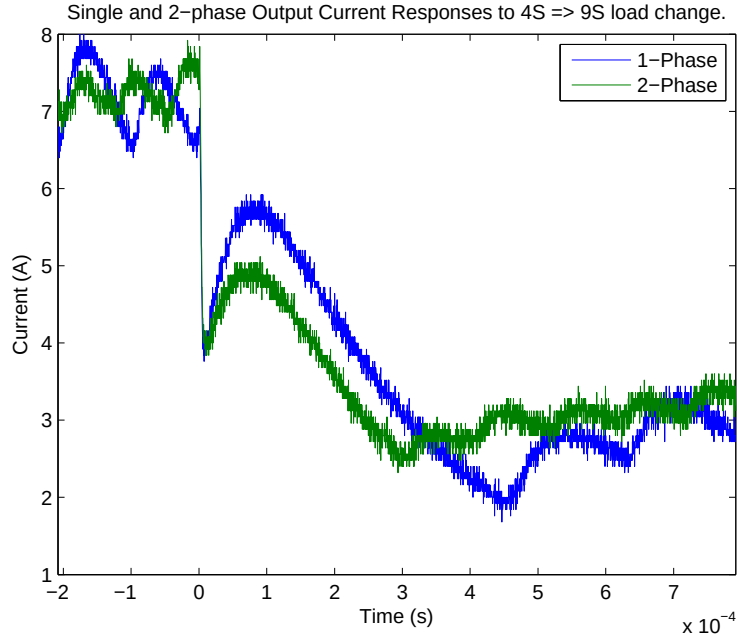


Figure 7.7: Output current response of the single and two-phase converter response with 44% load decrease ($9S \Rightarrow 4S$) at $t=0$.

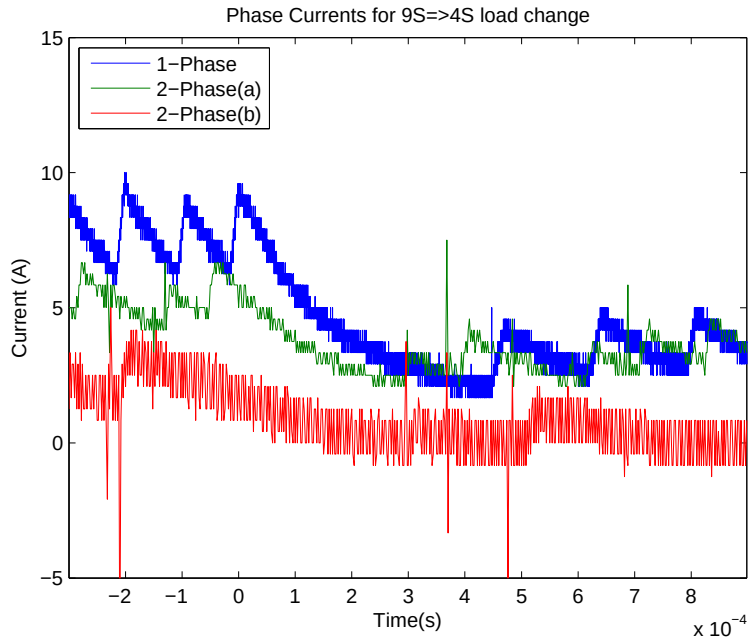


Figure 7.8: Phase current responses of the single and two-phase converter response with 44% load decrease ($9S \Rightarrow 4S$) at $t=0$.

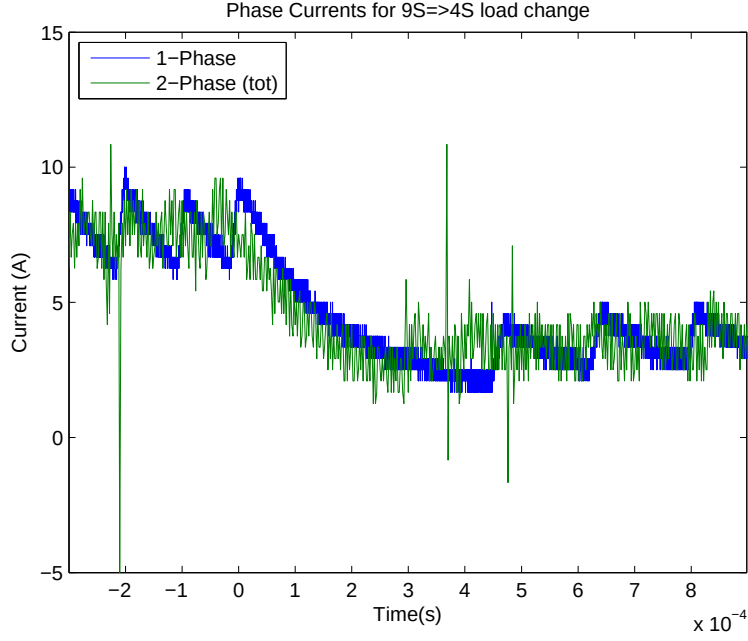


Figure 7.9: Total phase current responses of the single and two-phase converter response with 44% load decrease ($9S \Rightarrow 4S$) at $t=0$.

7.3 Experimental Analysis

As can be seen from the prior two sections, the transient response of the converter in the two phase converter is improved over the single phase converter. As shown in Figs. 7.5 and 7.9, the total output currents of the converter are very similar. Unfortunately, the experimental implementation of the converter does not keep the two phases in perfect phase synchronization. Even though the phase synchronization is not ideal, the steady-state voltage behavior, as shown in Fig 7.1 after $t=2 \times 10^{-4}$, is improved.

Chapter 8

Conclusions

This work designs, simulates, and implements a direct sliding mode controller for a two-phase buck converter. The technical approach extends the the single-phase controller that Tan et al. [66] developed to the multiphase case. Since there are two phases, two switching surfaces are necessary. Four new terms are introduced in each switching surface equation for synchronous rectifying buck converters. These terms address the challenges of phase delay, current balance, and inductor current monitoring. This work is tested in simulation with good results.

Next, the work is implemented on a hardware test platform consisting of a TI TMS320F28335 microcontroller and two buck converter phases. The proposed simulated algorithm is then modified for implementation on the microcontroller. Two load varying tests are performed and the proposed multiphase converter shows improved transient response relative to the single-phase converter. The proposed controller can be extended to support regulating to desired phase currents and dynamically changing the number of active phases based on load conditions.

This work confirms the sliding mode controller is a suitable controller design for multiphase buck converters. The proposed controller shows robustness to load disturbances and shows improved responses over the single-phase sliding mode controller. More importantly, it also forms the basis for future work improving current sharing, implementing phase shedding, and expanding the maximum number of active phases which will improve converter efficiency.

8.1 Future Work

This work can be extended in a number of ways. First, phase shedding or adding based on load conditions can be incorporated into the surfaces. This would allow more flexibility in the delivery of the load current by adding or removing additional phases, improving the efficiency of the overall converter by reducing the per-phase currents. However, this also requires monitoring the currents in each phase. Something that is not currently required by the controller as proposed in hardware and incurs additional cost and computation.

Second, current regulation per phase based on the load condition can also be incorporated to enable each phase to operate at or near its peak efficiency. Current regulation complements phase shedding so that the converter can maintain peak efficiency regardless of load conditions.

Finally, the work can be extended to support higher number of phases while maintaining the desired voltage responses and distributing current between the phases efficiently. This allows the converter to support wider operation ranges and higher maximum load currents for the same basic phase design.

Bibliography

- [1] A. Costabeber, P. Mattavelli, and S. Saggini, “Digital time-optimal phase shedding in multiphase buck converters,” *Power Electronics, IEEE Transactions on*, vol. 25, no. 9, pp. 2242–2247, sept. 2010.
- [2] M. Poess and R. Nambiar, “Tuning servers, storage and database for energy efficient data warehouses,” in *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, March 2010, pp. 1006–1017.
- [3] M. Patterson, “The effect of data center temperature on energy efficiency,” in *Thermal and Thermomechanical Phenomena in Electronic Systems, 2008. ITherm 2008. 11th Intersociety Conference on*, may 2008, pp. 1167–1174.
- [4] P. Stanley-Marbell and V. Cabezas, “Performance, Power, and Thermal Analysis of Low-Power Processors for Scale-Out Systems,” in *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, may 2011, pp. 863–870.
- [5] Y. Hayashi and M. Yamasaki, “Optimal design of a high density back-end converter for 400v dc distribution systematic,” in *Power Electronics and Motion Control Conference (EPE/PEMC), 2010 14th International*, Sept. 2010, pp. T6–40–T6–45.
- [6] Intel, *Intel Core™ i7-900 Mobile Processor Extreme Edition Series, Intel Core i7-800 and i7-700 Mobile Processor Series Datasheet - Volume 1*, Intel, 2009.
- [7] ALTERA, *Stratix V Device Handbook - Volume 1: Overview and Datasheet*, 11.1 ed., 2011.
- [8] M. Briere, “Advanced power devices for many-core processor power supplies,” in *Electron Devices Meeting (IEDM), 2010 IEEE International*, Dec. 2010, pp. 13.6.1–13.6.4.
- [9] R. Retegui, M. Benedetti, M. Funes, P. Antoszczuk, and D. Carrica, “Current control for high-dynamic high-power multiphase buck converters,” *Power Electronics, IEEE Transactions on*, vol. 27, no. 2, pp. 614–618, feb. 2012.
- [10] J. Vilchis, C. Aguilar, and J. Arau, “A Multiphase Structure Applied in VRs for Portable Applications,” in *Applied Power Electronics Conference and Exposition, 2009. APEC 2009. Twenty-Fourth Annual IEEE*, Feb. 2009, pp. 1353–1359.
- [11] L. Niu and G. Quan, “System-Wide Dynamic Power Management for Portable Multimedia Devices,” in *Multimedia, 2006. ISM’06. Eighth IEEE International Symposium on*, Dec. 2006, pp. 97–104.

- [12] A. Amaral and A. Cardoso, "Using a CLC Low-pass Filter to Reduce the Consequences of Aging in Steady-state Regime of DC-DC Converters," in *Power Electronics and Applications, 2005 European Conference on*, 2005, p. 10 pp.
- [13] V. Utkin, J. Guldner, and J. X. Shi, *Sliding Mode Control in Electromechanical Systems*. London, U.K.: Taylor & Francis, 1999.
- [14] M. Ahmed, M. Kuisma, K. Tolsa, and P. Silventoinen, "Implementing sliding mode control for buck converter," in *Power Electronics Specialist Conference, 2003. PESC '03. 2003 IEEE 34th Annual*, vol. 2, June 2003, pp. 634 – 637 vol.2.
- [15] A. Perry, G. Feng, Y.-F. Liu, and P. Sen, "A new sliding mode like control method for buck converter," in *Power Electronics Specialists Conference, 2004. PESC 04. 2004 IEEE 35th Annual*, vol. 5, June 2004, pp. 3688 – 3693.
- [16] H. Li and X. Ye, "Sliding-mode pid control of dc-dc converter," in *Industrial Electronics and Applications (ICIEA), 2010 the 5th IEEE Conference on*, june 2010, pp. 730 –734.
- [17] R. Kordkheili and M. Mehrasa, "Sliding mode control for three-phase ac/dc voltage-source boost converter," in *Power Electronics Electrical Drives Automation and Motion (SPEEDAM), 2010 International Symposium on*, june 2010, pp. 1213 –1217.
- [18] D. Biel, F. Guinjoan, E. Fossas, and J. Chavarria, "Sliding-mode control design of a boost-buck switching converter for ac signal generation," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 51, no. 8, pp. 1539 – 1551, aug. 2004.
- [19] H. Lee, V. I. Utkin, and A. Malinin, "Chattering reduction using multiphase sliding mode control," *International Journal of Control*, vol. 82, no. 9, pp. 1720–1737, 2009. [Online]. Available: <http://www.tandfonline.com/doi/abs/10.1080/00207170902725607>
- [20] W. Huang, G. Schuellein, and D. Clavette, "A scalable multiphase buck converter with average current share bus," in *Applied Power Electronics Conference and Exposition, 2003. APEC '03. Eighteenth Annual IEEE*, vol. 1, feb. 2003, pp. 438 –443 vol.1.
- [21] Y. Wen and O. Trescases, "DC-DC Converter With Digital Adaptive Slope Control in Auxiliary Phase for Optimal Transient Response and Improved Efficiency," *Power Electronics, IEEE Transactions on*, vol. 27, no. 7, pp. 3396 –3409, july 2012.
- [22] X. Du, L. Zhou, and H.-M. Tai, "Double-frequency buck converter," *Industrial Electronics, IEEE Transactions on*, vol. 56, no. 5, pp. 1690 –1698, May 2009.
- [23] B. Mahdavihah, P. Jain, and A. Prodic, "Digitally controlled multi-phase buck-converter with merged capacitive attenuator," in *Applied Power Electronics Conference and Exposition (APEC), 2012 Twenty-Seventh Annual IEEE*, feb. 2012, pp. 1083 –1087.

- [24] R. Panguloori, D. Kastha, A. Patra, and G. Capodivacca, "High performance voltage regulator for high step-down dc-dc conversion," in *Industrial Electronics, 2008. IECON 2008. 34th Annual Conference of IEEE*, nov. 2008, pp. 761 –765.
- [25] K. Abe, K. Nishijima, K. Harada, T. Nakano, T. Nabeshima, and T. Sato, "A novel multi-phase buck converter for lap-top pc," in *Power Conversion Conference - Nagoya, 2007. PCC '07*, april 2007, pp. 885 –891.
- [26] I.-O. Lee, S. young Cho, and G.-W. Moon, "Interleaved buck converter having low switching losses and improved step-down conversion ratio," in *Power Electronics and ECCE Asia (ICPE ECCE), 2011 IEEE 8th International Conference on*, 30 2011-june 3 2011, pp. 2136 –2143.
- [27] O. Garcia, P. Zumel, A. de Castro, P. Alou, and J. Cobos, "Current Self-Balance Mechanism in Multiphase Buck Converter," *Power Electronics, IEEE Transactions on*, vol. 24, no. 6, pp. 1600 –1606, june 2009.
- [28] K. Rahimi, A. Motlagh, and M. Pakdel, "A novel soft-switched synchronous buck converter," in *Vehicle Power and Propulsion Conference, 2009. VPPC '09. IEEE*, sept. 2009, pp. 1345 –1351.
- [29] K. Hwu and Y. Chen, "Applying differential-mode transformer to current sharing with current ripple considered," *Industrial Electronics, IEEE Transactions on*, vol. 58, no. 7, pp. 2755 –2771, july 2011.
- [30] Y. Jang, M. Jovanovic, and Y. Panov, "Multiphase buck converters with extended duty cycle," in *Applied Power Electronics Conference and Exposition, 2006. APEC '06. Twenty-First Annual IEEE*, march 2006, p. 7 pp.
- [31] P. Xu, J. Wei, K. Yao, Y. Meng, and F. Lee, "Investigation of candidate topologies for 12 v vrm," in *Applied Power Electronics Conference and Exposition, 2002. APEC 2002. Seventeenth Annual IEEE*, vol. 2, 2002, pp. 686 –692 vol.2.
- [32] P. Xu, J. Wei, and F. Lee, "Multiphase coupled-buck converter-a novel high efficient 12 v voltage regulator module," *Power Electronics, IEEE Transactions on*, vol. 18, no. 1, pp. 74 – 82, jan 2003.
- [33] H. Nagaraja, D. Kastha, and A. Petra, "Design principles of a symmetrically coupled inductor structure for multiphase synchronous buck converters," *Industrial Electronics, IEEE Transactions on*, vol. 58, no. 3, pp. 988 –997, march 2011.
- [34] W. Wu, N.-C. Lee, and G. Schuellein, "Multi-phase buck converter design with two-phase coupled inductors," in *Applied Power Electronics Conference and Exposition, 2006. APEC '06. Twenty-First Annual IEEE*, march 2006, p. 6 pp.
- [35] S. Wibowo, Z. Ting, M. Kono, T. Taura, Y. Kobori, and H. Kobayashi, "Analysis of coupled inductors for low-ripple fast-response buck converter," in *Circuits and Systems, 2008. APCCAS 2008. IEEE Asia Pacific Conference on*, 30 2008-dec. 3 2008, pp. 1860 –1863.

- [36] D. Garinto, “New converter architectures with multi-interleaving technique for future microprocessors,” in *Telecommunications Energy Conference, 2006. INTELEC '06. 28th Annual International*, sept. 2006, pp. 1 –7.
- [37] —, “A novel multiphase multi-interleaving buck converters for future microprocessors,” in *Power Electronics and Motion Control Conference, 2006. EPE-PEMC 2006. 12th International*, 30 2006-sept. 1 2006, pp. 82 –87.
- [38] T. Taufik, R. Prasetyo, A. Hernadi, and D. Garinto, “Multiphase interleaving buck converter with input-output bypass capacitor,” in *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*, april 2010, pp. 1207 – 1211.
- [39] T. Taufik, R. Prasetyo, D. Dolan, and D. Garinto, “A new multiphase multi-interleaving buck converter with bypass lc,” in *IECON 2010 - 36th Annual Conference on IEEE Industrial Electronics Society*, nov. 2010, pp. 291 –295.
- [40] Y. Qiu, M. Xu, K. Yao, J. Sun, and F. Lee, “Multifrequency small-signal model for buck and multiphase buck converters,” *Power Electronics, IEEE Transactions on*, vol. 21, no. 5, pp. 1185 –1192, sept. 2006.
- [41] N. Kondrath and M. Kazimierczuk, “Comparison of wide- and high-frequency duty-ratio-to-inductor-current transfer functions of dc-dc pwm buck converter in ccm,” *Industrial Electronics, IEEE Transactions on*, vol. 59, no. 1, pp. 641 –643, jan. 2012.
- [42] Y. Qiu, K. Yao, Y. Meng, M. Xu, F. Lee, and M. Ye, “Control-loop bandwidth limitations for multiphase interleaving buck converters,” in *Applied Power Electronics Conference and Exposition, 2004. APEC '04. Nineteenth Annual IEEE*, vol. 2, 2004, pp. 1322 – 1328 vol.2.
- [43] G. Eirea and S. Sanders, “Phase current unbalance estimation in multi-phase buck converters,” in *Power Electronics Specialists Conference, 2006. PESC '06. 37th IEEE*, june 2006, pp. 1 –6.
- [44] N. Bouhalli, E. Sarraute, T. Meynard, M. Cousineau, and E. Laboure, “Optimal multi-phase coupled buck converter architecture dedicated to strong power system integration,” in *Power Electronics, Machines and Drives, 2008. PEMD 2008. 4th IET Conference on*, april 2008, pp. 352 –356.
- [45] P. Shao, F. Chang, C. Reade, P. Ilavarasan, and D. Pommerenke, “Finding return current paths via synchronized measurements in a multiphase dcdc buck converters,” in *Electromagnetic Compatibility, 2009. EMC 2009. IEEE International Symposium on*, aug. 2009, pp. 53 –57.
- [46] A. Pascual, G. Eirea, and E. Ferreira, “A control strategy for multi-phase buck converters under dynamical selection of active phases,” in *Energy Conversion Congress and Exposition, 2009. ECCE 2009. IEEE*, Sept. 2009, pp. 1112 –1117.

- [47] C.-H. Chiu, D. Chen, C.-J. Chen, and W.-H. Chang, "Modeling and mitigation of dynamic load beat-frequency oscillation in multiphase voltage regulators with high-gain peak current control scheme," in *Applied Power Electronics Conference and Exposition (APEC), 2010 Twenty-Fifth Annual IEEE*, feb. 2010, pp. 1727 –1731.
- [48] K. Zhang, S. Luo, J. Breen, S. Lin, T. Wu, Z. Shen, and I. Batarseh, "Analysis and design of dynamic voltage regulation in multiphase buck converter," in *Applied Power Electronics Conference and Exposition (APEC), 2012 Twenty-Seventh Annual IEEE*, feb. 2012, pp. 1298 –1302.
- [49] J. Agrawal, D. Kastha, A. Patra, and B. Culpepper, "An improved control scheme for multiphase buck converter circuits used in voltage regulator modules," in *Power Electronics and Drives Systems, 2005. PEDS 2005. International Conference on*, vol. 1, 0-0 2005, pp. 418 –423.
- [50] P. Cheng, M. Vasi/c and, O. Garcia, J. Oliver, P. Alou, and J. Cobos, "Multiphase buck converter with minimum time control strategy for rf envelope modulation," in *Applied Power Electronics Conference and Exposition (APEC), 2011 Twenty-Sixth Annual IEEE*, march 2011, pp. 904 –909.
- [51] A. Borrell, M. Castilla, J. Miret, J. Matas, and L. de Vicuna, "Simple low-cost hysteretic controller for multiphase synchronous buck converters," *Industrial Electronics, IEEE Transactions on*, vol. 58, no. 6, pp. 2355 –2365, june 2011.
- [52] S. Concepción Huerta, P. Alou, J. Oliver, O. Garcia, J. Cobos, and A. Abou-Alfotouh, "Nonlinear control for dc-dc converters based on hysteresis of the current with a frequency loop to operate at constant frequency," *Industrial Electronics, IEEE Transactions on*, vol. 58, no. 3, pp. 1036 –1043, march 2011.
- [53] K. K.-S. Leung and H. S.-H. Chung, "Dynamic hysteresis band control of the buck converter with fast transient response," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol. 52, no. 7, pp. 398 – 402, july 2005.
- [54] J. Vilchis, C. Aguilar, and J. Arau, "Multi-mode synchronous buck converter with non-uniform current distribution for portable applications," in *Power Electronics Congress, 2008. CIEP 2008. 11th IEEE International*, aug. 2008, pp. 114 –120.
- [55] V. Boscaïno, M. Gaeta, G. Capponi, and F. Marino, "Non-linear digital control improving transient response: Design and test on a multiphase vrm," in *Power Electronics Electrical Drives Automation and Motion (SPEEDAM), 2010 International Symposium on*, june 2010, pp. 507 –512.
- [56] A. Barrado, J. Quintero, A. Lazaro, C. Fernandez, P. Zumel, and E. Olías, "Linear-non-linear control applied in multiphase vrm," in *Power Electronics Specialists Conference, 2005. PESC '05. IEEE 36th*, june 2005, pp. 904 –909.
- [57] W. Du, C. Wang, and Z. Liu, "Digital control approaches to mitigate the impact of loop competition in multiphase dc-dc converter," in *Industrial Electronics and Applications (ICIEA), 2011 6th IEEE Conference on*, june 2011, pp. 2337 –2342.

- [58] M. Zaman, P. Cao, O. Trescases, and W. T. Ng, "H2-optimal thermal management for multi-phase current mode buck converters," in *Energy Conversion Congress and Exposition (ECCE), 2011 IEEE*, sept. 2011, pp. 4177 –4182.
- [59] R. Orosco and N. Vazquez, "Discrete sliding mode control for dc/dc converters," in *Power Electronics Congress, 2000. CIEP 2000. VII IEEE International*, 2000, pp. 231 –236.
- [60] S. Janardhanan and B. Bandyopadhyay, "Output feedback sliding-mode control for uncertain systems using fast output sampling technique," *Industrial Electronics, IEEE Transactions on*, vol. 53, no. 5, pp. 1677 –1682, oct. 2006.
- [61] V. Bandal and P. Vernekar, "Design of a discrete-time sliding mode controller for a magnetic levitation system using multirate output feedback," in *American Control Conference (ACC), 2010*, 30 2010-july 2 2010, pp. 4289 –4294.
- [62] J. Matas, L. de Vicuna, O. Lopez, M. Lopez, and M. Castilla, "Discrete sliding mode control of a boost converter for output voltage tracking," in *Power Electronics and Variable Speed Drives, 2000. Eighth International Conference on (IEE Conf. Publ. No. 475)*, 2000, pp. 351 –354.
- [63] Y. Li, J.-B. Son, and J.-M. Lee, "Pmsm speed controller using switching algorithm of pd and sliding mode control," in *ICCAS-SICE, 2009*, aug. 2009, pp. 1260 –1266.
- [64] S.-C. Tan, Y. Lai, C. Tse, and M. Cheung, "A fixed-frequency pulsewidth modulation based quasi-sliding-mode controller for buck converters," *Power Electronics, IEEE Transactions on*, vol. 20, no. 6, pp. 1379 – 1392, nov. 2005.
- [65] S.-C. Tan, Y. Lai, M. Cheung, and C. Tse, "On the Practical Design of a Sliding Mode Voltage Controlled Buck Converter," *Power Electronics, IEEE Transactions on*, vol. 20, no. 2, pp. 425 – 437, march 2005.
- [66] S.-C. Tan, Y. Lai, and C. Tse, "General Design Issues of Sliding-Mode Controllers in DC-DC Converters," *Industrial Electronics, IEEE Transactions on*, vol. 55, no. 3, pp. 1160 –1174, march 2008.
- [67] H. Sira-Ramirez, "A geometric approach to pulse-width modulated control in nonlinear dynamical systems," *Automatic Control, IEEE Transactions on*, vol. 34, no. 2, pp. 184 –187, feb 1989.
- [68] M. Iqbal, A. Bhatti, and Q. Khan, "Dynamic sliding modes control of uncertain non-linear mimo three tank system," in *Multitopic Conference, 2009. INMIC 2009. IEEE 13th International*, dec. 2009, pp. 1 –7.
- [69] J. Ackermann and V. Utkin, "Sliding mode control design based on ackermann's formula," *Automatic Control, IEEE Transactions on*, vol. 43, no. 2, pp. 234 –237, feb 1998.

- [70] K. Mohanty, "Sensorless sliding mode control of induction motor drives," in *TENCON 2008 - 2008 IEEE Region 10 Conference*, nov. 2008, pp. 1 –6.
- [71] B. Cardoso, A. Moreira, B. Menezes, and P. Cortizo, "Analysis of switching frequency reduction methods applied to sliding mode controlled dc-dc converters," in *Applied Power Electronics Conference and Exposition, 1992. APEC '92. Conference Proceedings 1992., Seventh Annual*, feb 1992, pp. 403 –410.
- [72] V. Nguyen and C. Lee, "Tracking control of buck converter using sliding-mode with adaptive hysteresis," in *Power Electronics Specialists Conference, 1995. PESC '95 Record., 26th Annual IEEE*, vol. 2, jun 1995, pp. 1086 –1093 vol.2.
- [73] —, "Indirect implementations of sliding-mode control law in buck-type converters," in *Applied Power Electronics Conference and Exposition, 1996. APEC '96. Conference Proceedings 1996., Eleventh Annual*, vol. 1, mar 1996, pp. 111 –115 vol.1.
- [74] J. Fei, "A class of adaptive sliding mode controller with integral sliding surface," in *Mechatronics and Automation, 2009. ICMA 2009. International Conference on*, aug. 2009, pp. 1156 –1161.
- [75] H. Yu and U. Ozguner, "Adaptive Seeking Sliding Mode Control," in *American Control Conference, 2006*, June 2006, p. 6 pp.
- [76] M. Comanescu and L. Xu, "Sliding-mode MRAS Speed Estimators for Sensorless Vector Control of Induction Machine," *Industrial Electronics, IEEE Transactions on*, vol. 53, no. 1, pp. 146 – 153, feb. 2005.
- [77] R.-J. Wai, "Adaptive Sliding-mode Control for Induction Servomotor Drive," *Electric Power Applications, IEE Proceedings -*, vol. 147, no. 6, pp. 553 –562, Nov 2000.
- [78] C.-C. Kung and W.-C. Lai, "GA-based design of a region-wise fuzzy sliding mode controller," in *Electrical and Computer Engineering, 1999 IEEE Canadian Conference on*, vol. 2, may 1999, pp. 971 –976 vol.2.
- [79] F. Betin, A. Sivert, A. Yazidi, and G.-A. Capolino, "Determination of scaling factors for fuzzy logic control using the sliding-mode approach: Application to control of a dc machine drive," *Industrial Electronics, IEEE Transactions on*, vol. 54, no. 1, pp. 296 –309, feb. 2007.
- [80] X. Zhang, Z. Man, and X. Yu, "A new fuzzy sliding mode control scheme," in *Intelligent Control and Automation, 2000. Proceedings of the 3rd World Congress on*, vol. 3, 2000, pp. 1692 –1696 vol.3.
- [81] V. Raviraj and P. Sen, "Comparative study of proportional-integral, sliding mode, and fuzzy logic controllers for power converters," *Industry Applications, IEEE Transactions on*, vol. 33, no. 2, pp. 518 –524, mar/apr 1997.

- [82] M. Xiaojiang and C. Yangzhou, "Neural Sliding Mode Control for Multi-link Robots," in *Control and Decision Conference, 2008. CCDC 2008. Chinese*, July 2008, pp. 3513–3517.
- [83] A. V. Topalov, G. L. Cascella, V. Giordano, F. Cupertino, and O. Kaynak, "Sliding mode neuro-adaptive control of electric drives," *Industrial Electronics, IEEE Transactions on*, vol. 54, no. 1, pp. 671–679, feb. 2007.
- [84] D. Qian, J. Yi, and D. Zhao, "Hierarchical sliding mode control to swing up a pendubot," in *American Control Conference, 2007. ACC '07*, july 2007, pp. 5254–5259.
- [85] X. Yu, S. Wei, and L. Guo, "Cascade Sliding Mode Control for Bicycle Robot," in *Artificial Intelligence and Computational Intelligence (AICI), 2010 International Conference on*, vol. 1, Oct. 2010, pp. 62–66.
- [86] J. Mahdavi, A. Emadi, and H. Toliyat, "Application of state space averaging method to sliding mode control of pwm dc/dc converters," in *Industry Applications Conference, 1997. Thirty-Second IAS Annual Meeting, IAS '97., Conference Record of the 1997 IEEE*, vol. 2, Oct 1997, pp. 820–827 vol.2.
- [87] J. Mahdavi, M. Nasiri, and A. Agah, "Application of neural networks and state space averaging to a dc/dc pwm converter in sliding mode operation," in *Industrial Electronics Society, 2000. IECON 2000. 26th Annual Conference of the IEEE*, vol. 1, 2000, pp. 172–177 vol.1.
- [88] E. Fossas, L. Martinez, and J. Ordinas, "Sliding mode control reduces audiosusceptibility and load perturbation in the cuk converter," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, vol. 39, no. 10, pp. 847–849, oct 1992.
- [89] R. Giral, L. Martinez-Salamero, R. Leyva, and J. Maixe, "Sliding-mode control of interleaved boost converters," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, vol. 47, no. 9, pp. 1330–1339, sep 2000.
- [90] M. Lopez, L. de Vicuna, M. Castilla, P. Gaya, and O. Lopez, "Current distribution control design for paralleled dc/dc converters using sliding-mode control," *Industrial Electronics, IEEE Transactions on*, vol. 51, no. 2, pp. 419–428, april 2004.
- [91] Y. Shtessel, A. Zinober, and I. Shkolnikov, "Boost and buck-boost power converters control via sliding modes using method of stable system centre," in *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*, vol. 1, dec. 2002, pp. 340–345 vol.1.
- [92] M. Ahmed, M. Kuisma, P. Silventoinen, and O. Pyrhonen, "Effect of implementing sliding mode control on the dynamic behavior and robustness of switch mode power supply (buck converter)," in *Power Electronics and Drive Systems, 2003. PEDS 2003. The Fifth International Conference on*, vol. 2, nov. 2003, pp. 1364–1368 Vol.2.

- [93] N. Vazquez, C. Hernandez, J. Alvarez, and J. Arau, "Sliding Mode Control for DC/DC Converters: a New Sliding Surface," in *Industrial Electronics, 2003. ISIE '03. 2003 IEEE International Symposium on*, vol. 1, June 2003, pp. 422 – 426 vol. 1.
- [94] H. Sira-Ramirez, "Sliding mode- delta; modulation control of a "buck" converter," in *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, vol. 3, dec. 2003, pp. 2999 – 3004 Vol.3.
- [95] E. Alarcon, A. Romero, A. Poveda, S. Porta, and L. Martinez-Salamero, "Sliding-mode control analog integrated circuit for switching dc-dc power converters," in *Circuits and Systems, 2001. ISCAS 2001. The 2001 IEEE International Symposium on*, vol. 1, may 2001, pp. 500 –503 vol. 1.
- [96] E. Fossas and A. Ras, "Second-order sliding-mode control of a buck converter," in *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*, vol. 1, dec. 2002, pp. 346 – 347 vol.1.
- [97] S. Mazumder, A. Nayfeh, and A. Borojevic, "Robust control of parallel dc-dc buck converters by combining integral-variable-structure and multiple-sliding-surface control schemes," *Power Electronics, IEEE Transactions on*, vol. 17, no. 3, pp. 428 –437, may 2002.
- [98] S. Mazumder and S. Kamisetty, "Experimental validation of a novel multiphase non-linear vrm controller," in *Power Electronics Specialists Conference, 2004. PESC 04. 2004 IEEE 35th Annual*, vol. 3, june 2004, pp. 2114 – 2120 Vol.3.
- [99] V. Utkin, "Sliding Mode Control of DC/DC Multiphase Power Converters," in *Power Electronics and Motion Control Conference, 2008. EPE-PEMC 2008. 13th*, Sept. 2008, pp. 512 –514.
- [100] D. Biel, E. Fossas, and R. Ramos, "Sliding mode control multiphase buck converter implementation issues," in *Variable Structure Systems (VSS), 2010 11th International Workshop on*, June 2010, pp. 434 –439.
- [101] Y. Shtessel, O. Raznopolov, and L. Ozerov, "Control of multiple modular dc-to-dc power converters in conventional and dynamic sliding surfaces," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, vol. 45, no. 10, pp. 1091 –1100, oct 1998.
- [102] M. Lopez, L. Garcia De Vicuna, M. Castilla, O. Lopez, and J. Majo, "Interleaving of parallel dc-dc converters using sliding mode control," in *Industrial Electronics Society, 1998. IECON '98. Proceedings of the 24th Annual Conference of the IEEE*, vol. 2, aug-4 sep 1998, pp. 1055 –1059 vol.2.
- [103] A.-M. Lienhardt, G. Gateau, and T. Meynard, "Digital sliding-mode observer implementation using fpga," *Industrial Electronics, IEEE Transactions on*, vol. 54, no. 4, pp. 1865 –1875, Aug. 2007.

- [104] Z. Zhou, J. Wang, and L. Wu, "Observer-Based Sliding Mode Control for a Class of Nonlinear Systems with Distributed Delays," in *Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on*, vol. 1, 2006, pp. 2051 –2055.
- [105] K. Begam, N. Yahaya, M. Awan, and H. Hassan, "Load effects analysis of synchronous rectifier buck converter using fixed pwm technique," in *Intelligent and Advanced Systems (ICIAS), 2010 International Conference on*, june 2010, pp. 1 –5.
- [106] T. Taufik, R. Prasetyo, D. Dolan, and D. Garinto, "A new multiphase interleaving buck converter with bypass cell capacitor and inductor," in *Circuits and Systems (MWS-CAS), 2010 53rd IEEE International Midwest Symposium on*, aug. 2010, pp. 1013 –1016.
- [107] Y. He and F. Luo, "Sliding-mode control for dc-dc converters with constant switching frequency," *Control Theory and Applications, IEE Proceedings -*, vol. 153, no. 1, pp. 37 – 45, jan. 2006.
- [108] L. Ge, Y.-F. Liu, and W. Wang, "The study on dc/dc converter based on fixed-frequency sliding mode control method," in *Industrial Electronics and Applications, 2009. ICIEA 2009. 4th IEEE Conference on*, may 2009, pp. 1356 –1360.

Appendices

Appendix A

Source Code for Two Phase Controller:

main.c

```

1 //#####
2 //
3 // FILE:      main.c
4 //
5 // TITLE:     For use with Buck Carrier board v1 and buck daughter boards v3+.
6 //
7 // ASSUMPTIONS:
8 //
9 //      1) Phase 1 Gate connected to GPIO0
10 //      2) Phase 2 Gate connected to GPIO2
11 //      3) Vout connected to ADC A0, A1, A2, A3
12 //      4) Current sensor connected to B0, B2
13 //
14 // DESCRIPTION:
15 //      Communicates with the computer via serial port 9600 baud, 8 data,
16 //      1 stop, no parity.
17 //
18 //      As is, the program configures SCI-A for 9600 baud with
19 //      SYSCLKOUT = 150MHz and LSPCLK = 37.5 MHz
20 //#####
21 #include "SerialHandler.h"
22 #include "ADCHandler.h"
23 #include "PWMHandler.h"
24 #include "TimerHandler.h"
25 #include "I2CHandler.h"
26 #include "Globals.h"
27 #include "controller.h"
28 #include "DSP28x_Project.h"      // Device Headerfile and Examples Include File
29
30
31 Uint16 PhaseStatus;
32 Uint16 PhaseCount;
33 union LEDS LEDStatus;
34 union LEDS LEDStatus2;
35
36 union MAINFLAGS MainFlags;
37 union BUTTONS ButtonStatus;
38
39 int32 a1,a2,a3,a4,a5, S, S1old, S2old, S1, S2, dS1, dS2, xint;
40 int32 u1,u1old1,u1old2;
41 const int32 alpha1 = 10;//100;//-2;

```

```

42 const int32 alpha2 = -2;//4;
43 const int32 alpha3 = 1;//-2;
44 const int32 alpha4 = 10;//10;
45 const int32 alpha5 = 10;//10;//-1;
46 const int32 alpha6 = 1;//1;//-1;
47 const int32 alpha7 = 2;//2;//-1;
48 const int32 alphaC0 = -200;//-10;
49 const int32 alphaC1 = -200;//-10;
50 const int16 Vref = 1160;
51 const int32 kappa = 80;
52 const int32 negkappa = -80;
53 const int32 taud1 = 10;
54 const int32 taud2 = 5;
55
56 int32 adcbuffer[32];
57
58 #pragma DATA_SECTION(DMABuf1,"DMARAML4");
59 volatile Uint16 DMABuf1[40];
60
61 volatile Uint16 *DMADest;
62 volatile Uint16 *DMASource;
63
64 void main(void)
65 {
66     Uint16 i,i2ccounter,idx;
67     int32 ramp1,ramp2;
68     int32 adcV1,adcV2,adcI1,adcI2;
69     int32 x6,x7;
70     int32* adcP;
71     Uint32 c0,c1;
72
73     char *msg;
74
75     EALLOW;
76
77     MainFlags.all = 0;
78
79     // Step 1. Initialize System Control:
80     // PLL, WatchDog, enable Peripheral Clocks
81     // This example function is found in the DSP2833x_SysCtrl.c file.
82     InitSysCtrl();
83
84     // For this example, only init the pins for the SCI-A port.
85     // This function is found in the DSP2833x_Sci.c file.
86     InitSciaGpio();
87
88     // Specific clock setting for this example:
89     EALLOW;
90     SysCtrlRegs.HISPCP.all = ADCMODCLK; // HSPCLK = SYSCLKOUT/ADC.MODCLK
91     //ADC clock now = 25MHz.
92
93     // Step 3. Clear all interrupts and initialize PIE vector table:
94     // Disable CPU interrupts
95     DINT;

```

```

96
97 // Initialize PIE control registers to their default state.
98 // The default state is all PIE interrupts disabled and flags
99 // are cleared.
100 // This function is found in the DSP2833x_PieCtrl.c file.
101     InitPieCtrl();
102
103 // Disable CPU interrupts and clear all CPU interrupt flags:
104     IER = 0x0000;
105     IFR = 0x0000;
106
107 // Initialize the PIE vector table with pointers to the shell Interrupt
108 // Service Routines (ISR).
109 // This will populate the entire table, even if the interrupt
110 // is not used in this example. This is useful for debug purposes.
111 // The shell ISR routines are found in DSP2833x_DefaultIsr.c.
112 // This function is found in DSP2833x_PieVect.c.
113     InitPieVectTable();
114
115     EALLOW;
116     SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0;
117
118     InitPwmModule();
119
120     EALLOW;
121     SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;
122
123     InitADCModule();
124
125     EINT;           // Enable Global interrupt INTM
126     ERIM;           // Enable Global realtime interrupt DBGM
127
128
129 // Step 5. User specific code:
130     PhaseStatus = 0;
131     PhaseCount = 0;
132     for (i=0; i<BUF_SIZE; i++)
133     {
134         SampleTable[i] = 0;
135     }
136
137
138
139     scia_fifo_init_async(); // Initialize the SCI FIFO
140     scia_echoback_init(); // Initialize SCI for echoback
141
142 //Kill all PWMs initially.
143     EALLOW;
144     EPwm1Regs.TZFRM.bit.OST = 1; //Kill PWM1
145     EPwm2Regs.TZFRM.bit.OST = 1; //Kill PWM2
146     EPwm3Regs.TZFRM.bit.OST = 1; //Kill PWM3
147     EPwm4Regs.TZFRM.bit.OST = 1; //Kill PWM3
148     EPwm5Regs.TZFRM.bit.OST = 1; //Kill PWM3
149

```

```

150
151     msg = "\r\nWelcome to PowerControl! \r\nHopefully this controller will
152         work!\r\n\0";
153     scia_msg(msg);
154 //Configure GPIOs
155     GpioCtrlRegs.GPAPUD.bit.GPIO0 = 0;    // Enable pull-up on GPIO0
156     GpioCtrlRegs.GPAPUD.bit.GPIO1 = 0;    // Enable pull-up on GPIO1
157     GpioCtrlRegs.GPAPUD.bit.GPIO2 = 0;    // Enable pull-up on GPIO2
158     GpioCtrlRegs.GPAPUD.bit.GPIO3 = 0;    // Enable pull-up on GPIO3
159     GpioCtrlRegs.GPAPUD.bit.GPIO4 = 0;
160     GpioCtrlRegs.GPAPUD.bit.GPIO6 = 0;
161
162     GpioCtrlRegs.GPADIR.bit.GPIO0 = 1;    // Set as output
163     GpioCtrlRegs.GPADIR.bit.GPIO1 = 1;    // Set as output
164     GpioCtrlRegs.GPADIR.bit.GPIO2 = 1;    // Set as output
165     GpioCtrlRegs.GPADIR.bit.GPIO3 = 1;    // Set as output
166     GpioCtrlRegs.GPADIR.bit.GPIO4 = 1;    // Set as output
167     GpioCtrlRegs.GPADIR.bit.GPIO6 = 1;    // Set as output
168
169
170     GpioDataRegs.GPACLEAR.all = 0x0F;
171
172     GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 0;    // Configure GPIO0 as GPIO //EPWM1A
173     GpioCtrlRegs.GPAMUX1.bit.GPIO1 = 0;    // Configure GPIO1 as GPIO //EPWM1B
174     GpioCtrlRegs.GPAMUX1.bit.GPIO2 = 0;    // Configure GPIO2 as GPIO //EPWM1A
175     GpioCtrlRegs.GPAMUX1.bit.GPIO3 = 0;    // Configure GPIO3 as GPIO //EPWM1B
176     GpioCtrlRegs.GPAMUX1.bit.GPIO4 = 0;
177     GpioCtrlRegs.GPAMUX1.bit.GPIO6 = 0;
178
179 // Start SEQ1
180     AdcRegs.ADCTRL2.all = 0x2000;
181
182     idx = 9;
183     xint = 0;
184     S1old = 0;
185     S2old = 0;
186     dS1 = 0;
187     dS2 = 0;
188     u1old1 = 0;
189     u1old2 = 0;
190     S1 = 0;
191     S2 = 0;
192     ramp1 = 0;
193     ramp2 = 100;
194     adcV1 = 0;
195     adcV2 = 0;
196     adcI1 = 0;
197     adcI2 = 0;
198     x6 = 0;
199     x7 = 0;
200     c0 = 0;
201     c1 = 0;
202     while(1) //Infinite Loop

```

```

203 {
204     if (idx++ > 7)
205         idx = 0;
206     adcP = &adcbuffer[4*idx];
207     adcV1 = AdcMirror.ADCRESULT0;
208     adcI1 = AdcMirror.ADCRESULT1;
209     adcV2 = AdcMirror.ADCRESULT2;
210     adcI2 = AdcMirror.ADCRESULT3;
211     //ADC freerunning. Latest sample will be in ADCMirror.ADCRESULTn (Use
        mirror for fastest response times)
212     u1 = Vref - (adcV1 + adcV2)/2;
213     *(adcP++) = adcV1;
214     *(adcP++) = adcI1;
215     *(adcP++) = adcV2;
216     *(adcP++) = adcI2;
217
218     a2 = alpha2*u1old1;
219     a3 = alpha3*xint;
220     xint = xint + (u1 >> 4);
221     if( xint > 0x00004000){
222         xint = 0x00004000;
223     }
224     if( xint < -0x00004000){
225         xint = -0x00004000;
226     }
227     a1 = alpha1*u1;
228
229     S = a1+a2+a3; //+a4+a5;
230     S1 = S + alpha4*ramp1 + alpha6*x6 + alphaC1*c1;
231     u1old1 = u1;
232     if(S1 > kappa){
233         GpioDataRegs.GPASET.all = 0x05; //0x05; //Phase 2 for monitoring
234         EPwm1Regs.TBCTR = 0;
235         c0 = 1;
236         ramp1 = 0;
237     }
238     if(S1 < negkappa){
239         GpioDataRegs.GPACLEAR.all = 0x05; //Phase 2 for monitoring.
240         c0 = 0;
241     }
242     S2 = S + alpha5*ramp2 + alpha7*x7 + alphaC0*c0;
243     if(S2 > kappa){
244         GpioDataRegs.GPASET.all = 0x50; //Phase 4 for monitoring
245         EPwm1Regs.TBCTR = 0;
246         c1 = 1;
247         ramp2 = 0;
248     }
249     if(S2 < negkappa){
250         GpioDataRegs.GPACLEAR.all = 0x50; //Phase 4 for monitoring.
251         c1 = 0;
252     }
253     dS1 = S1 - S1old;
254     dS2 = S2 - S2old;
255     if(dS1 <= 0)

```

```

256     x6 -= 1;
257     else
258         x6 = taud1;
259     if (dS2 <= 0)
260         x7 -= 1;
261     else
262         x7 = taud2;
263     ramp1++; // = EPwm1Regs.TBCTR;
264     ramp2++; // = EPwm2Regs.TBCTR;
265     if ( x6 < -0x04000000 ) {
266         x6 = -0x04000000;
267     }
268     if ( x7 < -0x04000000 ) {
269         x7 = -0x04000000;
270     }
271     S1old = S1;
272     S2old = S2;
273 }
274
275 }
276
277 //=====
278 // End of main.c
279 //=====

```

controller.h

```
1 #ifndef CONTROLLER_H
2 #define CONTROLLER_H
3
4 ///SetupController(): Designed to set appropriate outputs back to GPIO for SMC
   to toggle at will.
5 void SetupController();
6
7 ///TurnOffController(): Returns control of GPIO pins back to PWM.
8 void TurnOffController();
9
10 ///TurnOnPhase1(): Turns off Phase1 Low side FET then turns on phase 1 high
   set FET.
11 void TurnOnPhase1();
12
13 ///TurnOffPhase1(): Turns off phase1 high side FET then turns on phase 1
   low side fet;
14 void TurnOffPhase1();
15
16 #endif /*CONTROLLER_H*/
```

Globals.h

```

1 #ifndef GLOBALS_H_
2 #define GLOBALS_H_
3
4 #include "DSP28x_Project.h"    // Device Headerfile and Examples Include File
5
6 //extern Uint16 LoopCount;
7 //extern Uint16 ErrorCount;
8 //extern Uint16 ConversionCount;
9 //extern Uint16 XmitFlag;
10 //extern Uint16 I2CTXFlag;
11 //extern Uint16 I2CRXFlag;
12 extern Uint16 PhaseStatus;
13 extern Uint16 PhaseCount;
14 //extern Uint16 LEDStatus;
15
16 struct MAINFLAGBITS{           // bits      description
17     Uint16  XmitFlag:1;         // 0   Transmit ADC information back
18     Uint16  I2CTXFlag:1;        // 1   Transmit Something over I2C
19     Uint16  I2CRXFlag:1;        // 2   Receive something over I2C
20     Uint16  Timer0Flag:1;       // 3   Do a low priority background task on timer0
21     Uint16  ControllerFlag:1;   // 4   Controller flag, set/cleared by serial
22     // comm
23     Uint16  OldControllerFlag:1; // 5   Old controller flag, used to do edge
24     // adjustments.
25     Uint16  rsv6:1;             // 6   Reserved
26     Uint16  rsv7:1;             // 7   Reserved
27     Uint16  rsv8:1;             // 8   Reserved
28     Uint16  rsv9:1;             // 9   Reserved
29     Uint16  rsv10:1;            // 10  Reserved
30     Uint16  rsv11:1;            // 11  Reserved
31     Uint16  rsv12:1;            // 12  Reserved
32     Uint16  rsv13:1;            // 13  Reserved
33     Uint16  rsv14:1;            // 14  Reserved
34     Uint16  rsv15:1;            // 15  Reserved
35 };
36
37 union MAINFLAGS{
38     Uint16  all;
39     struct MAINFLAGBITS bit;
40 };
41
42 extern union MAINFLAGS MainFlags;
43
44 struct BUTTONBITS{             // bits      description
45     Uint16  PB1:1;              // 0   Transmit ADC information back
46     Uint16  PB2:1;              // 1   Transmit Something over I2C
47     Uint16  PB3:1;              // 2   Receive something over I2C
48     Uint16  PB4:1;              // 3   Do a low priority background task on timer0
49     Uint16  SW1_1:1;            // 4   Reserved
50     Uint16  SW1_2:1;            // 5   Reserved
51     Uint16  SW1_3:1;            // 6   Reserved

```

```

51  Uint16  SW1_4:1;          // 7      Reserved
52  Uint16  rsv8:1;          // 8      Reserved
53  Uint16  rsv9:1;          // 9      Reserved
54  Uint16  rsv10:1;         // 10     Reserved
55  Uint16  rsv11:1;         // 11     Reserved
56  Uint16  rsv12:1;         // 12     Reserved
57  Uint16  rsv13:1;         // 13     Reserved
58  Uint16  rsv14:1;         // 14     Reserved
59  Uint16  rsv15:1;         // 15     Reserved
60 };
61 struct BUTTONNIBBLE{      // bits      description
62     Uint16  PB:4;          // 0      Transmit ADC information back
63     Uint16  SW1:4;         // 4      Reserved
64     Uint16  rsv8:1;        // 8      Reserved
65     Uint16  rsv9:1;        // 9      Reserved
66     Uint16  rsv10:1;       // 10     Reserved
67     Uint16  rsv11:1;       // 11     Reserved
68     Uint16  rsv12:1;       // 12     Reserved
69     Uint16  rsv13:1;       // 13     Reserved
70     Uint16  rsv14:1;       // 14     Reserved
71     Uint16  rsv15:1;       // 15     Reserved
72 };
73 union BUTTONS{
74     Uint16    all;
75     struct BUTTONBITS bit;
76     struct BUTTONNIBBLE nib;
77 };
78
79 extern union BUTTONS ButtonStatus;
80
81 struct LEDBITS{           // bits      description
82     Uint16  LED0:2;        // 0      Transmit ADC information back
83     Uint16  LED1:2;        // 4      Reserved
84     Uint16  LED2:2;        // 8      Reserved
85     Uint16  LED3:2;        // 9      Reserved
86     Uint16  LED4:2;        // 10     Reserved
87     Uint16  LED5:2;        // 11     Reserved
88     Uint16  LED6:2;        // 12     Reserved
89     Uint16  LED7:2;        // 13     Reserved
90 };
91 union LEDS{
92     Uint16    all;
93     struct LEDBITS bit;
94 };
95
96 extern union LEDS LEDStatus;
97 extern union LEDS LEDStatus2;
98
99 #endif /*GLOBALS_H*/

```

ADCHandler.h

```

1 #ifndef ADCHANDLER_H
2 #define ADCHANDLER_H
3
4 #include "DSP28x_Project.h"    // Device Headerfile and Examples Include File
5
6 //***** ADC start parameters
7 //*****
8 #if (CPU_FRQ_150MHZ)          // Default - 150 MHz SYSCLKOUT
9     #define ADCMODCLK 0x3    // HSPCLK = SYSCLKOUT/2*ADCMODCLK2 = 150/(2*3) =
10        25.0 MHz
11 #endif
12 #if (CPU_FRQ_100MHZ)
13     #define ADCMODCLK 0x2    // HSPCLK = SYSCLKOUT/2*ADCMODCLK2 = 100/(2*2) =
14        25.0 MHz
15 #endif
16 #define ADCCKPS 0x1          // ADC module clock = HSPCLK/2*ADCCKPS = 25.0MHz
17        /(1*2) = 12.5MHz
18 // #define ADCSHCLK 0xf      // S/H width in ADC module periods
19        = 16 ADC clocks
20 #define ADCSHCLK 0x3          // S/H width in ADC module periods (val+1)
21        = 4 ADC clocks
22 #define AVG 1000              // Average sample limit
23 #define ZOFFSET 0x00          // Average Zero offset
24 #define BUF_SIZE 1024         // Sample buffer size
25 //***** End ADC start parameters
26 //*****
27 void InitADCModule();
28
29 interrupt void adc_isr(void);
30
31 //ADC Variables
32 extern Uint16 SampleTable[BUF_SIZE];
33
34 #endif /*ADCHANDLER_H*/

```

ADCHandler.c

```

1 #include "ADCHandler.h"
2 #include "controller.h"
3 #include "DSP28x_Project.h"    // Device Headerfile and Examples Include File
4
5
6
7 //ADC Variables
8 Uint16 SampleTable[BUF_SIZE];
9 //const int16 alpha1 = 10;
10 //const int16 alpha2 = 726;
11 //const int16 alpha3 = 1;
12 //const int16 kappa = 45;
13 //const int16 Vref = 1160;
14
15 interrupt void adc_isr(void)
16 {
17
18 // static Uint16 idx = 16;
19 // static int32 S = 0;
20 // static int16 adcVold = 0;
21 // int16 adcV;
22 // int16 x1,x2;
23 // idx++;
24 // adcV = AdcRegs.ADCRESULT0 >>4;
25 //
26 // x1 = Vref - adcV;
27 // x2 = adcV - adcVold;
28 // S = alpha1*x1 + alpha2*x2+alpha3*adcVold;
29 // if (S > kappa){
30 // if ((idx % 1)==0){
31 // TurnOnPhase1();
32 // }
33 // if (S < kappa){
34 // else{
35 // TurnOffPhase1();
36 // }
37 // SampleTable[idx+4] = AdcRegs.ADCRESULT1 >>4;
38 // SampleTable[idx++] = adcV;
39 // adcVold = adcV;
40 // if (idx > 19){
41 // idx = 16;
42 // SampleTable[0] = (SampleTable[16] + SampleTable[17] + SampleTable
43 // [18] + SampleTable[19]) >> 2;
44 // SampleTable[1] = (SampleTable[20] + SampleTable[21] + SampleTable
45 // [22] + SampleTable[23]) >> 2;
46 // }
47 // Voltage2[0] = AdcRegs.ADCRESULT1; //>>4;
48 //
49 // // If 40 conversions have been logged, start over
50 // if (ConversionCount == 0x1FD) //two seconds
51 // {
52 // ConversionCount = 0;

```

```

51  //// XmitFlag = 1;
52  //// }
53  //// else ConversionCount++;
54  ////
55  // Reinitialize for next ADC sequence
56  AdcRegs.ADCTRL2.bit.RST_SEQ1 = 1;           // Reset SEQ1
57  AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1;         // Clear INT SEQ1 bit
58  PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;     // Acknowledge interrupt to PIE
59
60  return;
61 }
62
63
64 void InitADCModule() {
65     InitAdc();
66
67     // Specific ADC setup for this example:
68     // AdcRegs.ADCTRL1.bit.ACQ_PS = ADC_SHCLK;
69     AdcRegs.ADCTRL1.bit.ACQ_PS = 0;           // One clock sample&hold
70     AdcRegs.ADCTRL1.bit.CPS = 0;              // Disable an additional prescaler.
71     AdcRegs.ADCTRL3.bit.ADCCLKPS = 0; //ADC_CLKPS; trying for 12.5 MHz sample
       rate.
72     AdcRegs.ADCTRL3.bit.SMODE_SEL = 1;        // Simultaneous sampling mode
73     AdcRegs.ADCTRL1.bit.SEQ_CASC = 1;         // 1 Cascaded mode
74     AdcRegs.ADCTRL1.bit.CONT_RUN = 1;         // Setup continuous run
75
76     AdcRegs.ADCMAXCONV.all = 0x0001;          // Setup all conv's on SEQ1,
       only 1 conversion (CONV00)
77     AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0x0;    // Sample Phase 1 Vout/IOut
       store in ADCRESULT0, ADCRESULT1
78     AdcRegs.ADCCHSELSEQ1.bit.CONV01 = 0x4;    // Sample Phase 2 Vout/IOut
       store in ADCRESULT2, ADCRESULT3
79
80     // AdcRegs.ADCTRL2.bit.EPWM_SOCA_SEQ1 = 1; // Enable SOCA from ePWM to
       start SEQ1
81     // AdcRegs.ADCTRL2.bit.INT_ENA_SEQ1 = 1; // Enable SEQ1 interrupt (
       every EOS)
82
83
84 }

```

SerialHandler.h

```

1 #ifndef SERIALHANDLER_H
2 #define SERIALHANDLER_H
3
4 #include "DSP28x_Project.h"    // Device Headerfile and Examples Include File
5
6 #define TXBUFFSIZE 128
7 //Union and structure definitions
8 struct SCIISRBITS{           // bits      description
9     Uint16   TXFlag:1;       // 0        TX Interrupt Occured
10    Uint16   RXFlag:1;       // 1        RX Interrupt Occured
11    Uint16   unassigned:14;
12 };
13
14 union SCIISR_FLAGS{
15     Uint16   all;
16     struct SCIISRBITS bit;
17 };
18
19 struct SCIFLAGBITS{           // bits      description
20     Uint16   UpdateDuty:1;    // 0        Update Duty Cycle Flag
21     Uint16   UpdateHRDuty:1;  // 1        Update HR Duty cycle flag
22     Uint16   UpdateDBRising:1; // 2        Update Deadband rising flag
23     Uint16   UpdateDBFalling:1; // 3        Update deadband falling flag
24     Uint16   UpdatePeriod:1;  // 4        Update Period
25     Uint16   KillPWM:1;       // 5        Kill PWM
26     Uint16   UpdatePhase2:1;  // 6        Update Phase 2
27     Uint16   UpdatePhase3:1;  // 7        Update Phase 3
28     Uint16   UpdatePhase4:1;  // 8        Update Phase 4
29     Uint16   UpdatePhase5:1;  // 9        Update Phase 5
30     Uint16   SendEverything:1; // 10       Send Complete Status
31     Uint16   UpdateController:1; // 11      Turn on/off the controller
32     Uint16   unassigned:4;
33 };
34
35 union SCIFLAGS{
36     Uint16   all;
37     struct SCIFLAGBITS bit;
38 };
39
40
41 //Global Scope variables
42 extern union SCIISR_FLAGS SciIsrFlags;
43 extern union SCIFLAGS SciFlags;
44
45 extern Uint16 ReceivedChar;
46 //extern Uint16 XmitFlag;
47
48 //Async TXBuffer FIFO;
49 extern Uint16 TXBuffer[TXBUFFSIZE];
50 extern Uint16* TXBufferFIFOFront;
51 extern Uint16* TXBufferFIFOEnd;
52 extern Uint16* TXBufferEnd;

```

```
53 |
54 | interrupt void scirxa_isr(void);
55 | interrupt void scitxa_isr(void);
56 |
57 | void scia_echoback_init(void);
58 | void scia_fifo_init(void);
59 | void scia_xmit(int a);
60 | void scia_msg(char *msg);
61 | void scia_xmit_num(Uint16 val);
62 |
63 | void scia_fifo_init_async(void);
64 | void scia_xmit_async(int a);
65 | void scia_msg_async(char *msg);
66 | void scia_xmit_num_async(Uint16 val);
67 |
68 |
69 |
70 | #endif /*SERIALHANDLER_H*/
```

SerialHandler.c

```

1 #include "SerialHandler.h"
2 #include "PWMHandler.h"
3 #include "Globals.h"
4 #include "DSP28x_Project.h"           // Device Headerfile and Examples Include File
5
6 union SCLISR_FLAGS SciIsrFlags;
7 union SCIFLAGS SciFlags;
8
9 Uint16 ReceivedChar;
10
11 Uint16 TXBuffer[TXBUFFSIZE];
12 Uint16* TXBufferFIFOFront;
13 Uint16* TXBufferFIFOEnd;
14 Uint16* TXBufferEnd;
15
16 interrupt void scirxa_isr(void){
17 // char* msg;
18 static Uint16 value;
19 // static Uint16 phaseValue = 0;
20 ALLOW;
21 ReceivedChar = SciaRegs.SCIRXBUF.all;
22 switch(ReceivedChar){
23 case 'x':
24     EPwm1Regs.TZFRC.bit.OST = 1; // Kill PWM1
25     EPwm2Regs.TZFRC.bit.OST = 1; // Kill PWM2
26     EPwm3Regs.TZFRC.bit.OST = 1; // Kill PWM3
27     EPwm4Regs.TZFRC.bit.OST = 1; // Kill PWM4
28     EPwm5Regs.TZFRC.bit.OST = 1; // Kill PWM5
29     EPwm6Regs.TZFRC.bit.OST = 1; // Kill PWM6
30     SciFlags.bit.KillPWM = 1;
31     break;
32 case 'i':
33     value = EPwm1Regs.CMPA.half.CMPA + 1;
34     SciFlags.bit.UpdateDuty = 1;
35     break;
36 case 'k':
37     value = EPwm1Regs.CMPA.half.CMPA - 1;
38     SciFlags.bit.UpdateDuty = 1;
39     break;
40 case '.':
41     value = EPwm1Regs.CMPA.half.CMPAHR + 0x100;
42     SciFlags.bit.UpdateHRDuty = 1;
43     break;
44 case ',':
45     value = EPwm1Regs.CMPA.half.CMPAHR - 0x100;
46     SciFlags.bit.UpdateHRDuty = 1;
47     break;
48 case 'o':
49     value = EPwm1Regs.DBRED + 1;
50     SciFlags.bit.UpdateDBRising = 1;
51     break;
52 case 'l':

```

```

53     value = EPwm1Regs.DBRED - 1;
54     SciFlags.bit.UpdateDBRising = 1;
55     break;
56 case 'p':
57     value = EPwm1Regs.DBFED + 1;
58     SciFlags.bit.UpdateDBFalling = 1;
59     break;
60 case ',':
61     value = EPwm1Regs.DBFED - 1;
62     SciFlags.bit.UpdateDBFalling = 1;
63     break;
64 case 'u':
65     value = EPwm1Regs.TBPRD + 1;
66     SciFlags.bit.UpdatePeriod = 1;
67     break;
68 case 'j':
69     value = EPwm1Regs.TBPRD - 1;
70     SciFlags.bit.UpdatePeriod = 1;
71     break;
72 case 'y':
73     value = EPwm2Regs.TBPHS.half.TBPHS + 1;
74     SciFlags.bit.UpdatePhase2 = 1;
75     break;
76 case 'h':
77     value = EPwm2Regs.TBPHS.half.TBPHS - 1;
78     SciFlags.bit.UpdatePhase2 = 1;
79     break;
80 case 't':
81     value = EPwm3Regs.TBPHS.half.TBPHS + 1;
82     SciFlags.bit.UpdatePhase3 = 1;
83     break;
84 case 'g':
85     value = EPwm3Regs.TBPHS.half.TBPHS - 1;
86     SciFlags.bit.UpdatePhase3 = 1;
87     break;
88 case 'r':
89     value = EPwm4Regs.TBPHS.half.TBPHS + 1;
90     SciFlags.bit.UpdatePhase4 = 1;
91     break;
92 case 'f':
93     value = EPwm4Regs.TBPHS.half.TBPHS - 1;
94     SciFlags.bit.UpdatePhase4 = 1;
95     break;
96 case 'e':
97     value = EPwm5Regs.TBPHS.half.TBPHS + 1;
98     SciFlags.bit.UpdatePhase5 = 1;
99     break;
100 case 'd':
101     value = EPwm5Regs.TBPHS.half.TBPHS - 1;
102     SciFlags.bit.UpdatePhase5 = 1;
103     break;
104 case '1':
105     EPwm1Regs.TZCLR.all = 5;
106     PhaseCount += 1;

```

```

107     break;
108     case '2':
109         EPwm2Regs.TZCLR.all = 5;
110         PhaseCount += 1;
111         break;
112     case '3':
113         EPwm3Regs.TZCLR.all = 5;
114         PhaseCount += 1;
115         break;
116     case '4':
117         EPwm4Regs.TZCLR.all = 5;
118         PhaseCount += 1;
119         break;
120     case '5': //Phase 5
121         EPwm5Regs.TZCLR.all = 5;
122         PhaseCount += 1;
123         break;
124     case 'a': //ADC
125         MainFlags.bit.XmitFlag = 1;
126         break;
127     case 's': //status
128         SciFlags.bit.SendEverything = 1;
129         break;
130     case 'n':
131         if(--PwmPointer < PwmPeriods)
132             PwmPointer = PwmPeriods + 13;
133         value = *PwmPointer;
134         SciFlags.bit.UpdatePeriod = 1;
135         break;
136     case 'm':
137         if(++PwmPointer > PwmPeriods+13)
138             PwmPointer = PwmPeriods;
139         value = *PwmPointer;
140         SciFlags.bit.UpdatePeriod = 1;
141         break;
142     case 'c':
143         MainFlags.bit.I2CTXFlag = 1;
144         break;
145     case 'v':
146         MainFlags.bit.I2CRXFlag = 1;
147         break;
148     case 'Z':
149         SciFlags.bit.UpdateController = 1;
150         break;
151     // default:
152     //     scia_xmit_async(ReceivedChar);
153     //     break;
154 }
155 scia_xmit_async(ReceivedChar);
156 if(SciFlags.bit.KillPWM == 1){
157     //     scia_xmit_async('x');
158     PhaseCount = 0;
159     SciFlags.bit.KillPWM = 0;
160 }

```

```

161  if( SciFlags.bit.UpdateDuty == 1){
162      EPwm1Regs.CMPA.half.CMPA = value;
163      EPwm2Regs.CMPA.half.CMPA = value;
164      EPwm3Regs.CMPA.half.CMPA = value;
165      EPwm4Regs.CMPA.half.CMPA = value;
166      EPwm5Regs.CMPA.half.CMPA = value;
167  //      EPwm2Regs.TBPHS.half.TBPHS = (value>>1);
168      scia_xmit_num_async(value);
169      SciFlags.bit.UpdateDuty = 0;
170  }
171  if( SciFlags.bit.UpdateHRDuty == 1){
172      EPwm1Regs.CMPA.half.CMPAHR = value;
173      EPwm2Regs.CMPA.half.CMPAHR = value;
174      EPwm3Regs.CMPA.half.CMPAHR = value;
175      EPwm4Regs.CMPA.half.CMPAHR = value;
176      EPwm5Regs.CMPA.half.CMPAHR = value;
177      scia_xmit_num_async(value);
178      SciFlags.bit.UpdateHRDuty = 0;
179  }
180  if( SciFlags.bit.UpdateDBRising == 1){
181      EPwm1Regs.DBRED = value;
182      EPwm2Regs.DBRED = value;
183      EPwm3Regs.DBRED = value;
184      EPwm4Regs.DBRED = value;
185      EPwm5Regs.DBRED = value;
186  //      scia_xmit_async('o');
187  //      scia_xmit_async('l');
188      scia_xmit_num_async(value);
189      SciFlags.bit.UpdateDBRising = 0;
190  }
191  if( SciFlags.bit.UpdateDBFalling == 1){
192      EPwm1Regs.DBFED = value;
193      EPwm2Regs.DBFED = value;
194      EPwm3Regs.DBFED = value;
195      EPwm4Regs.DBFED = value;
196      EPwm5Regs.DBFED = value;
197  //      scia_xmit_async('p');
198  //      scia_xmit_async(';');
199      scia_xmit_num_async(value);
200      SciFlags.bit.UpdateDBFalling = 0;
201  }
202  if( SciFlags.bit.UpdatePeriod == 1){
203      EPwm1Regs.TBPRD = value;
204      EPwm2Regs.TBPRD = value;
205      EPwm3Regs.TBPRD = value;
206      EPwm4Regs.TBPRD = value;
207      EPwm5Regs.TBPRD = value;
208      scia_xmit_num_async(value);
209  //FLXME: Need to autmatically update periods if the phases are active.
210  //      phaseValue = (~PhaseStatus) & 0x1F);
211  //      value = 0;
212  //
213  //      EPwm1Regs.TBPHS.half.TBPHS = 0;
214  //      if(PhaseCount > 1){

```

```

215 //      value = (EPwm1Regs.TBPRD / PhaseCount);
216 //      if (phaseValue & 0x02)
217 //          EPwm2Regs.TBPHS.half.TBPHS = value;
218 //      else
219 //          EPwm2Regs.TBPHS.half.TBPHS = 0;
220 //      if (phaseValue & 0x04)
221 //          EPwm3Regs.TBPHS.half.TBPHS = value;
222 //      else
223 //          EPwm3Regs.TBPHS.half.TBPHS = 0;
224 //      if (phaseValue & 0x08)
225 //          EPwm4Regs.TBPHS.half.TBPHS = value;
226 //      else
227 //          EPwm4Regs.TBPHS.half.TBPHS = 0;
228 //      if (phaseValue & 0x10)
229 //          EPwm5Regs.TBPHS.half.TBPHS = value;
230 //      else
231 //          EPwm5Regs.TBPHS.half.TBPHS = 0;
232 //  }
233
234     SciFlags.bit.UpdatePeriod = 0;
235 }
236 if (SciFlags.bit.UpdatePhase2 == 1){
237     EPwm2Regs.TBPHS.half.TBPHS = value;
238 //     scia_xmit_async('y');
239 //     scia_xmit_async('h');
240     scia_xmit_num_async(value);
241     SciFlags.bit.UpdatePhase2 = 0;
242 }
243 if (SciFlags.bit.UpdatePhase3 == 1){
244     EPwm3Regs.TBPHS.half.TBPHS = value;
245 //     scia_xmit_async('t');
246 //     scia_xmit_async('g');
247     scia_xmit_num_async(value);
248     SciFlags.bit.UpdatePhase3 = 0;
249 }
250 if (SciFlags.bit.UpdatePhase4 == 1){
251     EPwm4Regs.TBPHS.half.TBPHS = value;
252 //     scia_xmit_async('r');
253 //     scia_xmit_async('f');
254     scia_xmit_num_async(value);
255     SciFlags.bit.UpdatePhase4 = 0;
256 }
257 if (SciFlags.bit.UpdatePhase5 == 1){
258     EPwm5Regs.TBPHS.half.TBPHS = value;
259 //     scia_xmit_async('e');
260 //     scia_xmit_async('d');
261     scia_xmit_num_async(value);
262     SciFlags.bit.UpdatePhase5 = 0;
263 }
264 if (SciFlags.bit.SendEverything == 1){
265     scia_xmit_num_async(EPwm1Regs.TBPRD);
266     scia_xmit_num_async(EPwm1Regs.CMPA.half.CMPA);
267     scia_xmit_num_async(EPwm1Regs.DBRED);
268     scia_xmit_num_async(EPwm1Regs.DBFED);

```

```

269     scia_xmit_num_async(EPwm1Regs.TZFRC.all);
270     scia_xmit_num_async(EPwm2Regs.TZFRC.all);
271     scia_xmit_num_async(EPwm2Regs.TBPHS.half.TBPHS);
272     scia_xmit_num_async(EPwm3Regs.TZFRC.all);
273     scia_xmit_num_async(EPwm3Regs.TBPHS.half.TBPHS);
274     scia_xmit_num_async(EPwm4Regs.TZFRC.all);
275     scia_xmit_num_async(EPwm4Regs.TBPHS.half.TBPHS);
276     scia_xmit_num_async(EPwm5Regs.TZFRC.all);
277     scia_xmit_num_async(EPwm5Regs.TBPHS.half.TBPHS);
278     SciFlags.bit.SendEverything = 0;
279 }
280 if(SciFlags.bit.UpdateController == 1){
281     if(MainFlags.bit.ControllerFlag == 1){
282         //Controller on, turn it off.
283         MainFlags.bit.ControllerFlag = 0;
284     }
285     else{
286         //Controller off, turn it on.
287         MainFlags.bit.ControllerFlag = 1;
288     }
289 }
290 //     msg = "\r\n";
291 //     scia_msg_async(msg);
292     SciIsrFlags.bit.RXFlag = 1;
293     SciaRegs.SCIFFRX.bit.RXFFINTCLR = 1; // Reset interrupt.
294     PieCtrlRegs.PIEACK.all = PIEACK_GROUP9; // Acknowledge interrupt to PIE
295 }
296
297
298
299 // Test 1, SCIA DLB, 8-bit word, baud rate 0x000F, default, 1 STOP bit, no
    parity
300 void scia_echoback_init()
301 {
302     // Note: Clocks were turned on to the SCIA peripheral
303     // in the InitSysCtrl() function
304     SciFlags.all = 0x00;
305
306     SciaRegs.SCICCR.all = 0x0007; // 1 stop bit, No loopback
307                                   // No parity, 8 char bits,
308                                   // async mode, idle-line protocol
309     SciaRegs.SCICTL1.all = 0x0003; // enable TX, RX, internal SCICLK,
310                                   // Disable RX ERR, SLEEP, TXWAKE
311     SciaRegs.SCICTL2.all = 0x0003;
312     SciaRegs.SCICTL2.bit.TXINTENA = 1;
313 //     SciaRegs.SCICTL2.bit.RXBKINTENA = 1;
314     SciaRegs.SCIFFRX.bit.RXFFIENA = 1;
315
316     #if (CPU_FRQ_150MHZ)
317         SciaRegs.SCHBAUD = 0x0001; // 9600 baud @LSPCLK = 37.5MHz.
318         SciaRegs.SCILBAUD = 0x00E7;
319     #endif
320     #if (CPU_FRQ_100MHZ)
321         SciaRegs.SCHBAUD = 0x0001; // 9600 baud @LSPCLK = 20MHz.

```

```

322     SciaRegs.SCILBAUD      =0x0044;
323 #endif
324     SciaRegs.SCICTL1.all =0x0023;  // Relinquish SCI from Reset
325 }
326
327 // Transmit a character from the SCI
328 void scia_xmit(int a)
329 {
330     while (SciaRegs.SCIFFTX.bit.TXFFST != 0) {}
331     SciaRegs.SCITXBUF=a;
332
333 }
334
335 void scia_msg(char * msg)
336 {
337     int i;
338     i = 0;
339     while(msg[i] != '\0')
340     {
341         scia_xmit(msg[i]);
342         i++;
343     }
344 }
345
346 // Initalize the SCI FIFO
347 void scia_fifo_init()
348 {
349     SciaRegs.SCIFFTX.all=0xE040;
350     // SciaRegs.SCIFFRX.all=0x204f;
351     SciaRegs.SCIFFRX.all=0x2061; //1 byte fifo generates interrupt
352     // SciaRegs.SCIFFRX.all=0x2071;
353     SciaRegs.SCIFFCT.all=0x0;
354
355 }
356
357 void scia_xmit_num(Uint16 val){
358     int nibble = val & 0x000F;
359     int ascii[4];
360     int counter = 0;
361     for(counter = 0; counter < 4; counter++){
362         if(nibble > 9)
363             ascii[3-counter] = nibble+55;
364         else
365             ascii[3-counter] = nibble+48;
366         val = val >> 4;
367         nibble = val & 0x000F;
368     }
369     scia_xmit(ascii[0]);
370     scia_xmit(ascii[1]);
371     scia_xmit(ascii[2]);
372     scia_xmit(ascii[3]);
373 }
374
375

```

```

376 interrupt void scitxa_isr(void){
377     Uint16 i;
378     //Interrupt occured, meaning TX FIFO empty
379     //Check if TXBuffer is empty. If so, disable interrupts and finish out.
380     if(TXBufferFIFOFront == TXBufferFIFOEnd){
381         SciaRegs.SCIFFTX.bit.TXFFIENA = 0; //disable interrupts
382     }
383     else{ //Data in TXBuffer
384         for(i = 0; i < 15; i++)
385         {
386             SciaRegs.SCITXBUF=*TXBufferFIFOFront;
387             TXBufferFIFOFront++;
388             if(TXBufferFIFOFront > TXBufferEnd)
389                 TXBufferFIFOFront = TXBuffer;
390             if(TXBufferFIFOFront == TXBufferFIFOEnd)
391                 break;
392         }
393     }
394     SciIsrFlags.bit.TXFlag = 1;
395     SciaRegs.SCIFFTX.bit.TXFFINTCLR = 1; // Reset interrupt.
396     PieCtrlRegs.PIEACK.all = PIEACK_GROUP9; // Acknowledge interrupt to PIE
397 }
398
399
400 // Initalize the SCI FIFO
401 void scia_fifo_init_async()
402 {
403     SciaRegs.SCIFFTX.all=0xE040; //Empty TX generates interrupt, but don't
404         enable the interrupt yet.
405     SciaRegs.SCIFFRX.all=0x2061; //1 byte fifo generates interrupt
406     SciaRegs.SCIFFCT.all=0x0;
407     TXBufferFIFOFront = TXBuffer;
408     TXBufferFIFOEnd = TXBuffer;
409     TXBufferEnd = TXBuffer + (TXBUFFSIZE-1)*sizeof(Uint16); // the -1 is there
410         because TXBuffer counts.
411 }
412
413 // Transmit a character from the SCI
414 void scia_xmit_async(int a)
415 {
416     //while (SciaRegs.SCIFFTX.bit.TXFFST != 0) {}
417     *TXBufferFIFOEnd++ = a; //Store a in TXBufferFIFOEnd and then increment
418         TXBufferFIFOEnd
419     // TXBufferFIFOEnd++;
420     if(TXBufferFIFOEnd > TXBufferEnd)
421         TXBufferFIFOEnd = TXBuffer;
422     // SciaRegs.SCITXBUF=a;
423     SciaRegs.SCIFFTX.bit.TXFFIENA = 1;
424 }
425
426 void scia_msg_async(char * msg)
427 {
428     int i;
429     i = 0;

```

```

427     while(msg[i] != '\0')
428     {
429         scia_xmit_async(msg[i]);
430         i++;
431     }
432 }
433
434
435
436 void scia_xmit_num_async(Uint16 val){
437     int nibble = val & 0x000F;
438     int ascii[4];
439     int counter = 0;
440     for(counter = 0; counter < 4; counter++){
441         if(nibble > 9)
442             ascii[3-counter] = nibble+55;
443         else
444             ascii[3-counter] = nibble+48;
445         val = val >> 4;
446         nibble = val & 0x000F;
447     }
448     scia_xmit_async(ascii[0]);
449     scia_xmit_async(ascii[1]);
450     scia_xmit_async(ascii[2]);
451     scia_xmit_async(ascii[3]);
452 }

```

TimerHandler.h

```
1 #ifndef TIMERHANDLER_H
2 #define TIMERHANDLER_H
3
4 #include "Globals.h"
5 #include "DSP28x_Project.h"    // Device Headerfile and Examples Include File
6
7
8 void InitTimerModule(void);
9 void Timer0Callback(void);
10
11 interrupt void timer0_isr();
12
13 #endif /*TIMERHANDLER_H*/
```

TimerHandler.c

```

1 #include "TimerHandler.h"
2 #include "I2CHandler.h"
3 #include "Globals.h"
4
5 Uint16 timerCounter;
6
7 void InitTimerModule(){
8     CpuTimer0Regs.TCR.bit.TSS = 1; //Stop timer.
9     CpuTimer0Regs.TCR.bit.TIF = 1; //Clear any triggered interrupts.
10    CpuTimer0Regs.TPR.all = 0x0000;
11    CpuTimer0Regs.TPRH.all = 0x0000;
12    // CpuTimer0Regs.PRD.half.MSW = 0x0907; //Setup timer for 1 second at 150MHz.
13    // CpuTimer0Regs.PRD.half.LSW = 0xF00F;
14    // CpuTimer0Regs.PRD.half.MSW = 0x0241; //Setup timer for 0.25 second at 150
15    MHz.
16    // CpuTimer0Regs.PRD.half.LSW = 0xFC03;
17    CpuTimer0Regs.PRD.half.MSW = 0x002D; //Setup timer for 20ms second at 150MHz
18    CpuTimer0Regs.PRD.half.LSW = 0xFE02;
19    CpuTimer0Regs.TCR.all = 0xC020; //Start the timer back and reload the period
20    into the timer.
21    //Enable the interrupt.
22    //Timers count down from the period value.
23    timerCounter = 0;
24 }
25 interrupt void timer0_isr(){
26
27     // if(timerCounter == 0)
28     // {
29     //     EPwm2Regs.CMPA.half.CMPA = (EPwm2Regs.CMPA.half.CMPA>>1);
30     //     timerCounter++;
31     // }
32     // else
33     // {
34     //     EPwm2Regs.CMPA.half.CMPA = (EPwm2Regs.CMPA.half.CMPA<<1);
35     //     timerCounter = 0;
36     // }
37     MainFlags.bit.Timer0Flag = 1;
38     CpuTimer0Regs.TCR.bit.TIF = 1; //Clear timer interrupt.
39     PieCtrlRegs.PIEACK.all = PIEACK.GROUP1; // Acknowledge interrupt to PIE
40 }
41
42 void Timer0Callback(){
43     // static Uint16 counter = 0;
44     Uint16 tmp = 0;
45     LEDStatus.all = 0;
46     LEDStatus2.all = 0;
47     ButtonStatus.all = PollButtons();
48     if(ButtonStatus.nib.PB < 0xF){
49         EPwm1Regs.TZFRC.bit.OST = 1; //Kill PWM1

```

```

50     EPwm2Regs.TZFRC.bit.OST = 1; //Kill PWM2
51     EPwm3Regs.TZFRC.bit.OST = 1; //Kill PWM3
52     EPwm4Regs.TZFRC.bit.OST = 1; //Kill PWM4
53     EPwm5Regs.TZFRC.bit.OST = 1; //Kill PWM5
54     EPwm6Regs.TZFRC.bit.OST = 1; //Kill PWM6
55 }
56 if(ButtonStatus.bit.PB1 == 0)
57     LEDStatus2.bit.LED0 = LCD_ON;
58 if(ButtonStatus.bit.PB2 == 0)
59     LEDStatus2.bit.LED1 = LCD_ON;
60 if(ButtonStatus.bit.PB3 == 0)
61     LEDStatus2.bit.LED2 = LCD_ON;
62 if(ButtonStatus.bit.PB4 == 0)
63     LEDStatus2.bit.LED3 = LCD_ON;
64 if(ButtonStatus.bit.SW1_1 == 0)
65     LEDStatus2.bit.LED4 = LCD_ON;
66 if(ButtonStatus.bit.SW1_2 == 0)
67     LEDStatus2.bit.LED5 = LCD_ON;
68 if(ButtonStatus.bit.SW1_3 == 0)
69     LEDStatus2.bit.LED6 = LCD_ON;
70 if(ButtonStatus.bit.SW1_4 == 0)
71     LEDStatus2.bit.LED7 = LCD_ON;
72 SetPCA3Status(LEDStatus2.all);
73 PhaseStatus = PollPhases();
74 if((PhaseStatus & 0x0001) == 0){ //Phase 1 Board plugged in.
75     if(EPwm1Regs.TZFLG.bit.OST == 1){ //channel off
76         LEDStatus.bit.LED0 = LCD_BLINK1;
77     }
78     else{
79         LEDStatus.bit.LED0 = LCD_ON;
80         tmp += 1;
81     }
82 }
83
84 if((PhaseStatus & 0x0002) == 0){ // Phase 2 Board plugged in.
85     if(EPwm2Regs.TZFLG.bit.OST == 1){ //channel off
86         LEDStatus.bit.LED1 = LCD_BLINK1;
87     }
88     else{
89         LEDStatus.bit.LED1 = LCD_ON;
90         tmp += 1;
91     }
92 }
93
94 if((PhaseStatus & 0x0004) == 0){ // Phase 3 Board plugged in.
95     if(EPwm3Regs.TZFLG.bit.OST == 1){ //channel off
96         LEDStatus.bit.LED2 = LCD_BLINK1;
97     }
98     else{
99         LEDStatus.bit.LED2 = LCD_ON;
100         tmp += 1;
101     }
102 }
103

```

```

104  if((PhaseStatus & 0x0008) == 0){ // Phase 4 Board plugged in.
105      if(EPwm4Regs.TZFLG.bit.OST == 1){ //channel off
106          LEDStatus.bit.LED3 = LCD_BLINK1;
107      }
108      else{
109          LEDStatus.bit.LED3 = LCD_ON;
110          tmp += 1;
111      }
112  }
113
114  if((PhaseStatus & 0x0010) == 0){ // Phase 5 Board plugged in.
115      if(EPwm5Regs.TZFLG.bit.OST == 1){ //channel off
116          LEDStatus.bit.LED4 = LCD_BLINK1;
117      }
118      else{
119          LEDStatus.bit.LED4 = LCD_ON;
120          tmp += 1;
121      }
122  }
123  PhaseCount = tmp;
124  // if(counter == 0){
125  //     tmp2 = LCD_BLINK2
126  //     LEDStatus += tmp2 <<10;
127  //     counter++;
128  // }
129  // else{
130  //     tmp2 = LCD_OFF
131  //     LEDStatus += tmp2 <<10;
132  //     counter = 0;
133  // }
134  // tmp += 0xAFA0;
135  SetPCA1Status(LEDStatus.all);
136
137
138 }

```

PWMHandler.h

```

1 #ifndef PWMHANDLER_H
2 #define PWMHANDLER_H
3
4 #include "DSP28x_Project.h"    // Device Headerfile and Examples Include File
5
6 //***** PWM Parameters
7
8 //Deadband definitions
9 #define EPWM_MAX_DB    0x03FF
10 #define EPWM_MIN_DB    0x0001
11
12 #define DB_UP    1
13 #define DB_DOWN 0
14
15 //***** End PWM Parameters
16
17 //EPWM Variables
18 //extern Uint32  EPwm1TimerIntCount;
19 //extern Uint32  EPwm2TimerIntCount;
20 //extern Uint32  EPwm3TimerIntCount;
21 //extern Uint16  EPwm1_DB_Direction;
22 //extern Uint16  EPwm2_DB_Direction;
23 //extern Uint16  EPwm3_DB_Direction;
24
25 extern Uint16 PwmPeriods[14];
26 extern Uint16 *PwmPointer;
27
28 void InitPwmModule();
29
30 //Pwm init functions
31 void EPwm1Init(void);
32 void EPwm2Init(void);
33 void EPwm3Init(void);
34 void EPwm4Init(void);
35 void EPwm5Init(void);
36 void EPwm6Init(void);
37
38
39 interrupt void epwm1_isr(void);
40 interrupt void epwm2_isr(void);
41 interrupt void epwm3_isr(void);
42
43 #endif /*PWMHANDLER_H*/

```

PWMHandler.c

```

1 #include "PWMHandler.h"
2
3 //EPWM Variables
4 //Uint32  EPwm1TimerIntCount;
5 //Uint32  EPwm2TimerIntCount;
6 //Uint32  EPwm3TimerIntCount;
7 //Uint16  EPwm1_DB_Direction;
8 //Uint16  EPwm2_DB_Direction;
9 //Uint16  EPwm3_DB_Direction;
10
11 Uint16 PwmPeriods[14];
12 Uint16 *PwmPointer;
13
14
15 //
16 //interrupt void epwm1_isr(void)
17 //{
18 //    if (EPwm1_DB_Direction == DB_UP)
19 //    {
20 //        if (EPwm1Regs.DBFED < EPWM1_MAX_DB)
21 //        {
22 //            EPwm1Regs.DBFED++;
23 //            EPwm1Regs.DBRED++;
24 //        }
25 //        else
26 //        {
27 //            EPwm1_DB_Direction = DB_DOWN;
28 //            EPwm1Regs.DBFED--;
29 //            EPwm1Regs.DBRED--;
30 //        }
31 //    }
32 //    else
33 //    {
34 //        if (EPwm1Regs.DBFED == EPWM1_MIN_DB)
35 //        {
36 //            EPwm1_DB_Direction = DB_UP;
37 //            EPwm1Regs.DBFED++;
38 //            EPwm1Regs.DBRED++;
39 //        }
40 //        else
41 //        {
42 //            EPwm1Regs.DBFED--;
43 //            EPwm1Regs.DBRED--;
44 //        }
45 //    }
46 //    EPwm1TimerIntCount++;
47 //
48 //    // Clear INT flag for this timer
49 //    EPwm1Regs.ETCLR.bit.INT = 1;
50 //
51 //    // Acknowledge this interrupt to receive more interrupts from group 3
52 //    PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;

```

```

53 //
54 //}
55 //
56 //interrupt void epwm2_isr(void)
57 //{
58 //
59 //    if (EPwm2_DB_Direction == DB_UP)
60 //    {
61 //        if (EPwm2Regs.DBFED < EPWM2_MAX_DB)
62 //        {
63 //            EPwm2Regs.DBFED++;
64 //            EPwm2Regs.DBRED++;
65 //        }
66 //        else
67 //        {
68 //            EPwm2_DB_Direction = DB_DOWN;
69 //            EPwm2Regs.DBFED--;
70 //            EPwm2Regs.DBRED--;
71 //        }
72 //    }
73 //    else
74 //    {
75 //        if (EPwm2Regs.DBFED == EPWM2_MIN_DB)
76 //        {
77 //            EPwm2_DB_Direction = DB_UP;
78 //            EPwm2Regs.DBFED++;
79 //            EPwm2Regs.DBRED++;
80 //        }
81 //        else
82 //        {
83 //            EPwm2Regs.DBFED--;
84 //            EPwm2Regs.DBRED--;
85 //        }
86 //    }
87 //
88 //    EPwm2TimerIntCount++;
89 //
90 //    // Clear INT flag for this timer
91 //    EPwm2Regs.ETCLR.bit.INT = 1;
92 //
93 //    // Acknowledge this interrupt to receive more interrupts from group 3
94 //    PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
95 //
96 //}
97 //
98 //interrupt void epwm3_isr(void)
99 //{
100 //    if (EPwm3_DB_Direction == DB_UP)
101 //    {
102 //        if (EPwm3Regs.DBFED < EPWM3_MAX_DB)
103 //        {
104 //            EPwm3Regs.DBFED++;
105 //            EPwm3Regs.DBRED++;
106 //        }

```

```

107 //      else
108 //      {
109 //          EPwm3_DB_Direction = DB_DOWN;
110 //          EPwm3Regs.DBFED--;
111 //          EPwm3Regs.DBRED--;
112 //      }
113 //  }
114 //  else
115 //  {
116 //      if (EPwm3Regs.DBFED == EPWM3_MIN_DB)
117 //      {
118 //          EPwm3_DB_Direction = DB_UP;
119 //          EPwm3Regs.DBFED++;
120 //          EPwm3Regs.DBRED++;
121 //      }
122 //      else
123 //      {
124 //          EPwm3Regs.DBFED--;
125 //          EPwm3Regs.DBRED--;
126 //      }
127 //  }
128 //
129 //
130 //      EPwm3TimerIntCount++;
131 //
132 //      // Clear INT flag for this timer
133 //      EPwm3Regs.ETCLR.bit.INT = 1;
134 //
135 //      // Acknowledge this interrupt to receive more interrupts from group 3
136 //      PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
137 //
138 //}
139
140 void InitPwmModule() {
141
142     InitEPwm1Gpio();
143     InitEPwm2Gpio();
144     InitEPwm3Gpio();
145
146     PwmPointer = PwmPeriods;
147     PwmPeriods[0] = 0x358; //175kHz
148     PwmPeriods[1] = 0x2ED; //200kHz
149     PwmPeriods[2] = 0x29A; //225kHz
150     PwmPeriods[3] = 0x257; //250
151     PwmPeriods[4] = 0x220; //275
152     PwmPeriods[5] = 0x1F3; //300
153     PwmPeriods[6] = 0x1CC; //325
154     PwmPeriods[7] = 0x1AB; //350
155     PwmPeriods[8] = 0x18F; //375
156     PwmPeriods[9] = 0x176; //400
157     PwmPeriods[10] = 0x160; //425
158     PwmPeriods[11] = 0x14C; //450
159     PwmPeriods[12] = 0x13B; //475
160     PwmPeriods[13] = 0x12B; //500

```

```

161
162 EPwm1Init();
163     EPwm2Init();
164     EPwm3Init();
165     EPwm4Init();
166     EPwm5Init();
167     EPwm6Init();
168 }
169
170 void EPwm1Init()
171 {
172
173     EPwm1Regs.TBPRD = 374;                // Set timer period
174     EPwm1Regs.TBPHS.half.TBPHS = 0x0000; // Phase is 0
175     EPwm1Regs.TBCTR = 0x0000;            // Clear counter
176
177     // Setup TBCLK
178     EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Count up
179     EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;     // Disable phase loading
180 // EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV4;     // Clock ratio to
    SYSCLKOUT
181 // EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV4;
182 EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;        // Clock ratio to SYSCLKOUT
183 EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;
184 EPwm1Regs.TBCTL.bit.SYNCSEL = TB_CTR_ZERO;
185
186 EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;    // Load registers every ZERO
187 EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
188 EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
189 EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;
190
191 EALLOW;
192 EPwm1Regs.HRCNFG.all = 0x0;
193 EPwm1Regs.HRCNFG.bit.EDGMODE = HR_FEP;         // Falling Edge Position
194 EPwm1Regs.HRCNFG.bit.CTLMODE = HR_CMP;         // CMPAHR controls value.
195 EPwm1Regs.HRCNFG.bit.HRLOAD = HR_CTR_ZERO;     // Shadow load on CTR=0
196 // Setup compare
197 EPwm1Regs.CMPA.half.CMPA = 45;
198 EPwm1Regs.CMPA.half.CMPAHR = 0x0000;
199 // EPwm1Regs.CMPB = 3000;
200
201 // Set actions
202 EPwm1Regs.AQCTLA.bit.ZRO = AQ_SET;
203 EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR;           // Set PWM1A on Zero
204 // EPwm1Regs.AQCTLA.bit.CAD = AQ_CLEAR;
205
206 EPwm1Regs.AQCTLB.bit.ZRO = AQ_SET;
207 EPwm1Regs.AQCTLB.bit.CAU = AQ_CLEAR;           // Set PWM1A on Zero
208 // EPwm1Regs.AQCTLB.bit.CAD = AQ_SET;
209
210 // Active Low PWMs – Setup Deadband
211 EPwm1Regs.DBCTL.bit.OUTMODE = DB_FULL_ENABLE;
212 //EPwm1Regs.DBCTL.bit.POLSEL = DB_ACTV_LO;
213 EPwm1Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;

```

```

214 EPwm1Regs.DBCTL.bit.IN_MODE = DBA_ALL;
215 EPwm1Regs.DBRED = EPWM_MIN_DB;
216 EPwm1Regs.DBFED = EPWM_MIN_DB;
217
218 // Setup Trip-Zone for ePWM1:
219 EALLOW;
220 EPwm1Regs.TZSEL.bit.OSHT1 = TZ_ENABLE; //TZ1 is a one-shot killer
221 EPwm1Regs.TZCTL.bit.TZA = TZ_FORCE_LO;
222 EPwm1Regs.TZCTL.bit.TZB = TZ_FORCE_LO;
223 EPwm1Regs.TZEINT.bit.OST = TZ_ENABLE;
224 // EDIS;
225
226 EPwm1Regs.ETSEL.bit.INTEN = 0; // disable INT
227 }
228
229 void EPwm2Init()
230 {
231
232 EPwm2Regs.TBPRD = 374; // Set timer period
233 EPwm2Regs.TBPHS.half.TBPHS = 0x0000; // Phase is 0
234 EPwm2Regs.TBCTR = 0x0000; // Clear counter
235
236 // Setup TBCLK
237 EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Count up
238 EPwm2Regs.TBCTL.bit.PHSEN = TB_DISABLE; // Disable phase loading
239 // EPwm2Regs.TBCTL.bit.HSPCLKDIV = TB_DIV4; // Clock ratio to
SYSCLKOUT
240 // EPwm2Regs.TBCTL.bit.CLKDIV = TB_DIV4;
241 EPwm2Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1; // Clock ratio to SYSCLKOUT
242 EPwm2Regs.TBCTL.bit.CLKDIV = TB_DIV1;
243 EPwm2Regs.TBCTL.bit.SYNCSEL = TB_CTR_ZERO;
244
245 EPwm2Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW; // Load registers every ZERO
246 EPwm2Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
247 EPwm2Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
248 EPwm2Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;
249
250 EALLOW;
251 EPwm2Regs.HRCNFG.all = 0x0;
252 EPwm2Regs.HRCNFG.bit.EDGMODE = HR_FEP; //Falling Edge Position
253 EPwm2Regs.HRCNFG.bit.CTLMODE = HR_CMP; //CMPAHR controls value.
254 EPwm2Regs.HRCNFG.bit.HRLOAD = HR_CTR_ZERO; //Shadow load on CTR=0
255 // Setup compare
256 EPwm2Regs.CMPA.half.CMPA = 45;
257 EPwm2Regs.CMPA.half.CMPAHR = 0x0000;
258 // EPwm2Regs.CMPB = 3000;
259
260 // Set actions
261 EPwm2Regs.AQCTLA.bit.ZRO = AQ_SET;
262 EPwm2Regs.AQCTLA.bit.CAU = AQ_CLEAR; // Set PWM1A on Zero
263 // EPwm2Regs.AQCTLA.bit.CAD = AQ_CLEAR;
264
265 EPwm2Regs.AQCTLB.bit.ZRO = AQ_SET;
266 EPwm2Regs.AQCTLB.bit.CAU = AQ_CLEAR; // Set PWM1A on Zero

```

```

267 // EPwm2Regs.AQCTLB.bit.CAD = AQ_SET;
268
269 // Active Low PWMs - Setup Deadband
270 EPwm2Regs.DBCTL.bit.OUTMODE = DB_FULL_ENABLE;
271 //EPwm2Regs.DBCTL.bit.POLSEL = DB_ACTV_LO;
272 EPwm2Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
273 EPwm2Regs.DBCTL.bit.IN_MODE = DBA_ALL;
274 EPwm2Regs.DBRED = EPWM_MIN_DB;
275 EPwm2Regs.DBFED = EPWM_MIN_DB;
276
277 // Setup Trip-Zone for ePWM1:
278 EALLOW;
279 EPwm2Regs.TZSEL.bit.OSHT1 = TZ_ENABLE; //TZ1 is a one-shot killer
280 EPwm2Regs.TZCTL.bit.TZA = TZ_FORCE_LO;
281 EPwm2Regs.TZCTL.bit.TZB = TZ_FORCE_LO;
282 EPwm2Regs.TZEINT.bit.OST = TZ_ENABLE;
283 // EDIS;
284
285 EPwm2Regs.ETSEL.bit.INTEN = 0; // disable INT
286 }
287
288 void EPwm3Init()
289 {
290
291 EPwm3Regs.TBPRD = 374; // Set timer period
292 EPwm3Regs.TBPHS.half.TBPHS = 0x0000; // Phase is 0
293 EPwm3Regs.TBCTR = 0x0000; // Clear counter
294
295 // Setup TBCLK
296 EPwm3Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Count up
297 EPwm3Regs.TBCTL.bit.PHSEN = TB_DISABLE; // Disable phase loading
298 // EPwm3Regs.TBCTL.bit.HSPCLKDIV = TB_DIV4; // Clock ratio to
SYSCLKOUT
299 // EPwm3Regs.TBCTL.bit.CLKDIV = TB_DIV4;
300 EPwm3Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1; // Clock ratio to SYSCLKOUT
301 EPwm3Regs.TBCTL.bit.CLKDIV = TB_DIV1;
302 EPwm3Regs.TBCTL.bit.SYNCSEL = TB_CTR_ZERO;
303
304 EPwm3Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW; // Load registers every ZERO
305 EPwm3Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
306 EPwm3Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
307 EPwm3Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;
308
309 EALLOW;
310 EPwm3Regs.HRCNFG.all = 0x0;
311 EPwm3Regs.HRCNFG.bit.EDGMODE = HR_FEP; //Falling Edge Position
312 EPwm3Regs.HRCNFG.bit.CTLMODE = HR_CMP; //CMPAHR controls value.
313 EPwm3Regs.HRCNFG.bit.HRLOAD = HR_CTR_ZERO; //Shadow load on CTR=0
314 // Setup compare
315 EPwm3Regs.CMPA.half.CMPA = 45;
316 EPwm3Regs.CMPA.half.CMPAHR = 0x0000;
317 // EPwm3Regs.CMPB = 3000;
318
319 // Set actions

```

```

320 EPwm3Regs.AQCTLA.bit.ZRO = AQ_SET;
321 EPwm3Regs.AQCTLA.bit.CAU = AQ_CLEAR;           // Set PWM1A on Zero
322 // EPwm3Regs.AQCTLA.bit.CAD = AQ_CLEAR;
323
324 EPwm3Regs.AQCTLB.bit.ZRO = AQ_SET;
325 EPwm3Regs.AQCTLB.bit.CAU = AQ_CLEAR;           // Set PWM1A on Zero
326 // EPwm3Regs.AQCTLB.bit.CAD = AQ_SET;
327
328 // Active Low PWMs – Setup Deadband
329 EPwm3Regs.DBCTL.bit.OUTMODE = DB_FULL_ENABLE;
330 //EPwm3Regs.DBCTL.bit.POLSEL = DB_ACTV_LO;
331 EPwm3Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
332 EPwm3Regs.DBCTL.bit.IN_MODE = DBA_ALL;
333 EPwm3Regs.DBRED = EPWMLMIN_DB;
334 EPwm3Regs.DBFED = EPWMLMIN_DB;
335
336 // Setup Trip-Zone for ePWM1:
337 EALLOW;
338 EPwm3Regs.TZSEL.bit.OSHT1 = TZ_ENABLE; //TZ1 is a one-shot killer
339 EPwm3Regs.TZCTL.bit.TZA = TZ_FORCE_LO;
340 EPwm3Regs.TZCTL.bit.TZB = TZ_FORCE_LO;
341 EPwm3Regs.TZEINT.bit.OST = TZ_ENABLE;
342 // EDIS;
343
344 EPwm3Regs.ETSEL.bit.INTEN = 0;           // disable INT
345 }
346
347 void EPwm4Init()
348 {
349
350 EPwm4Regs.TBPRD = 374;           // Set timer period
351 EPwm4Regs.TBPHS.half.TBPHS = 0x0000; // Phase is 0
352 EPwm4Regs.TBCTR = 0x0000;       // Clear counter
353
354 // Setup TBCLK
355 EPwm4Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Count up
356 EPwm4Regs.TBCTL.bit.PHSEN = TB_DISABLE;     // Disable phase loading
357 // EPwm4Regs.TBCTL.bit.HSPCLKDIV = TB_DIV4; // Clock ratio to
SYSCLKOUT
358 // EPwm4Regs.TBCTL.bit.CLKDIV = TB_DIV4;
359 EPwm4Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1; // Clock ratio to SYSCLKOUT
360 EPwm4Regs.TBCTL.bit.CLKDIV = TB_DIV1;
361 EPwm4Regs.TBCTL.bit.SYNCSEL = TB_CTR_ZERO;
362
363 EPwm4Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW; // Load registers every ZERO
364 EPwm4Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
365 EPwm4Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
366 EPwm4Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;
367
368 EALLOW;
369 EPwm4Regs.HRCNFG.all = 0x0;
370 EPwm4Regs.HRCNFG.bit.EDGMODE = HR_FEP; //Falling Edge Position
371 EPwm4Regs.HRCNFG.bit.CILMODE = HR_CMP; //CMPAHR controls value.
372 EPwm4Regs.HRCNFG.bit.HRLOAD = HR_CTR_ZERO; //Shadow load on CTR=0

```

```

373 // Setup compare
374 EPwm4Regs.CMPA.half.CMPA = 45;
375 EPwm4Regs.CMPA.half.CMPAHR = 0x0000;
376 // EPwm4Regs.CMPB = 3000;
377
378 // Set actions
379 EPwm4Regs.AQCTLA.bit.ZRO = AQ_SET;
380 EPwm4Regs.AQCTLA.bit.CAU = AQ_CLEAR; // Set PWM1A on Zero
381 // EPwm4Regs.AQCTLA.bit.CAD = AQ_CLEAR;
382
383 EPwm4Regs.AQCTLB.bit.ZRO = AQ_SET;
384 EPwm4Regs.AQCTLB.bit.CAU = AQ_CLEAR; // Set PWM1A on Zero
385 // EPwm4Regs.AQCTLB.bit.CAD = AQ_SET;
386
387 // Active Low PWMs - Setup Deadband
388 EPwm4Regs.DBCTL.bit.OUTMODE = DB_FULL_ENABLE;
389 //EPwm4Regs.DBCTL.bit.POLSEL = DB_ACTV_LO;
390 EPwm4Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
391 EPwm4Regs.DBCTL.bit.IN_MODE = DBA_ALL;
392 EPwm4Regs.DBRED = EPWMLMIN_DB;
393 EPwm4Regs.DBFED = EPWMLMIN_DB;
394
395 // Setup Trip-Zone for ePWM1:
396 EALLOW;
397 EPwm4Regs.TZSEL.bit.OSHT1 = TZ_ENABLE; //TZ1 is a one-shot killer
398 EPwm4Regs.TZCTL.bit.TZA = TZ_FORCE_LO;
399 EPwm4Regs.TZCTL.bit.TZB = TZ_FORCE_LO;
400 EPwm4Regs.TZEINT.bit.OST = TZ_ENABLE;
401 // EDIS;
402
403 EPwm4Regs.ETSEL.bit.INTEN = 0; // disable INT
404 }
405
406 void EPwm5Init()
407 {
408
409 EPwm5Regs.TBPRD = 374; // Set timer period
410 EPwm5Regs.TBPHS.half.TBPHS = 0x0000; // Phase is 0
411 EPwm5Regs.TBCTR = 0x0000; // Clear counter
412
413 // Setup TBCLK
414 EPwm5Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Count up
415 EPwm5Regs.TBCTL.bit.PHSEN = TB_DISABLE; // Disable phase loading
416 // EPwm5Regs.TBCTL.bit.HSPCLKDIV = TB_DIV4; // Clock ratio to
SYSCLKOUT
417 // EPwm5Regs.TBCTL.bit.CLKDIV = TB_DIV4;
418 EPwm5Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1; // Clock ratio to SYSCLKOUT
419 EPwm5Regs.TBCTL.bit.CLKDIV = TB_DIV1;
420 EPwm5Regs.TBCTL.bit.SYNCSEL = TB_CTR_ZERO;
421
422 EPwm5Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW; // Load registers every ZERO
423 EPwm5Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
424 EPwm5Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
425 EPwm5Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

```

```

426
427     EALLOW;
428     EPwm5Regs.HRCNFG.all = 0x0;
429     EPwm5Regs.HRCNFG.bit.EDGMODE = HR_FEP;           //Falling Edge Position
430     EPwm5Regs.HRCNFG.bit.CTLMODE = HR_CMP;           //CMPAHR controls value.
431     EPwm5Regs.HRCNFG.bit.HRLOAD = HR_CTR_ZERO;       //Shadow load on CTR=0
432     // Setup compare
433     EPwm5Regs.CMPA.half.CMPA = 45;
434     EPwm5Regs.CMPA.half.CMPAHR = 0x0000;
435 //    EPwm5Regs.CMPB = 3000;
436
437     // Set actions
438     EPwm5Regs.AQCTLA.bit.ZRO = AQ_SET;
439     EPwm5Regs.AQCTLA.bit.CAU = AQ_CLEAR;              // Set PWM5A on Zero
440 //    EPwm5Regs.AQCTLA.bit.CAD = AQ_CLEAR;
441
442     EPwm5Regs.AQCTLB.bit.ZRO = AQ_SET;
443     EPwm5Regs.AQCTLB.bit.CAU = AQ_CLEAR;              // Set PWM5A on Zero
444 //    EPwm5Regs.AQCTLB.bit.CAD = AQ_SET;
445
446     // Active Low PWMs – Setup Deadband
447     EPwm5Regs.DBCTL.bit.OUTMODE = DB_FULL_ENABLE;
448     //EPwm5Regs.DBCTL.bit.POLSEL = DB_ACTV_LO;
449     EPwm5Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
450     EPwm5Regs.DBCTL.bit.IN_MODE = DBA_ALL;
451     EPwm5Regs.DBRED = EPWMLMIN_DB;
452     EPwm5Regs.DBFED = EPWMLMIN_DB;
453
454 // Setup Trip-Zone for ePWM1:
455     EALLOW;
456     EPwm5Regs.TZSEL.bit.OSHT1 = TZ_ENABLE; //TZ1 is a one-shot killer
457     EPwm5Regs.TZCTL.bit.TZA = TZ_FORCE_LO;
458     EPwm5Regs.TZCTL.bit.TZB = TZ_FORCE_LO;
459     EPwm5Regs.TZEINT.bit.OST = TZ_ENABLE;
460 //    EDIS;
461
462     EPwm5Regs.ETSEL.bit.INTEN = 0;                    // disable INT
463 }
464 void EPwm6Init()
465 {
466
467 //    EPwm6Regs.TBPRD = 6000;                          // Set timer period ~25
468 //    kHz
469     EPwm6Regs.TBPRD = 50;                             // Set timer period ~ 3MHz
470     EPwm6Regs.TBPHS.half.TBPHS = 0x0000;              // Phase is 0
471     EPwm6Regs.TBCTR = 0x0000;                          // Clear counter
472
473     // Setup TBCLK
474     EPwm6Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Count up
475     EPwm6Regs.TBCTL.bit.PHSEN = TB_DISABLE;          // Disable phase loading
476     EPwm6Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;         // Clock ratio to SYSCLKOUT
477     EPwm6Regs.TBCTL.bit.CLKDIV = TB_DIV1;            // Slow so we can observe on
478     the scope

```

```

478
479 // Setup compare
480 EPwm6Regs.CMPA.half.CMPA = 50;
481
482 // Set actions
483 EPwm6Regs.AQCTLA.bit.CAU = AQ_SET; // Set PWM6A on Zero
484 EPwm6Regs.AQCTLA.bit.CAD = AQ_CLEAR;
485
486 EPwm6Regs.AQCTLB.bit.CAU = AQ_CLEAR; // Set PWM6B on Zero
487 EPwm6Regs.AQCTLB.bit.CAD = AQ_SET;
488
489 // Active high complementary PWMs - Setup the deadband
490 EPwm6Regs.DBCTL.bit.OUTMODE = DB_DISABLE; //DB_FULL_ENABLE;
491 EPwm6Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
492 EPwm6Regs.DBCTL.bit.IN_MODE = DBA_ALL;
493 EPwm6Regs.DBRED = 0;
494 EPwm6Regs.DBFED = 0;
495
496 // Interrupt where we will change the deadband
497 // EPwm6Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO; // Select INT on Zero
    event
498 // EPwm6Regs.ETSEL.bit.INTEN = 1; // Enable INT
499 // EPwm6Regs.ETPS.bit.INTPRD = ET_3RD; // Generate INT on 3rd
    event
500 EPwm6Regs.ETSEL.bit.INTEN = 0; // Disable interrupt
    generation (EPWMx_INT)
501 EPwm6Regs.ETCLR.bit.SOCA = 1; // Clear any existing SOCA flag.
502 EPwm6Regs.ETSEL.bit.SOCAEN = 1; // Enable SOCA pulse
503 EPwm6Regs.ETSEL.bit.SOCASEL = ET_CTR_PRD; // SOCA when timebase = period
504 EPwm6Regs.ETPS.bit.SOCAPRD = 1; // Every event generates SOCA
505
506
507 }

```

I2CHandler.h

```

1 #ifndef I2CHANDLER_H
2 #define I2CHANDLER_H
3
4 #include "DSP28x_Project.h"           // Device Headerfile and Examples Include File
5
6 #define I2C_PCA0_ADDR      0x60 //0xC0
7 #define I2C_PCA1_ADDR      0x61 //0xC1
8 #define I2C_PCA2_ADDR      0x62 //0xC2
9 #define I2C_PCA3_ADDR      0x63 //0xC3
10
11 #define I2C_PCA_CTL_INPUT   0x00
12 #define I2C_PCA_CTL_PSC0    0x01
13 #define I2C_PCA_CTL_PWM0    0x02
14 #define I2C_PCA_CTL_PSC1    0x03
15 #define I2C_PCA_CTL_PWM1    0x04
16 #define I2C_PCA_CTL_LS0     0x05
17 #define I2C_PCA_CTL_LS1     0x06
18
19 #define I2C_PCA_CTL_AUTO    0x10;
20
21 #define LCD_OFF              0x00;
22 #define LCD_ON               0x01;
23 #define LCD_BLINK1          0x02;
24 #define LCD_BLINK2          0x03;
25
26 #define I2C_NUMBYTES        4
27 #define I2C_EEPROM_HIGH_ADDR 0x00
28 #define I2C_EEPROM_LOW_ADDR 0x30
29
30 #define I2C_TX               0x00
31 #define I2C_RX               0x01
32
33 #define I2C_BUFFER_SIZE      16
34 //extern Uint16 I2CTXFlag;
35 //extern Uint16 I2CRXFlag;
36
37 struct I2C_MSG{
38     Uint16 SlaveAddress;
39     Uint16 CommandByte;
40     Uint16 MsgStatus;
41     Uint16 NumOfBytes;
42 // void (*CallbackFunc)(struct I2C_MSG* msg);
43     Uint16 MsgBuffer[8]; //Should not need full buffer size
44 // Uint16 MsgBuffer[I2C_MAX_BUFFER_SIZE];
45 };
46
47 extern struct I2C_MSG I2cMsgOut1;
48
49 extern struct I2C_MSG I2cMsgIn1;
50
51 void I2CA_Init();
52 //void InitI2CModule(void);

```

```

53 void InitPCAs();
54 Uint16 PollButtons(); //Determine which phases are inserted into the carrier.
55 Uint16 PollPhases(); //Determine which phases are inserted into the carrier.
56 void SetPCA1Status(Uint16 value); //Set PCA1 LED Status. 2 bits per LED:
57 void SetPCA3Status(Uint16 value); //Set PCA1 LED Status. 2 bits per LED:
58
59 Uint16 I2CA_WriteData(struct I2C_MSG *msg);
60 Uint16 I2CA_ReadData(struct I2C_MSG *msg);
61 Uint16 I2CA_ReadDataBlocking(struct I2C_MSG *msg);
62 //Uint16 I2CA_ReadDataBlocking(struct I2C_MSG *msg);
63 interrupt void i2c_int1a_isr(void);
64
65
66 //Uint16 I2CA_WriteDataAsync(struct I2C_MSG *msg);
67
68 #endif /*I2CHANDLER_H-*/

```

I2CHandler.c

```

1 #include "I2CHandler.h"
2 #include "SerialHandler.h"
3 #include "DSP28x_Project.h"           // Device Headerfile and Examples Include File
4 //
5 //
6 struct I2C_MSG I2cMsgOut1={I2C_PCA1_ADDR,
7                             I2C_PCA_CTL_LS0,
8                             1,
9                             0x42 };
10
11 struct I2C_MSG I2cMsgIn1={ I2C_PCA2_ADDR,
12                             I2C_PCA_CTL_INPUT,
13                             1};
14 //
15 //Uint16 i2cMode = 0;
16 //
17 struct I2C_MSG I2CBuffer[I2CBUFFSIZE];
18 struct I2C_MSG *I2CFIFOFront;
19 struct I2C_MSG *I2CFIFOBack;
20 struct I2C_MSG *I2CBufferEnd;
21 //
22 //
23 //Uint16 I2CTXFlag;
24 //Uint16 I2CRXFlag;
25 //
26 //
27 //void InitI2CModule(void){
28 //    // Initialize I2C
29 //    I2caRegs.I2CSAR = I2C_PCA0_ADDR;    // Slave address - EEPROM control
30 //    code
31 //    //if (CPU_FRQ_150MHZ)                // Default - For 150MHz SYSCLKOUT
32 //        I2caRegs.I2CPSC.all = 14;    // Prescaler - need 7-12 Mhz on module
33 //        clk (150/15 = 10MHz)
34 //    //endif
35 //    //if (CPU_FRQ_100MHZ)                // For 100 MHz SYSCLKOUT
36 //        I2caRegs.I2CPSC.all = 9;    // Prescaler - need 7-12 Mhz on module
37 //        clk (100/10 = 10MHz)
38 //    //endif
39 //    //
40 //    I2caRegs.I2CCLKL = 10;    // NOTE: must be non zero
41 //    I2caRegs.I2CCLKH = 5;    // NOTE: must be non zero
42 //    I2caRegs.I2CCLKL = 100;    // NOTE: must be non zero
43 //    I2caRegs.I2CCLKH = 100;    // NOTE: must be non zero
44 //    I2caRegs.I2CIER.all = 0x24;    // Enable SCD & ARDY interrupts
45 //    //
46 //    I2caRegs.I2CFFTX.all = 0x6000; // Enable FIFO mode and TXFIFO
47 //    I2caRegs.I2CFFRX.all = 0x2040; // Enable RXFIFO, clear RXFFINT,
48 //    //
49 //    I2caRegs.I2CMDR.all = 0x0020; // Take I2C out of reset
50 //    // Stop I2C when suspended
51 //    i2cMode = I2C_TX;

```

```

50  //// return;
51  ////}
52  ////
53  void InitPCAs() {
54      Uint16 result;
55      I2cMsgOut1.SlaveAddress = I2C_PCA1_ADDR;
56      I2cMsgOut1.CommandByte = I2C_PCA_CTL_PSC0 | I2C_PCA_CTL_AUTO;
57      I2cMsgOut1.MsgBuffer[0] = 0x03; //PCS0 ~ 40Hz
58      I2cMsgOut1.MsgBuffer[1] = 0x30; //PWM0
59      I2cMsgOut1.MsgBuffer[2] = 0x97; //PCS1 ~ half a second
60      I2cMsgOut1.MsgBuffer[3] = 0x80; //PWM1
61      I2cMsgOut1.MsgBuffer[4] = 0x00; //LS0
62      I2cMsgOut1.MsgBuffer[5] = 0x00; //LS1
63      I2cMsgOut1.NumOfBytes = 6;
64      do{
65          result = I2CA_WriteData(&I2cMsgOut1);
66      }while(result != I2C_SUCCESS);
67      I2cMsgOut1.SlaveAddress = I2C_PCA3_ADDR;
68      do{
69          result = I2CA_WriteData(&I2cMsgOut1);
70      }while(result != I2C_SUCCESS);
71  }
72
73
74  Uint16 PollPhases() {
75      Uint16 result;
76      I2cMsgIn1.SlaveAddress = I2C_PCA0_ADDR;
77      I2cMsgIn1.CommandByte = I2C_PCA_CTL_INPUT;
78      I2cMsgIn1.NumOfBytes = 1;
79      do{
80          result = I2CA_ReadDataBlocking(&I2cMsgIn1);
81      }while(result != I2C_SUCCESS);
82      return I2cMsgIn1.MsgBuffer[1];
83  }
84
85  Uint16 PollButtons() {
86      Uint16 result;
87      I2cMsgIn1.SlaveAddress = I2C_PCA2_ADDR;
88      I2cMsgIn1.CommandByte = I2C_PCA_CTL_INPUT;
89      I2cMsgIn1.NumOfBytes = 1;
90      do{
91          result = I2CA_ReadDataBlocking(&I2cMsgIn1);
92      }while(result != I2C_SUCCESS);
93      return I2cMsgIn1.MsgBuffer[1];
94  }
95
96  void SetPCA1Status(Uint16 value){
97      Uint16 retval = 0;
98      I2cMsgOut1.SlaveAddress = I2C_PCA1_ADDR;
99      I2cMsgOut1.CommandByte = I2C_PCA_CTL_LS0 | I2C_PCA_CTL_AUTO;
100     I2cMsgOut1.MsgBuffer[0] = value & 0x00FF;
101     I2cMsgOut1.MsgBuffer[1] = (value >> 8) & 0x00FF;
102     I2cMsgOut1.NumOfBytes = 2;
103     do{

```

```

104     retval = I2CA_WriteData(&I2cMsgOut1);
105 }
106 while (retval != I2C_SUCCESS);
107 if(retval != I2C_SUCCESS){
108     char* msg;
109     msg = "Error in Writing I2C SetPCA1\0";
110     scia_msg_async(msg);
111 }
112 }
113 void SetPCA3Status(Uint16 value){
114     Uint16 retval = 0;
115     I2cMsgOut1.SlaveAddress = I2C_PCA3_ADDR;
116     I2cMsgOut1.CommandByte = I2C_PCA_CTL_LS0 | I2C_PCA_CTL_AUTO;
117     I2cMsgOut1.MsgBuffer[0] = value & 0x00FF;
118     I2cMsgOut1.MsgBuffer[1] = (value>>8) & 0x00FF;
119     I2cMsgOut1.NumOfBytes = 2;
120     do{
121         retval = I2CA_WriteData(&I2cMsgOut1);
122     }
123     while (retval != I2C_SUCCESS);
124     if(retval != I2C_SUCCESS){
125         char* msg;
126         msg = "Error in Writing I2C SetPCA1\0";
127         scia_msg_async(msg);
128     }
129 }
130
131 void I2CA_Init(void)
132 {
133     // Initialize I2C
134     I2caRegs.I2CSAR = I2C_PCA0_ADDR;    // Slave address - EEPROM control code
135
136     #if (CPU_FRQ_150MHZ)                // Default - For 150MHz SYSCLKOUT
137         I2caRegs.I2CPSC.all = 14;    // Prescaler - need 7-12 Mhz on module clk
138         (150/15 = 10MHz)
139     #endif
140     #if (CPU_FRQ_100MHZ)                // For 100 MHz SYSCLKOUT
141         I2caRegs.I2CPSC.all = 9;    // Prescaler - need 7-12 Mhz on module clk
142         (100/10 = 10MHz)
143     #endif
144
145     // I2caRegs.I2CCLKL = 10;    // NOTE: must be non zero
146     // I2caRegs.I2CCLKH = 5;    // NOTE: must be non zero
147     I2caRegs.I2CCLKL = 50;    // NOTE: must be non zero
148     I2caRegs.I2CCLKH = 50;    // NOTE: must be non zero
149     I2caRegs.I2CIER.all = 0x24;    // Enable SCD & ARDY interrupts
150
151     I2caRegs.I2CFFTX.all = 0x6000; // Enable FIFO mode and TXFIFO
152     I2caRegs.I2CFFRX.all = 0x2040; // Enable RXFIFO, clear RXFFINT,
153
154     I2caRegs.I2CMDR.all = 0x0020; // Take I2C out of reset
155     // Stop I2C when suspended
156     //// I2CBuffer[I2CBUFFSIZE];

```

```

156 // I2CFIFOFront = I2CBuffer;
157 // I2CFIFOBack = I2CBuffer;
158 // I2CBufferEnd = I2CBuffer + sizeof(struct I2C_MSG)*(I2CBUFFSIZE-1);
159 // I2CBufferEnd = &(I2CBuffer[I2CBUFFSIZE-1]);
160 return;
161 }
162
163 Uint16 I2CA_WriteData(struct I2C_MSG *msg)
164 {
165     Uint16 i;
166
167     // Wait until the STP bit is cleared from any previous master communication
168
169     // Clearing of this bit by the module is delayed until after the SCD bit is
170     // set. If this bit is not checked prior to initiating a new message, the
171     // I2C could get confused.
172     if (I2caRegs.I2CMDR.bit.STP == 1)
173     {
174         return I2C_STP_NOT_READY_ERROR;
175     }
176
177     // Setup slave address
178     I2caRegs.I2CSAR = msg->SlaveAddress;
179     // I2caRegs.I2CSAR = I2C_PCA1_ADDR;
180
181     // Check if bus busy
182     if (I2caRegs.I2CSTR.bit.BB == 1)
183     {
184         return I2C_BUS_BUSY_ERROR;
185     }
186
187     // Setup number of bytes to send
188     // MsgBuffer + Address
189     I2caRegs.I2CCNT = msg->NumOfBytes+1;
190     // I2caRegs.I2CCNT = 1; //send control byte and data byte.
191
192     // Setup data to send
193     I2caRegs.I2CDXR = msg->CommandByte;
194     // I2caRegs.I2CDXR = msg->MemoryLowAddr;
195     // for (i=0; i<msg->NumOfBytes-2; i++)
196     for (i=0; i<msg->NumOfBytes; i++)
197     {
198         I2caRegs.I2CDXR = *(msg->MsgBuffer+i);
199     }
200     // I2caRegs.I2CDXR = I2C_PCA_CTL_LS0;
201     // I2caRegs.I2CDXR = 0x42;
202
203     // Send start as master transmitter
204     I2caRegs.I2CMDR.all = 0x6E20;
205
206     return I2C_SUCCESS;
207 }
208
209 Uint16 I2CA_ReadData(struct I2C_MSG *msg)

```

```

209 {
210     // Wait until the STP bit is cleared from any previous master communication
211     .
212     // Clearing of this bit by the module is delayed until after the SCD bit is
213     // set. If this bit is not checked prior to initiating a new message, the
214     // I2C could get confused.
215     if (I2caRegs.I2CMDR.bit.STP == 1)
216     {
217         return I2C_STP_NOT_READY_ERROR;
218     }
219     I2caRegs.I2CSAR = msg->SlaveAddress;
220
221     // I2caRegs.I2CCNT = 2;
222     // I2caRegs.I2CDXR = msg->MemoryHighAddr;
223     // I2caRegs.I2CDXR = msg->MemoryLowAddr;
224
225     // I2caRegs.I2CDXR = msg->MemoryHighAddr;
226     I2caRegs.I2CMDR.all = 0x2620; // Send data to setup EEPROM address
227
228     //else if(msg->MsgStatus == I2C_MSGSTAT_RESTART)
229     {
230         I2caRegs.I2CCNT = msg->NumOfBytes; // Setup how many bytes to expect
231         // I2caRegs.I2CMDR.all = 0x2C20; // Send restart as master receiver
232         I2caRegs.I2CMDR.all = 0xAC20; // Send restart as master receiver,
233         // enable NACK.
234     }
235     return I2C_SUCCESS;
236 }
237
238
239 Uint16 I2CA_ReadDataBlocking(struct I2C_MSG *msg)
240 {
241     // Wait until the STP bit is cleared from any previous master communication
242     .
243     // Clearing of this bit by the module is delayed until after the SCD bit is
244     // set. If this bit is not checked prior to initiating a new message, the
245     // I2C could get confused.
246     if (I2caRegs.I2CMDR.bit.STP == 1)
247     {
248         return I2C_STP_NOT_READY_ERROR;
249     }
250     I2caRegs.I2CSAR = msg->SlaveAddress;
251
252     // I2caRegs.I2CCNT = 2;
253     // I2caRegs.I2CDXR = msg->MemoryHighAddr;
254     // I2caRegs.I2CDXR = msg->MemoryLowAddr;
255
256     // I2caRegs.I2CDXR = msg->MemoryHighAddr;
257     I2caRegs.I2CMDR.all = 0x2620; // Send data to setup EEPROM address
258
259     //else if(msg->MsgStatus == I2C_MSGSTAT_RESTART)

```

```

260 {
261     I2caRegs.I2CCNT = msg->NumOfBytes; // Setup how many bytes to expect
262 //     I2caRegs.I2CMDR.all = 0x2C20; // Send restart as master receiver
263     I2caRegs.I2CMDR.all = 0xAC20; // Send restart as master receiver,
        enable NACK.
264 }
265 while(I2caRegs.I2CSTR.bit.NACKSNT == 0){
266     ;
267 }
268 I2cMsgIn1.MsgBuffer[1] = I2caRegs.I2CDRR;
269 I2caRegs.I2CSTR.bit.NACKSNT = 1;
270
271
272 return I2C_SUCCESS;
273 }
274
275
276 interrupt void i2c_int1a_isr(void) // I2C-A
277 {
278     Uint16 IntSource;
279 //     Uint16 res;
280
281     // Read interrupt source
282     IntSource = I2caRegs.I2CISRC.all;
283
284     // Interrupt source = stop condition detected
285     switch(IntSource){
286     case I2C.SCD_ISRC: // stop condition detected
287 //         if(I2CFIFOFront != I2CFIFOBack) // Data in I2CBuffer
288 //             { //try to send out data in our queue.
289 //                 res = I2CA_WriteData(I2CFIFOFront);
290 //                 if(res == I2C_SUCCESS){
291 //                     I2CFIFOFront++;
292 //                     if(I2CFIFOFront > I2CBufferEnd)
293 //                         I2CFIFOFront = I2CBuffer;
294 //                 }
295 //                 I2caRegs.I2CFFTX.bit.TXFFINTCLR = 1;
296 //                 break;
297 //             }
298         //nothing in queue, must be receiving data:
299     case I2C.RX_ISRC:
300         I2cMsgIn1.MsgBuffer[0] = I2caRegs.I2CCNT;
301         I2cMsgIn1.MsgBuffer[1] = I2caRegs.I2CDRR;
302         break;
303     case I2C.ARDY_ISRC: // register access ready
304         break;
305     case I2C.NO_ISRC: //No Interrupt source
306         break;
307     case I2C.ARB_ISRC:
308         break; // Arbitration lost
309     case I2C.NACK_ISRC: // No-ack condition detected
310         break;
311     case I2C.TX_ISRC: // Transmit data ready
312         break;

```

```

313     case I2C_AAS_ISRC: // Addressed as slave
314         break;
315     default:
316         break;
317 }
318 // Enable future I2C (PIE Group 8) interrupts
319 PieCtrlRegs.PIEACK.all = PIEACK_GROUP8;
320 }
321 //
322 // Uint16 I2CA_WriteDataAsync(struct I2C_MSG *msg){
323 //     *I2CFIFOBack++ = *msg; //Possibly very expensive.
324 //     if(I2CFIFOBack > I2CBufferEnd)
325 //         I2CFIFOBack = I2CBuffer;
326 //     if(I2caRegs.I2CFFTX.bit.TXFFIENA == 0){
327 //         I2CA_WriteData(I2CFIFOFront);
328 //         I2caRegs.I2CFFTX.bit.TXFFIENA = 1;
329 //     }
330 //     return I2C_SUCCESS;
331 //}
332 //

```

DSP28x_Project.h

```

1
2 // TI File $Revision: /main/1 $
3 // Checkin $Date: April 22, 2008 14:35:56 $
4 //#####
5 //
6 // FILE:    DSP28x_Project.h
7 //
8 // TITLE:   DSP28x Project Headerfile and Examples Include File
9 //
10 //#####
11 // $TI Release: 2833x/2823x Header Files V1.32 $
12 // $Release Date: June 28, 2010 $
13 //#####
14
15 #ifndef DSP28x_PROJECT_H
16 #define DSP28x_PROJECT_H
17
18 #include "DSP2833x_Device.h"    // DSP2833x Headerfile Include File
19 #include "DSP2833x_Examples.h"  // DSP2833x Examples Include File
20
21 #endif // end of DSP28x_PROJECT_H definition

```

DSP2833x_Device.h

```

1 // TI File $Revision: /main/5 $
2 // Checkin $Date: January 22, 2008 16:55:35 $
3 //#####
4 //
5 // FILE:    DSP2833x_Device.h
6 //
7 // TITLE:   DSP2833x Device Definitions.
8 //
9 //#####
10 // $TI Release: 2833x/2823x Header Files V1.32 $
11 // $Release Date: June 28, 2010 $
12 //#####
13
14 #ifndef DSP2833x_DEVICE_H
15 #define DSP2833x_DEVICE_H
16
17
18 #ifdef __cplusplus
19 extern "C" {
20 #endif
21
22
23 #define TARGET 1
24 //-----
25 // User To Select Target Device:
26
27 #define DSP28_28335 TARGET // Selects '28335/'28235
28 #define DSP28_28334 0 // Selects '28334/'28234
29 #define DSP28_28332 0 // Selects '28332/'28232
30
31
32 //-----
33 // Common CPU Definitions:
34 //
35
36 extern cregister volatile unsigned int IFR;
37 extern cregister volatile unsigned int IER;
38
39 #define EINT asm(" clrc INTM")
40 #define DINT asm(" setc INTM")
41 #define ERIM asm(" clrc DBGM")
42 #define DRIM asm(" setc DBGM")
43 #define EALLOW asm(" EALLOW")
44 #define EDIS asm(" EDIS")
45 #define ESTOP0 asm(" ESTOP0")
46
47 #define MINT1 0x0001
48 #define MINT2 0x0002
49 #define MINT3 0x0004
50 #define MINT4 0x0008
51 #define MINT5 0x0010
52 #define MINT6 0x0020

```

```

53 #define M_INT7    0x0040
54 #define M_INT8    0x0080
55 #define M_INT9    0x0100
56 #define M_INT10   0x0200
57 #define M_INT11   0x0400
58 #define M_INT12   0x0800
59 #define M_INT13   0x1000
60 #define M_INT14   0x2000
61 #define MDLOG     0x4000
62 #define MRTOS     0x8000
63
64 #define BIT0       0x0001
65 #define BIT1       0x0002
66 #define BIT2       0x0004
67 #define BIT3       0x0008
68 #define BIT4       0x0010
69 #define BIT5       0x0020
70 #define BIT6       0x0040
71 #define BIT7       0x0080
72 #define BIT8       0x0100
73 #define BIT9       0x0200
74 #define BIT10      0x0400
75 #define BIT11      0x0800
76 #define BIT12      0x1000
77 #define BIT13      0x2000
78 #define BIT14      0x4000
79 #define BIT15      0x8000
80
81
82
83 //-----
84 // For Portability, User Is Recommended To Use Following Data Type Size
85 // Definitions For 16-bit and 32-Bit Signed/Unsigned Integers:
86 //
87
88 #ifndef DSP28_DATA_TYPES
89 #define DSP28_DATA_TYPES
90 typedef int          int16;
91 typedef long         int32;
92 typedef long long    int64;
93 typedef unsigned int  Uint16;
94 typedef unsigned long Uint32;
95 typedef unsigned long long Uint64;
96 typedef float        float32;
97 typedef long double   float64;
98 #endif
99
100
101 //-----
102 // Include All Peripheral Header Files:
103 //
104
105 #include "DSP2833x_Adc.h"           // ADC Registers
106 #include "DSP2833x_DevEmu.h"       // Device Emulation Registers

```

```

107 #include "DSP2833x_CpuTimers.h"           // 32-bit CPU Timers
108 #include "DSP2833x_ECan.h"               // Enhanced eCAN Registers
109 #include "DSP2833x_ECap.h"               // Enhanced Capture
110 #include "DSP2833x_DMA.h"                // DMA Registers
111 #include "DSP2833x_EPwm.h"               // Enhanced PWM
112 #include "DSP2833x_EQep.h"               // Enhanced QEP
113 #include "DSP2833x_Gpio.h"               // General Purpose I/O Registers
114 #include "DSP2833x_I2c.h"                // I2C Registers
115 #include "DSP2833x_McBSP.h"              // McBSP
116 #include "DSP2833x_PieCtrl.h"            // PIE Control Registers
117 #include "DSP2833x_PieVect.h"            // PIE Vector Table
118 #include "DSP2833x_Spi.h"                // SPI Registers
119 #include "DSP2833x_Sci.h"                // SCI Registers
120 #include "DSP2833x_SysCtrl.h"            // System Control/Power Modes
121 #include "DSP2833x_XIntrupt.h"           // External Interrupts
122 #include "DSP2833x_Xintf.h"              // XINTF External Interface
123
124 #if DSP28_28335
125 #define DSP28_EPWM1 1
126 #define DSP28_EPWM2 1
127 #define DSP28_EPWM3 1
128 #define DSP28_EPWM4 1
129 #define DSP28_EPWM5 1
130 #define DSP28_EPWM6 1
131 #define DSP28_ECAP1 1
132 #define DSP28_ECAP2 1
133 #define DSP28_ECAP3 1
134 #define DSP28_ECAP4 1
135 #define DSP28_ECAP5 1
136 #define DSP28_ECAP6 1
137 #define DSP28_EQEP1 1
138 #define DSP28_EQEP2 1
139 #define DSP28_ECANA 1
140 #define DSP28_ECANB 1
141 #define DSP28_MCBSPA 1
142 #define DSP28_MCBSPB 1
143 #define DSP28_SPIA 1
144 #define DSP28_SCIA 1
145 #define DSP28_SCIB 1
146 #define DSP28_SCIC 1
147 #define DSP28_I2CA 1
148 #endif // end DSP28_28335
149
150 #if DSP28_28334
151 #define DSP28_EPWM1 1
152 #define DSP28_EPWM2 1
153 #define DSP28_EPWM3 1
154 #define DSP28_EPWM4 1
155 #define DSP28_EPWM5 1
156 #define DSP28_EPWM6 1
157 #define DSP28_ECAP1 1
158 #define DSP28_ECAP2 1
159 #define DSP28_ECAP3 1
160 #define DSP28_ECAP4 1

```

```

161 #define DSP28_ECAP5 0
162 #define DSP28_ECAP6 0
163 #define DSP28_EQEP1 1
164 #define DSP28_EQEP2 1
165 #define DSP28_ECANA 1
166 #define DSP28_ECANB 1
167 #define DSP28_MCBSPA 1
168 #define DSP28_MCBSPB 1
169 #define DSP28_SPIA 1
170 #define DSP28_SCIA 1
171 #define DSP28_SCIB 1
172 #define DSP28_SCIC 1
173 #define DSP28_I2CA 1
174 #endif // end DSP28-28334
175
176 #if DSP28_28332
177 #define DSP28_EPWM1 1
178 #define DSP28_EPWM2 1
179 #define DSP28_EPWM3 1
180 #define DSP28_EPWM4 1
181 #define DSP28_EPWM5 1
182 #define DSP28_EPWM6 1
183 #define DSP28_ECAP1 1
184 #define DSP28_ECAP2 1
185 #define DSP28_ECAP3 1
186 #define DSP28_ECAP4 1
187 #define DSP28_ECAP5 0
188 #define DSP28_ECAP6 0
189 #define DSP28_EQEP1 1
190 #define DSP28_EQEP2 1
191 #define DSP28_ECANA 1
192 #define DSP28_ECANB 1
193 #define DSP28_MCBSPA 1
194 #define DSP28_MCBSPB 0
195 #define DSP28_SPIA 1
196 #define DSP28_SCIA 1
197 #define DSP28_SCIB 1
198 #define DSP28_SCIC 0
199 #define DSP28_I2CA 1
200 #endif // end DSP28-28332
201
202 #ifdef __cplusplus
203 }
204 #endif /* extern "C" */
205
206 #endif // end of DSP2833x_DEVICE_H definition
207
208
209 //=====
210 // End of file.
211 //=====

```

Appendix B

Source Code for Single Phase Controller:

The single phase controller utilizes the same files as the two phase controller, with only a different main.c, shown below.

main.c

```

1 //#####
2 //
3 // FILE:      main.c
4 //
5 // TITLE:     For use with Buck Carrier board v1 and buck daughter boards v3+.
6 //
7 // ASSUMPTIONS:
8 //
9 //      This program requires the DSP2833x header files.
10 //      As supplied, this project is configured for "boot to SARAM" operation.
11 //
12 //      Connect the SCI-A port to a PC via a transceiver and cable.
13 //      The PC application 'hypterterminal' can be used to view the data
14 //      from the SCI and to send information to the SCI. Characters recieved
15 //      by the SCI port are sent back to the host.
16 //
17 //      As supplied, this project is configured for "boot to SARAM"
18 //      operation. The 2833x Boot Mode table is shown below.
19 //      For information on configuring the boot mode of an eZdsp,
20 //      please refer to the documentation included with the eZdsp,
21 //
22 //      $Boot_Table:
23 //
24 //      GPIO87      GPIO86      GPIO85      GPIO84
25 //      XA15        XA14        XA13        XA12
26 //      PU          PU          PU          PU
27 //
28 //      1           1           1           1      Jump to Flash
29 //      1           1           1           0      SCI-A boot
30 //      1           1           0           1      SPI-A boot
31 //      1           1           0           0      I2C-A boot
32 //      1           0           1           1      eCAN-A boot
33 //      1           0           1           0      McBSP-A boot
34 //      1           0           0           1      Jump to XINTF x16
35 //      1           0           0           0      Jump to XINTF x32
36 //      0           1           1           1      Jump to OTP
37 //      0           1           1           0      Parallel GPIO I/O boot

```

```

38 //          0          1          0          1    Parallel XINTF boot
39 //          0          1          0          0    Jump to SARAM      <- "boot to
    SARAM"
40 //          0          0          1          1    Branch to check boot mode
41 //          0          0          1          0    Boot to flash , bypass ADC cal
42 //          0          0          0          1    Boot to SARAM, bypass ADC cal
43 //          0          0          0          0    Boot to SCI-A, bypass ADC cal
44 //                                     Boot_Table_End$
45 //
46 // DESCRIPTION:
47 //     Communicates with the computer via serial port 9600 baud, 8 data,
48 //     1 stop, no parity.
49 //
50 //     As is, the program configures SCI-A for 9600 baud with
51 //     SYSCLKOUT = 150MHz and LSPCLK = 37.5 MHz
52 //#####
53 #include "SerialHandler.h"
54 #include "ADCHandler.h"
55 #include "PWMHandler.h"
56 #include "TimerHandler.h"
57 #include "I2CHandler.h"
58 #include "Globals.h"
59 #include "controller.h"
60 #include "DSP28x_Project.h"      // Device Headerfile and Examples Include File
61
62
63 //Uint16 LoopCount;
64 //Uint16 ErrorCount;
65 //Uint16 ConversionCount;
66 //Uint16 XmitFlag;
67 Uint16 PhaseStatus;
68 Uint16 PhaseCount;
69 union LEDS LEDStatus;
70 union LEDS LEDStatus2;
71
72
73 union MAINFLAGS MainFlags;
74 union BUTTONS ButtonStatus;
75
76 int32 a1,a2,a3,a4,a5, S,S1old,S2old,S1,S2,dS1,dS2,xint;
77 int32 u1,u1old1,u1old2;
78 const int32 alpha1 = 10; //100; //-2;
79 const int32 alpha2 = -2; //4;
80 const int32 alpha3 = 1; //-2;
81 const int32 alpha4 = 0; //10; //10;
82 const int32 alpha5 = 0; //10; //10; //-1;
83 const int32 alpha6 = 0; //1; //1; //-1;
84 const int32 alpha7 = 0; //2; //2; //-1;
85 const int32 alphaC0 = 0; //-200; //-10;
86 const int32 alphaC1 = 0; //-200; //-10;
87 const int16 Vref = 1160;
88 const int32 kappa = 80;
89 const int32 negkappa = -80;
90 const int32 taud1 = 0; //10;

```

```

91 const int32 taud2 = 0;//5;
92
93 int32 adcbuffer[32];
94
95 #pragma DATA_SECTION(DMABuf1,"DMARAML4");
96 volatile Uint16 DMABuf1[40];
97
98 volatile Uint16 *DMADest;
99 volatile Uint16 *DMASource;
100
101 void main(void)
102 {
103     Uint16 i,i2ccounter,idx;
104     int32 ramp1,ramp2;
105     int32 adcV1,adcV2,adcI1,adcI2;
106     int32 x6,x7;
107     int32* adcP;
108     Uint32 c0,c1;
109     // Uint16 tmp;
110
111
112     char *msg;
113
114     EALLOW;
115
116     MainFlags.all = 0;
117
118     // Step 1. Initialize System Control:
119     // PLL, WatchDog, enable Peripheral Clocks
120     // This example function is found in the DSP2833x_SysCtrl.c file.
121     InitSysCtrl();
122
123     // Step 2. Initalize GPIO:
124     // This example function is found in the DSP2833x_Gpio.c file and
125     // illustrates how to set the GPIO to it's default state.
126     // InitGpio(); Skipped for this example
127
128     // For this example, only init the pins for the SCI-A port.
129     // This function is found in the DSP2833x_Sci.c file.
130     InitSciaGpio();
131     // InitI2CGpio();
132
133     // Specific clock setting for this example:
134     EALLOW;
135     SysCtrlRegs.HISPCP.all = ADCMODCLK; // HSPCLK = SYSCLKOUT/ADC.MODCLK
136     //ADC clock now = 25MHz.
137     // EDIS;
138
139     // Step 3. Clear all interrupts and initialize PIE vector table:
140     // Disable CPU interrupts
141     DINT;
142
143     // Initialize PIE control registers to their default state.
144     // The default state is all PIE interrupts disabled and flags

```

```

145 // are cleared.
146 // This function is found in the DSP2833x_PieCtrl.c file.
147     InitPieCtrl();
148
149 // Disable CPU interrupts and clear all CPU interrupt flags:
150     IER = 0x0000;
151     IFR = 0x0000;
152
153 // Initialize the PIE vector table with pointers to the shell Interrupt
154 // Service Routines (ISR).
155 // This will populate the entire table, even if the interrupt
156 // is not used in this example. This is useful for debug purposes.
157 // The shell ISR routines are found in DSP2833x_DefaultIsr.c.
158 // This function is found in DSP2833x_PieVect.c.
159     InitPieVectTable();
160
161
162 // Interrupts that are used in this example are re-mapped to
163 // ISR functions found within this file.
164     EALLOW; // This is needed to write to EALLOW protected register
165 //     PieVectTable.ADCINT = &adc_isr;
166 //     PieVectTable.SCIRXINTA = &scirxa_isr;
167 //     PieVectTable.SCITXINTA = &scitxa_isr;
168 //     PieVectTable.TINT0 = &timer0_isr;
169 //     PieVectTable.I2CINT1A = &i2c_int1a_isr;
170 //     PieVectTable.EPWM1INT = &epwm1_isr;
171 //     PieVectTable.EPWM2INT = &epwm2_isr;
172 //     PieVectTable.EPWM3INT = &epwm3_isr;
173 //     EDIS; // This is needed to disable write to EALLOW protected
        registers
174
175     EALLOW;
176     SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0;
177 //     EDIS;
178
179 // InitPwmModule();
180 // InitTimerModule();
181 // InitI2CModule();
182 // I2CA_Init();
183     InitPwmModule();
184
185     EALLOW;
186     SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;
187 //     EDIS;
188
189     InitADCModule();
190
191 //     PieCtrlRegs.PIEIER1.bit.INTx7 = 1; // Timer0 Interrupt
192 //     PieCtrlRegs.PIEIER9.bit.INTx1 = 1; // Enable SCIRXINTA in PIE
193 //     PieCtrlRegs.PIEIER9.bit.INTx2 = 1; // Enable SCITXINTA in PIE
194 //     PieCtrlRegs.PIEIER1.bit.INTx6 = 1; // Enable ADCINT in PIE
195 //     PieCtrlRegs.PIEIER3.bit.INTx1 = 1; // Enable EPWM1INT in PIE
196 //     PieCtrlRegs.PIEIER3.bit.INTx2 = 1; // Enable EPWM2INT in PIE
197 //     PieCtrlRegs.PIEIER3.bit.INTx3 = 1; // Enable EPWM3INT in PIE

```

```

198 //      PieCtrlRegs.PIEIER8.bit.INTx1 = 1; // Enable I2C in PIE
199
200 //      IER |= M_INT1; // Enable CPU Interrupt block 1 – ADC, Timer0
201 //      IER |= M_INT3; // Enable CPU Interrupt block 3 – EPWM
202 //      IER |= M_INT8; // Enable CPU Interrupt block 8 – I2C
203 //      IER |= M_INT9; // Enable CPU Interrupt block 9 – SCI
204      EINT;           // Enable Global interrupt INTM
205      ERIM;           // Enable Global realtime interrupt DBGM
206
207
208 // Step 5. User specific code:
209 //      LoopCount = 0;
210 //      ErrorCount = 0;
211 //      ConversionCount = 0;
212 //      XmitFlag = 0;
213      PhaseStatus = 0;
214      PhaseCount = 0;
215      for (i=0; i<BUF_SIZE; i++)
216      {
217          SampleTable[i] = 0;
218      }
219
220
221
222      scia_fifo_init_async(); // Initialize the SCI FIFO
223      scia_echoback_init(); // Initalize SCI for echoback
224
225 //Kill all PWMs initially.
226      EALLOW;
227      EPwm1Regs.TZFRC.bit.OST = 1; //Kill PWM1
228      EPwm2Regs.TZFRC.bit.OST = 1; //Kill PWM2
229      EPwm3Regs.TZFRC.bit.OST = 1; //Kill PWM3
230      EPwm4Regs.TZFRC.bit.OST = 1; //Kill PWM3
231      EPwm5Regs.TZFRC.bit.OST = 1; //Kill PWM3
232
233
234      msg = "\r\nWelcome to PowerControl! \r\nHopefully this controller will
235          work!\r\n\0";
236      scia_msg(msg);
237
238 //Configure GPIOs
239      GpioCtrlRegs.GPAPUD.bit.GPIO0 = 0; // Enable pull-up on GPIO0
240      GpioCtrlRegs.GPAPUD.bit.GPIO1 = 0; // Enable pull-up on GPIO1
241      GpioCtrlRegs.GPAPUD.bit.GPIO2 = 0; // Enable pull-up on GPIO2
242      GpioCtrlRegs.GPAPUD.bit.GPIO3 = 0; // Enable pull-up on GPIO3
243      GpioCtrlRegs.GPAPUD.bit.GPIO4 = 0;
244      GpioCtrlRegs.GPAPUD.bit.GPIO6 = 0;
245
246      GpioCtrlRegs.GPADIR.bit.GPIO0 = 1; // Set as output
247      GpioCtrlRegs.GPADIR.bit.GPIO1 = 1; // Set as output
248      GpioCtrlRegs.GPADIR.bit.GPIO2 = 1; // Set as output
249      GpioCtrlRegs.GPADIR.bit.GPIO3 = 1; // Set as output
250      GpioCtrlRegs.GPADIR.bit.GPIO4 = 1; // Set as output
251      GpioCtrlRegs.GPADIR.bit.GPIO6 = 1; // Set as output

```

```

251
252
253 GpioDataRegs.GPACLEAR.all = 0x0F;
254
255     GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 0;    // Configure GPIO0 as GPIO //EPWM1A
256     GpioCtrlRegs.GPAMUX1.bit.GPIO1 = 0;    // Configure GPIO1 as GPIO //EPWM1B
257 GpioCtrlRegs.GPAMUX1.bit.GPIO2 = 0;    // Configure GPIO2 as GPIO //EPWM1A
258     GpioCtrlRegs.GPAMUX1.bit.GPIO3 = 0;    // Configure GPIO3 as GPIO //EPWM1B
259     GpioCtrlRegs.GPAMUX1.bit.GPIO4 = 0;
260     GpioCtrlRegs.GPAMUX1.bit.GPIO6 = 0;
261
262 // Start SEQ1
263 AdcRegs.ADCTRL2.all = 0x2000;
264
265
266 idx = 9;
267 xint = 0;
268 S1old = 0;
269 S2old = 0;
270 dS1 = 0;
271 dS2 = 0;
272 u1old1 = 0;
273 u1old2 = 0;
274 S1 = 0;
275 S2 = 0;
276 ramp1 = 0;
277 ramp2 = 100;
278 adcV1 = 0;
279 adcV2 = 0;
280 adcI1 = 0;
281 adcI2 = 0;
282 x6 = 0;
283 x7 = 0;
284 c0 = 0;
285 c1 = 0;
286 while(1) //Infinite Loop
287 {
288     if (idx++ > 7)
289         idx = 0;
290     adcP = &adcbuffer[4*idx];
291     adcV1 = AdcMirror.ADCRESULT0;
292     adcI1 = AdcMirror.ADCRESULT1;
293     adcV2 = AdcMirror.ADCRESULT2;
294     adcI2 = AdcMirror.ADCRESULT3;
295     //ADC freerunning. Latest sample will be in ADCMirror.ADCRESULTn (Use
296         mirror for fastest response times)
297     u1 = Vref - (adcV1 + adcV2)/2;
298 //     adcOld = AdcMirror.ADCRESULT0;
299     *(adcP++) = adcV1;
300     *(adcP++) = adcI1;
301     *(adcP++) = adcV2;
302     *(adcP++) = adcI2;
303 //     a5 = alpha5*Sold2;

```

```

304 //      a4 = alpha4*Sold1;
305      a2 = alpha2*u1old1;
306      a3 = alpha3*xint;
307      xint = xint + (u1 >> 4);
308      if( xint > 0x00004000){
309          xint = 0x00004000;
310      }
311      if( xint < -0x00004000){
312          xint = -0x00004000;
313      }
314      a1 = alpha1*u1;
315 //      idx1++;
316      S = a1+a2+a3; //+a4+a5;
317      S1 = S + alpha4*ramp1 + alpha6*x6 + alphaC1*c1;
318      u1old1 = u1;
319      if(S1 > kappa){
320 //          if((idx & 1) == 0){
321              GpioDataRegs.GPASET.all = 0x05; //0x05; //Phase 2 for monitoring
322              EPwm1Regs.TBCTR = 0;
323              c0 = 1;
324 //          if(S1old < kappa)
325              ramp1 = 0;
326          }
327          if(S1 < negkappa){
328 //              else{
329 //                  GpioDataRegs.GPACLEAR.all = 0x05; //Phase 2 for monitoring
330                  GpioDataRegs.GPACLEAR.all = 0x05; //Phase 2 for monitoring.
331 //                  x6 = taud1;
332                  c0 = 0;
333              }
334              S2 = 0; //S + alpha5*ramp2 + alpha7*x7 + alphaC0*c0;
335              if(S2 > kappa){
336 //                  if((idx & 1) == 0){
337                      GpioDataRegs.GPASET.all = 0x50; //Phase 4 for monitoring
338                      EPwm1Regs.TBCTR = 0;
339                      c1 = 1;
340 //                  if(S2old < kappa)
341                      ramp2 = 0;
342                  }
343                  if(S2 < negkappa){
344 //                      else{
345                          GpioDataRegs.GPACLEAR.all = 0x50; //Phase 4 for monitoring.
346 //                          x7 = taud2;
347                          c1 = 0;
348 //                          ramp2 += 1;
349                      }
350                      dS1 = S1 - S1old;
351                      dS2 = S2 - S2old;
352                      if(dS1 <= 0)
353                          x6 -= 1;
354                      else
355                          x6 = taud1;
356                      if(dS2 <= 0)
357                          x7 -= 1;

```

```

358     else
359         x7 = taud2;
360         ramp1++; // = EPwm1Regs.TBCTR;
361         ramp2++; // = EPwm2Regs.TBCTR;
362         if( x6 < -0x04000000 ){
363             x6 = -0x04000000;
364         }
365         if( x7 < -0x04000000 ){
366             x7 = -0x04000000;
367         }
368         S1old = S1;
369         S2old = S2;
370     }
371 }
372 }
373
374
375
376
377
378
379
380 //=====
381 // No more.
382 //=====

```