Single and Multi-Phase Synchronous Buck Converter Voltage Control with Modeling, PID Controller Design, and Digital Implementation

by

Benjamin Keaton Rhea

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

Auburn, Alabama
August 2, 2014

Keywords: Discrete Control, Synchronous Buck Converter, Efficiency

Approved by

Robert Dean, Associate Professor of Electrical and Computer Engineering
John Hung, Professor of Electrical and Computer Engineering
Stuart Wentworth, Associate Professor of Electrical and Computer Engineering

Abstract

This paper covers the modeling and design of a digital controller for a single and multi-phase synchronous buck converter. First, a general overview of the power switching point-of-load (POL) converters will be presented. This will be accompanied by a general overview of a digital system which explains the differences between discrete and continuous timed domains. This system overview will look at the primary components of a digital system which includes the plant, sampler, reconstruction block as well as the control system block. With a general idea of a discrete timed system, a plant model must be developed. This model is constructed from a single synchronous buck converter's circuit schematic. From this, the equations defining the converter are solved and converted into a block diagram. This block diagram is used to develop a discrete and continuous timed PID controller for the system using Matlab's Simulink. This single phase simulation is expanded to higher phase counts. Finally this digital controller is implemented on hardware. The controlling algorithm is programmed into a microcontroller and tested on single and multi-phase converters.

Acknowledgments

I would like to thank my parents, Jeff and Lori for their support and patience. Dr. Robert Dean for his leadership and guidance throughout the project. Dr. Christopher Wilson for sharing his knowledge of programming, control systems and software simulations relevant to this project. Dr. John Hung for control system simulation guidance during this project. I would like to thank my girlfriend, Carroll Lowery for her support throughout the undergraduate and graduate program. Finally, I would like to acknowledge Auburn University and all of its faculty who have contributed to completing this project.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

This paper focuses on the controlling of a multi-stage Point-of-Load (POL) converter using GaN power switching devices. The design uses multiple 12-1 V DC-DC step-down synchronous buck power converters which are optimized for efficiency. It is desirable to reduce extra heat generation near the load, reduce the overall power dissipation and lessen the cooling requirement. This work focuses on low voltage applications with a high current demand, such as modern processors. As the load current increases for a buck converter operating in continuous conduction mode (CCM) operation, the efficiency of the converter steadily declines. The CCM operation is more suitable for higher current applications where as the discontinuous conduction mode (DCM) operation is more suitable for lower current applications [1]. Another challenge of operating at low voltage and high current in buck converters is that efficiency decreases as the output voltage drops [2]. Additionally the high step down voltage requires a small duty cycle [3]. This requires the power switching devices to be addressed. Switching losses can be reduced by using GaN high electron mobility transistors (HEMT) as switches in place of silicon switches [4].

The optimum operational frequency and load current for the buck converter is directly related to the total losses in the converter. As the switching frequency increases, the voltage ripple decreases. However, this does not take into account efficiency [5]. One solution for increased efficiency at higher loads is to use multiple buck phases [6]. These phases can be interleaved to reduce the voltage ripple and the current sharing maximizes the efficiency. For interleaved operation, each phase of the converter must be spaced evenly apart from one another. For each phase of operation, two driving signals are needed. One signal for the switch and one for the synchronous rectifier are needed so that the dead time between

the two signals can be modified. To address these issues, a microcontroller will be used to regulate the output voltage, evenly space the driving signals and control the dead band. The microcontroller chosen for this project is the Texas Instraments F28335 controlCard.

1.1 Point of Load Converters

A power system is typically divided into three stages of conversion when flowing from the grid to the load. These three stages, the front end, intermediate bus and the POL stages can be seen in 1.1.

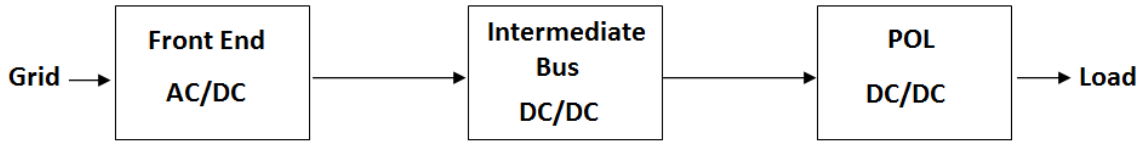

Figure 1.1: The three stages of conversion in a typical power system

The first stage, the front end, is connected directly to the grid and converts the high voltage 3-phase AC voltage to a high voltage DC voltage signal. A typical conversion would be 480 V AC converted to 400 V DC. The second stage steps down this high voltage to a lower more manageable DC signal. This would take the 400 V down to 12 V DC. The final stage is the POL converter. This could take 12 V DC down to 1 V DC when a step-down POL converter is used. The efficiency of each stage has a multiplicative relationship making each stage in the chain important. The POL stage is one of the more important stages because it is the closest to the load meaning that any extra heat generated by this stage can possibly affect the sensitive loads.

DC/DC converters have some form of a magnetic element, either a transformer or an inductor. Some types of converters are the Buck-Boost [7], Step-Down Buck [8], and

Cuk [9] configurations. The Buck-Boost is used to step voltages up. The Step-Down Buck converter converts higher voltages to lower voltages. The Cuk converter can both Step-Down and Step-Up mode. Since the goal is to convert 12 V to 1 V, the Step-Down Buck converter is chosen. With an emphasis placed on maximizing efficiency rather than reducing cost, the synchronous buck converter is chosen because it replaces the diode with a rectifying switch [10].

1.2 Synchronous Buck Converter Topology

This buck converter is used to convert the input voltage of 12 V down to 1 V output. A schematic of a synchronous buck converter where GaN is used as the power device can be seen in Fig. 1.2. This converter has one switch and two synchronous rectifiers (SR) in parallel to lower the on-resistance. This is done to increase efficiency of the converter by reducing the switching losses found in the rectifiers. The values of the capacitor and inductor as well as their respective resistances are summarized in the Table 1.1.

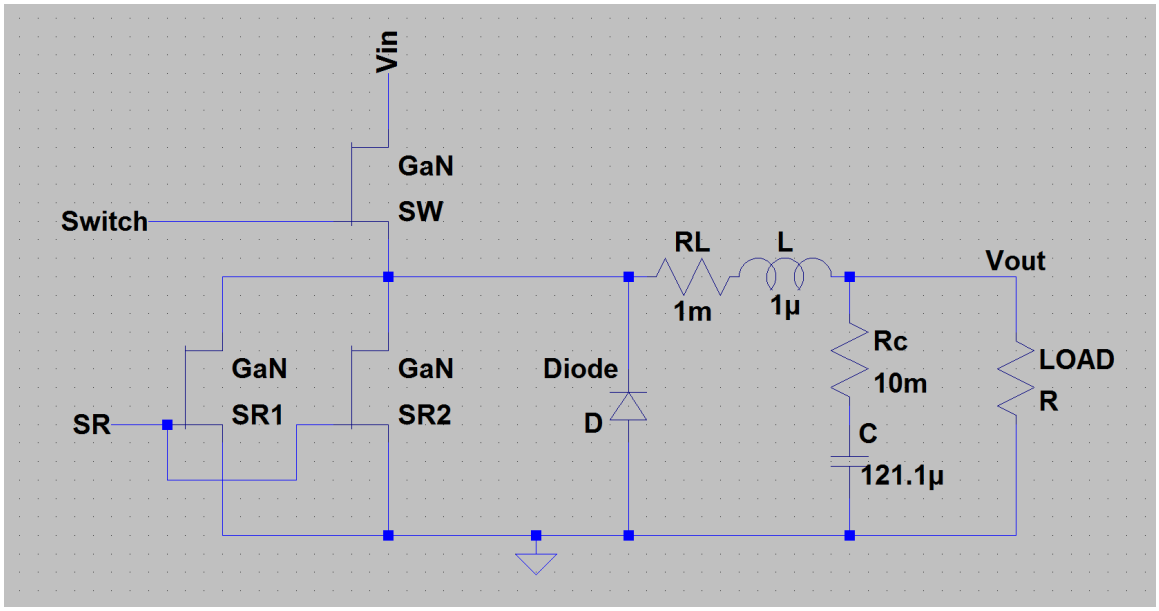

Figure 1.2: The schematic of the Synchronous Buck Converter

Parameter	Properties Value	Units
Vin	12	V
Vout	1	V
L	1	uH
C	121.1	uF
R_C	10	mOhm
R_L	1	mOhm

Table 1.1: Summary of the values of the parts used in the converter

Numerous inductor values and manufacturers were tested with the goal of maximizing the efficiency of the converter. The majority of the inductor values tested ranged from 0.22 uH up to 3 uH. This was an iterative process where the board layout was modified each time to minimize switching and parasitic losses. Layout used anywhere from two up to five GaN power switching devices. Typically only one switch was used but there were anywhere from one to four synchronous rectifiers. To minimize switching losses, the power devices were brought closer together, as seen in Fig. 1.3 and Fig. 1.4.

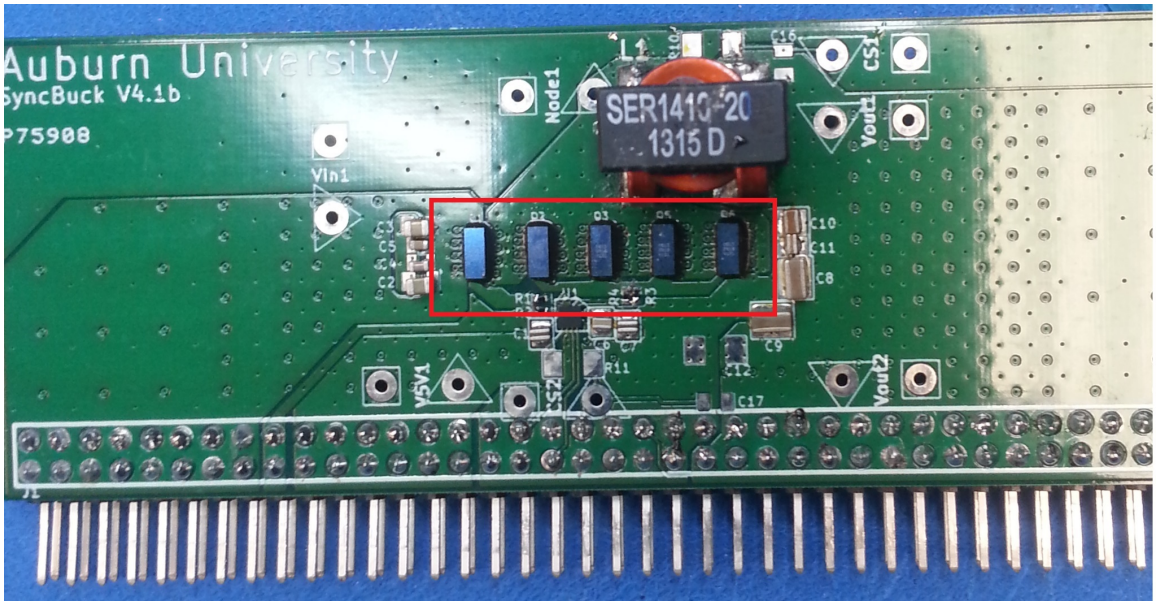

Figure 1.3: Board layout of an early board iteration with spaced power devices

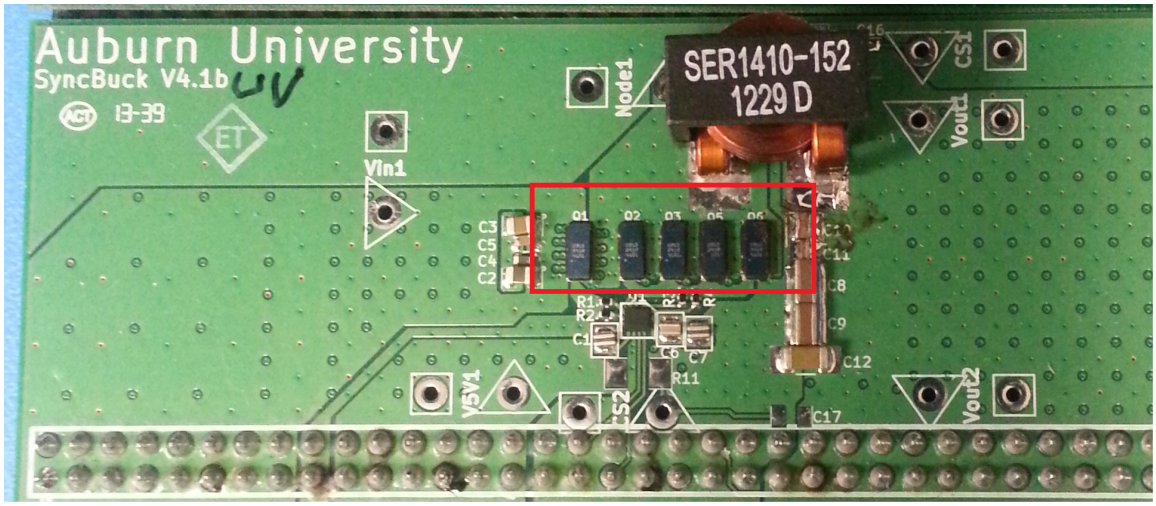

Figure 1.4: Board layout of an later board iteration with closer together power devices

1.3 Previous Efficiency Efforts

With the primary objective of this project being focused on efficiency, a few previous efforts involving this POL converter will be presented. A lot of research went into improving the efficiency GaN HEMT power switching converters. One of the popular solutions is to increase the switching frequency beyond 1 MHz [11]. This is helpful because the size of the inductor and capacitor is proportional to the switching frequency. As the switching frequency increases, the size of the these magnetic components decreases which allows for the converter to take up less space. The major drawback to this solution is the added stress placed on the power switching device [12]. This is one of the reasons that GaN is used over other typical power switches.

The effectiveness of this approach can be seen in a similar 12 to 1 V synchronous buck converter which operates at 5 MHz [13]. Here the authors were able to estimate that close to 90% efficiency by using GaN as the power switching devices was possible using the parameters from data sheets. While the higher frequency approach was considered, it was not pursued in this particular project. Instead a lower switching frequency, ranging from 100 - 500 kHz, was chosen. This was done for reliability reasons since the higher switching frequencies put

more strain on the power devices and very little reliability testing had been performed for GaN technologies at the time. This was also an area of research in GaN that had not been thoroughly investigated.

1.4 Transient Analysis

Some of the most important factors in a switched mode power supply are the efficiency, power density, cost of manufacturing and the transient response. The transient response is how long it takes the output signal to settle to steady state when a disturbance is applied. When operating in voltage controlled mode, this refers to the amount of time it takes for the voltage to settle at the reference value when the output load current is changed. Another aspect of transient analysis is the overshoot or undershoot when a disturbance is applied. In our case, this is how much the voltage drops with a current increase.

A limiting factor to DC/DC converters transient response is related to the conversion ratio between the input and output [14]. This means as the ratio of input voltage to the output voltage grows, the transient response tends to degrade with the same circuit topology and parameters. For example, a 3.3 V to 1 V conversion would be considered on the smaller side and a 12 V to 1 V conversion would be on the higher side. The higher ratios tend to be subject to a larger initial voltage drop when a transient is applied.

One of the solutions to this problem is to increase the output capacitance [15]. This can help mitigate the initial voltage drop. This physical limiting factor in this approach is the ESR of the added capacitors [16]. It has also been shown that the addition of capacitors can help a variety of simulated control strategies [17]. These methods will be experimentally confirmed later in this paper.

Chapter 2

Digital System Overview

There has been a growing effort to improve digital control of switching power converters instead of the conventional analog methods [18],[19]. For this reason, a microcontroller was chosen to control the converters. The selection of the microcontroller is application specific [20]. This project requires at least ten different PWM signals, ten ADC channels to sample the output current and voltage of each phase, and enough processing power to run the controlling algorithm with potential interruptions from a serial input.

All of this means that the full process of discrete time design must be considered. A general digital control system can be seen in Fig. 2.1. This system consists of a discrete timed control algorithm, a reconstruction element as well as a sampler. In this diagram, the process model (also known as the plant) is the portion that is to be controlled. This process exists and is continuous for all time. For this process to be controlled digitally, the plant must be periodically sampled. This is a conversion from analog to digital (ADC). This sampled data then can be compared to the desired reference value and fed into the digital control algorithm. The digital algorithm must then be reconstructed into an analog signal so it can be applied to the process. This process is often called a digital to analog conversion (DAC).

2.1 Reconstruction

One way to reconstruct these samples is to use a zero-order hold [21]. This is a way to model how a digital-to-analog (DAC) would operate. The sample of the signal will be held for one sample interval. A simulation diagram using Simulink is developed to demonstrate the difference between the continuous time signal and the ZOH which is shown in Fig 2.2.

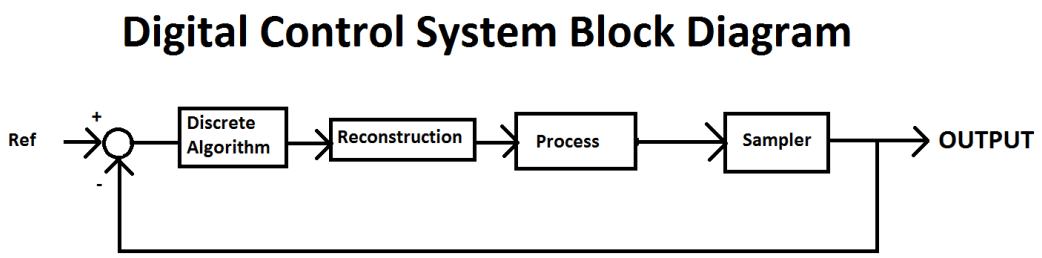

Figure 2.1: A generalized discrete system block diagram

A continuous sine wave is generated and fed through a ZOH block where the input and the output of the block are compared real time with a scope. These values are also saved as variables in the workspace to generate the plot.

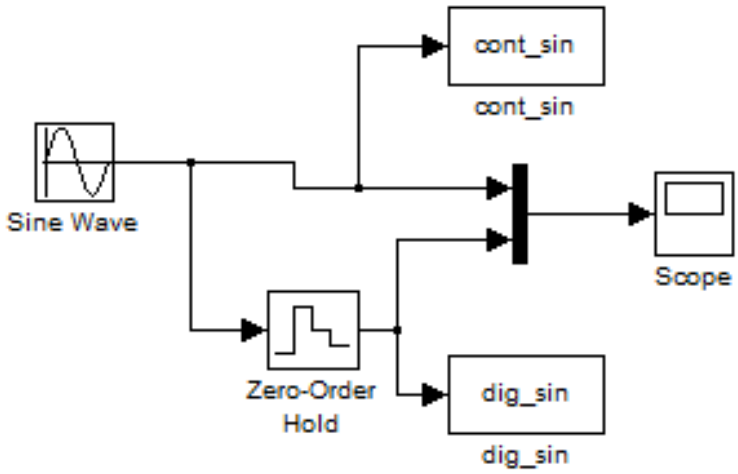

Figure 2.2: Simulation block diagram to show difference in a continous time signal and a ZOH signal

The results of this simulation, seen in Fig. 2.3, show how the value of each sample is held for one sample interval. This shows that the number of samples taken would not be sufficient to replace the continuous sine signal. If more samples were taken then the signal would more closely resemble the continuous time signal. The key element that is taken

from this simulation is that the zero order hold has a delay. Since this is a sine wave, the reconstruction of the green stair step function would have a slight phase delay.

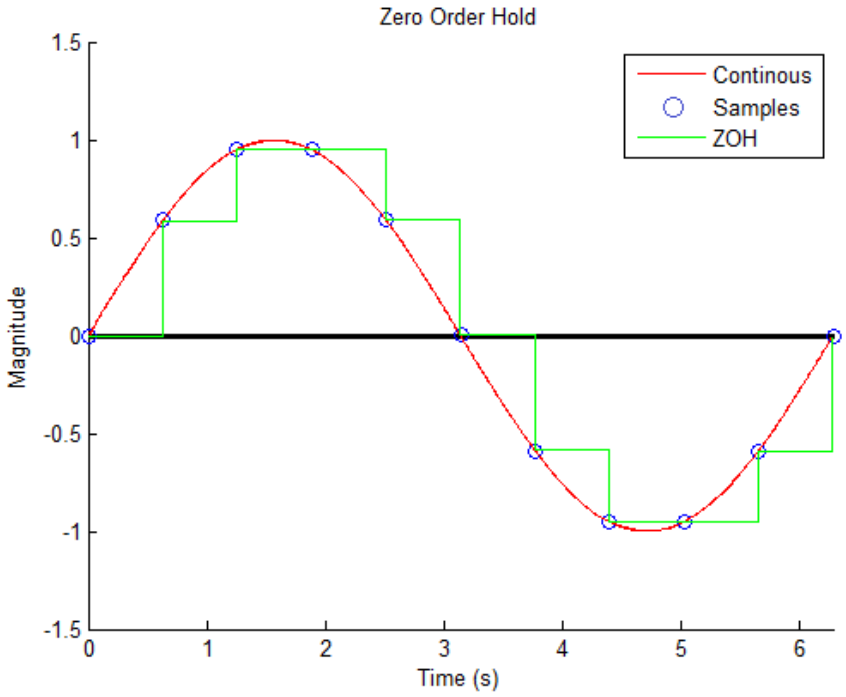

Figure 2.3: Example of a zero order hold

Another way to reconstruct the digital signal to an analog signal that is popular in power electronics is the use of pulse width modulation (PWM). An ideal PWM signal has two possible values: ON and OFF. An ideal PWM signal takes no time to transition from these two different states. Switching between these two values creates a periodic square wave. This square pulse train is characterized by the amount of time the signal is ON verses the total period. This expression is seen in Eq. 2.1 where the duty cycle is expressed as a percentage which is equal to the ratio of the time on to the full period. These values are illustrated in Eq. 2.4, where a single period of the PWM signal is shown.

$$Duty(\%) = \frac{Ton}{T} \tag{2.1}$$

$$T = \frac{1}{f} \tag{2.2}$$

The buck converter's efficiency is optimized at 170 kHz which means that the length of one period is aproximetly 5.88 μs, as seen in the inverse relationship in Eq. 2.2. The converter is designed to convert 12 V to 1 V which means that the ratio of 12:1 is aproximetly 0.0833 or 8.33%. This value is how long the switch needs to be ON to convert the 12 V to 1 V. This makes the typical duty cycles for the switch around 8% while the synchronous rectifier will be the opposite making it operate around 90%.

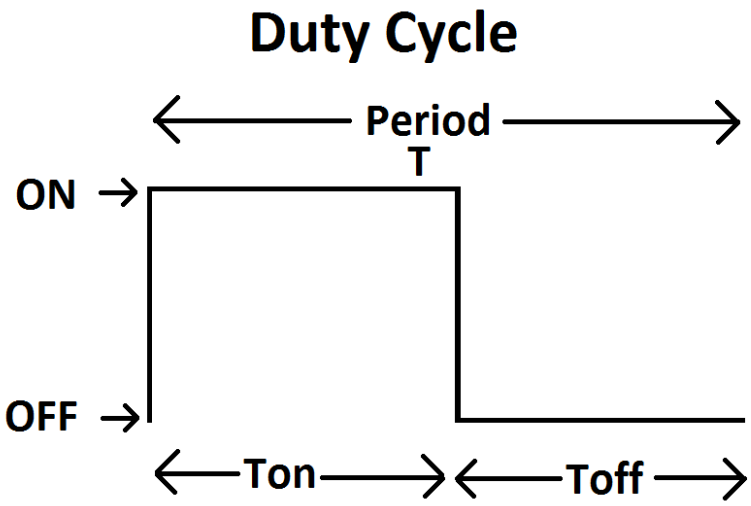

Figure 2.4: A single period of a PWM signal showing the on and off times that are used to calculate duty cycle

One PWM signal and its inverse could be used for the switching and synchronous rectifier signals as depicted in Fig. 2.5. This simplifies the design by only requiring a single PWM for each phase of the converter.

However, this is not preferred when applied to real world applications. The PWM signals are not ideal so that the switch and synchronous rectifier could be ON at the same time which allows for possible shoot through current on the power device which often leads to failure of the device For this reason, two PWM signals are required for each POL converter

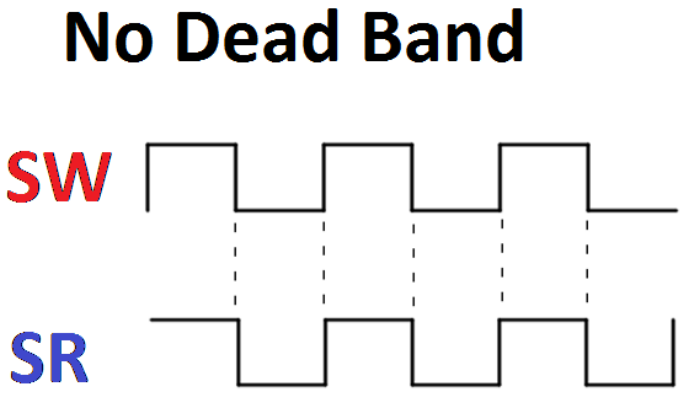

Figure 2.5: Two PWM signals, the switch (red) and the synchronous rectifier (blue), which are switched at the same time

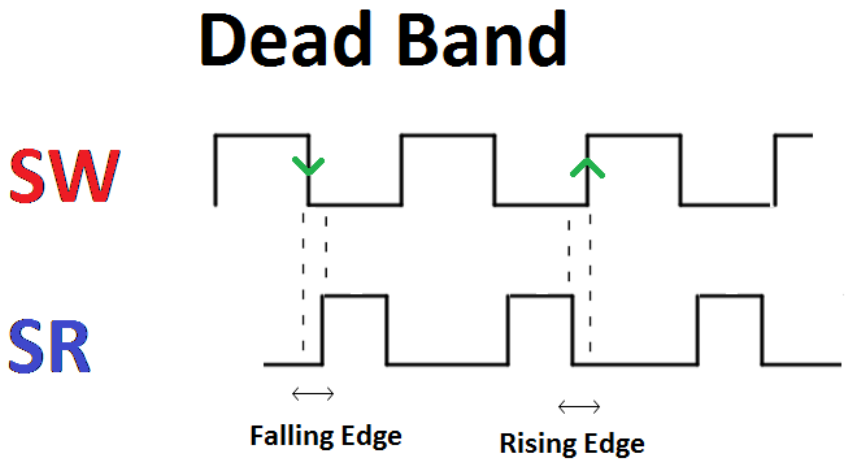

Figure 2.6: Example of dead band between two PWM signals, the switch (red) and the synchronous rectifier (blue), which is shown by the dotted lines on both the rising and the falling edges. This shows that the switch and the synchronous rectifier are never switching at the same time and can never be ON at the same time

to control the dead band, a period where both the signals are OFF. An example of a dead band can be seen in Fig. 2.6. It should be noted that a dead band is typically very small and that the figure over exaggerates this to make it easier to see that the dead band occurs when both the switch and synchronous rectifiers are OFF. If the signals were used in this application, it would greatly decrease the efficiency of the converter since both signals would be off at the same time. In this example the synchronous rectifier is slightly delayed from the switching signal so that the two PWM signals do not line up on top of each other. The main drawback to this approach is that a specific PWM signal is required for each switch and synchronous rectifier of each phase, effectively doubling the number of required PWM signals. Another negative effect of this approach is that it decreases efficiency since there is no power conducted when both the switch and synchronous rectifier are OFF. Since the dead band is a very short amount of time, the loss in efficiency is negligible and the added benefit of reliability in the power device is gained.

2.2 Sampler

Another aspect to a digital system is the sampler. The output of the plant model is a continuous time signal which means that it needs to be converted to a digital signal to be processed by the digital algorithm. An example of this can been seen in Fig. 2.7, where there is one full period of a continuous sine function. In this example, the red line is the continuous time sine function and the blue circles are the ten samples taken of the function. These samples are represented as impulse functions.

These samples are then able to be stored in memory as a discrete value. The magnitude and the time values are both needed, so an array would be appropriate. Since this is an ideal case, the value sampled is assumed to be exactly the value of the continuous sine signal. This is no always the case in application due to outside interference and nonlinearities in the software conversion. This can be supplemented by taking more samples, like four or five, and then averaging them together to get a more accurate value. This requires the sampler to

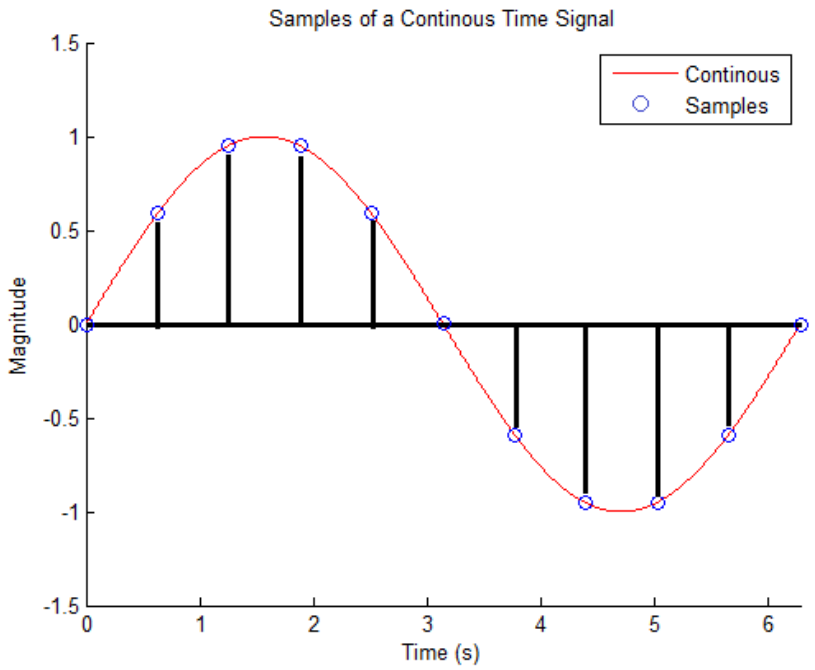

Figure 2.7: Example of ideal sampled data from a continuous function

take samples at four to five times the rate of the ideal example as well as be able to compute the average and be ready for the next set of samples in this amount of time. This means that better hardware is required for this approach.

2.3 Discrete Time Algorithm

The chosen method for controlling the buck converter is using a proportional-integral-derivative (PID) controller. The PID controller is a popular choice for the buck converter [22]. The typical layout of a PID controller is shown in Fig. 2.8.

As the name implies, the PID controller has a proportional or multiplicity term, an integral term and a derivative term. Each of these terms contains a certain weight which is modeled as the gain blocks K_p, K_i and K_d in Fig. 2.8. These are arbitrarily chosen to be 1, 2 and 3, but the exact values will need to be determined in the simulations shown later in the paper. The proportional term is associated with the overall gain of the controller.

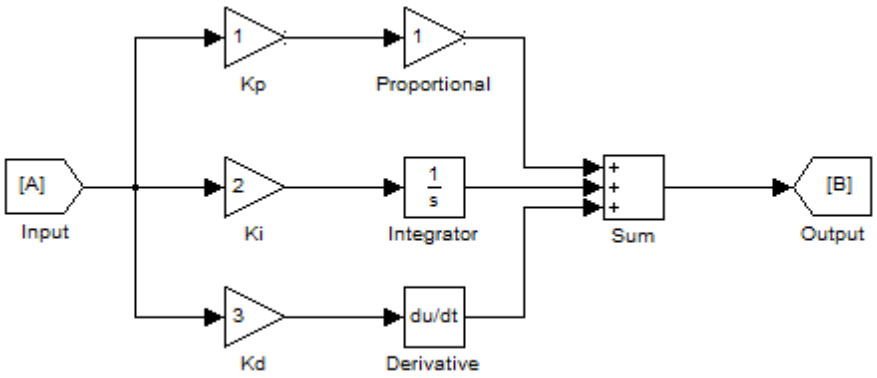

Figure 2.8: PID controller configuration with weighted gains for each of the proportional, integral and derivative terms

The integral term is an accumulation of the error over time. The integral term is related to the system's steady state error. This error is when the system's output settles at a value different than the desired set point value. To eliminate steady state error, a model of the process must be included in the controller. The final term, derivative, accounts for the rate of change of the input signal.

Chapter 3

Model of the Buck Converter

Before any controller development can begin, there needs to be accurate model of the buck converter for simulations. First a single buck phase will be considered, which will make the expansion to a multiple phase model easier. This will be done by looking at the circuit schematic and then simplifying it for easier analysis. Then differential equations for the current running through the inductor and the voltage across the capacitor can be derived and the output voltage. These are the governing equations for the operation of the buck converter. These differential equations can be used to create a block diagram using integrator, summing and gain blocks.

Modeling of various converters follows a similar process of determining the small signal model for the mode of operation which can be calculated by hand and then confirmed by simulation [23]. The buck converter will be operating in CCM in voltage controlled mode. This model will be linearized for frequency analysis to determine the bandwidth of the system [24]. The bandwidth of the buck converter has been determined to be one of the limiting factors in the performance of the converter's transient response [25]. This model will also be used to determine the PID gains in later simulations.

3.1 Simplified Schematic

To begin modeling the buck converter, a simplified schematic would be helpful. The first simplification can be to remove the diode and reduce the number of synchronous rectifiers to only one as seen in Fig. 3.1. In this simplified circuit, you can see that there are still only two possible states for this circuit since the switch and synchronous rectifier can never be on at the same time. Looking at steady state operation, the switch would need to be on 8.33%

of the time and the synchronous rectifier would be on for the remainder of that time with out factoring in the small dead time. When the switch is ON, the input current is able to flow directly through the inductor and then the rest of the circuit like in Fig. 3.2. This is the conducting state of the converter. When the synchronous rectifier is high, the circuit is in recovery mode as seen in Fig. 3.3.

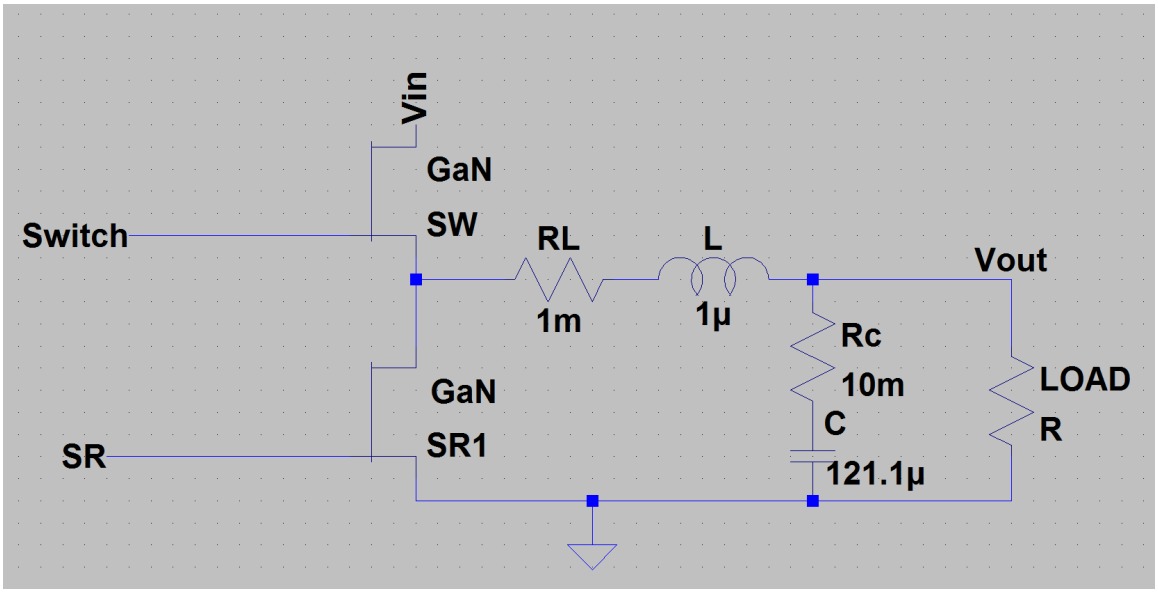

Figure 3.1: Simplified circuit schematic of buck converter for a model

When looking at both of these states, the only time the input voltage is seen by the load is when the switch is high. This effectively means, from the load's point of view, that the synchronous rectifier and the switch can both be modeled as a single input voltage source that is multiplied times the input voltage as seen in Fig. 3.4. This factor that is multiplied by the input voltage is the amount of time the switch is ON, or the duty cycle.

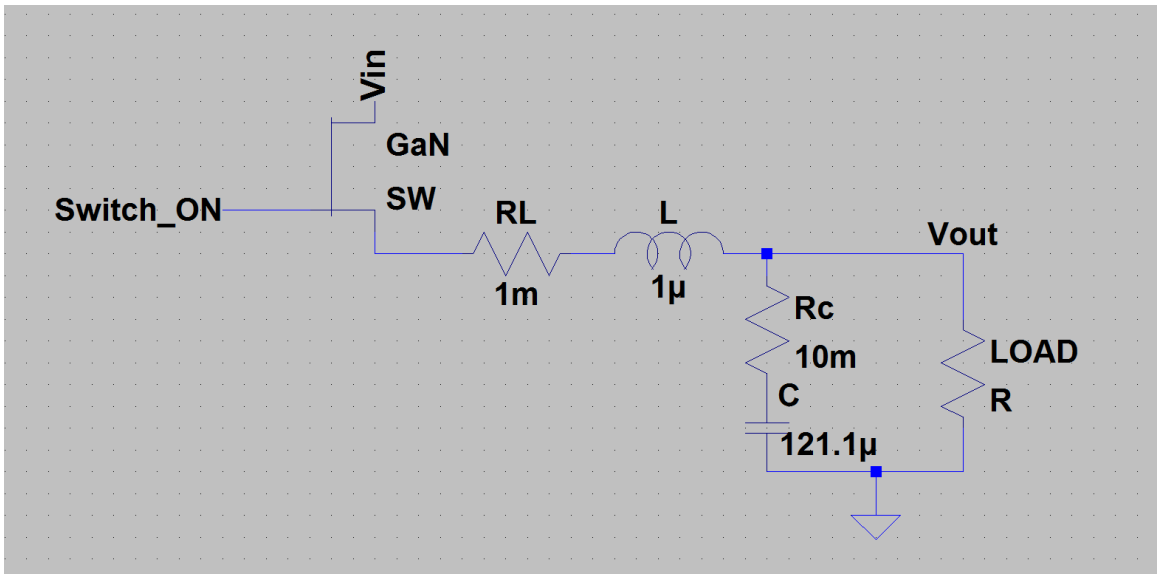

Figure 3.2: A view of the state of the converter when the switch is ON and the SR is OFF which is the conducting state of the converter

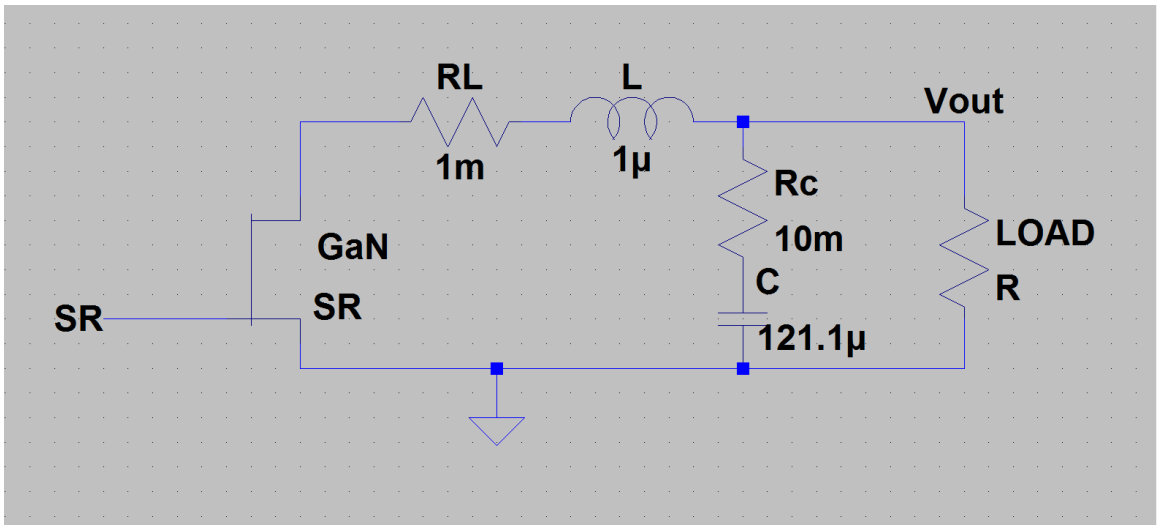

Figure 3.3: A view of the state of the converter when the SR is ON and the switch is OFF which is the recovery state of the converter

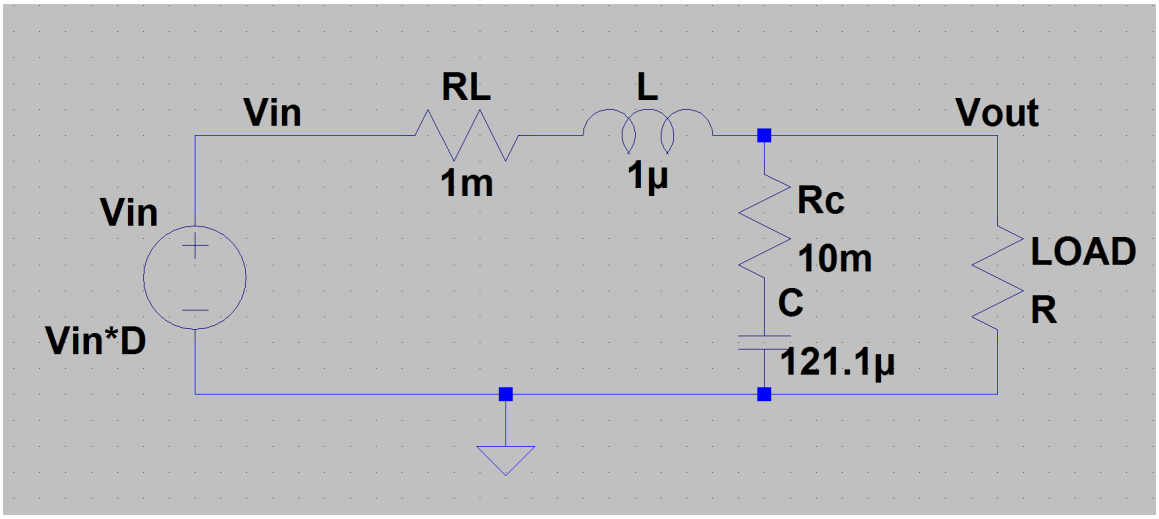

Figure 3.4: A simplified schematic derived from the two states of the converter

3.2 Equations from the Schematic

Now with this simplified model we can look at some differential equations to derive a model for the converter. All this analysis is fundamentally based on Ohms Law where the current equals the product of the voltage and impedance seen in Eq 3.1. All of this analysis will be referring to our simplified circuit schematic of the converter found in Fig. 3.4.

The first thing to look at is the rate of change of the current in the inductor where the general equation is shown in Eq. 3.2. The inductance is a constant value so it can be factored out. Solving for the voltage across the inductor is simple by summing the voltages at the node as seen in Eq. 3.3. Next, the voltage across the inductor can be subsistued by the product of the current through the inductor and the internal resistance of the inductor to get the equation in terms of current as seen in Eq 3.4. This is one of the three main equations needed to create a simulation model of the converter. Finally, the integral of both sides of the equation are taken and some algebra is done which results in the result in Eq. 3.5.

$$i = \frac{V}{R} \tag{3.1}$$

$$\frac{di_L}{dt} = \frac{1}{L}(V) \tag{3.2}$$

$$\frac{di_L}{dt} = \frac{1}{L}(V_{in}D - V_{inductor} - V_o) \tag{3.3}$$

$$\frac{di_L}{dt} = \frac{1}{L}(V_{in}D - i_L R_L - V_o) \tag{3.4}$$

$$i_L = \frac{1}{L}\int(V_{in}D - i_L R_L - V_o)dt \tag{3.5}$$

Next, the voltage across the capacitor is analyzed. The general equation for the change in voltage across the capacitor can be seen in Eq. 3.6. Here it can be seen that the change in voltage is dependent on the value of the capacitor and the change in the current through the device. Referring to the schematic in Fig. 3.4, it can be seen that the current passing through the capacitor is equal to the difference of the current leaving the inductor and the current flowing to load. This substitution can be seen in Eq. 3.7. Finally, by taking the integral to both sides and some algebra you are left with the results found int Eq. 3.8.

$$\frac{dV_C}{dt} = \frac{1}{C}(i) \tag{3.6}$$

$$\frac{dV_C}{dt} = \frac{1}{C}(i_L - i_o) \tag{3.7}$$

$$V_C = \frac{1}{C}\int (i_L - i_o)dt \tag{3.8}$$

Finally, an equation for the output voltage should be derived. Once again, by looking at Fig. 3.4, it can be seen that the load is in parallel with the capacitor and the internal resistance of the capactior. This means that the voltage across the capactior is equal to the output voltage as seen in Eq. 3.9. It is also known that the voltage drop due to the resistance in the capacitor is defined by the resistance and the current flowing through it. The current through the capacitor has already been investigated in the previous equation found back in Eq. 3.7. Substitution reveals the output voltage equation which is found in Eq 3.10.

$$V_o = V_C + V_{Rc} \tag{3.9}$$

$$V_o = V_C + R_C(i_L - i_o) \tag{3.10}$$

3.3 Block Diagram for Converter

With the equations derived in the previous section, a block diagram can be developed. The equations were simplified into indefinite integrals so that the block diagram will consist of only integral, summing and gain blocks, which is shown in Fig. 3.5. Starting with integral blocks for both the inductor and capacitor. The output of the inductor integral block is the inductor current and the output of the capactior block is the output voltage. These integral blocks are fed a signal which is scaled by the inverse of the inductor and capacitor values respectively. As defined by the equation, the integral gain block is fed by the product of the input voltage which is subtracted from the voltage drop across the inductor and the output voltage defined by Eq. 3.5. The capacitor gain is fed the difference of the inductor current and the output current from Eq. 3.8. The output current is calculated by the quotient of the output voltage and the load resistance. Finally, a parallel gain of the resistane of the capacitor is added to the capacitor integral which is from the output voltage equation Eq. 3.10.

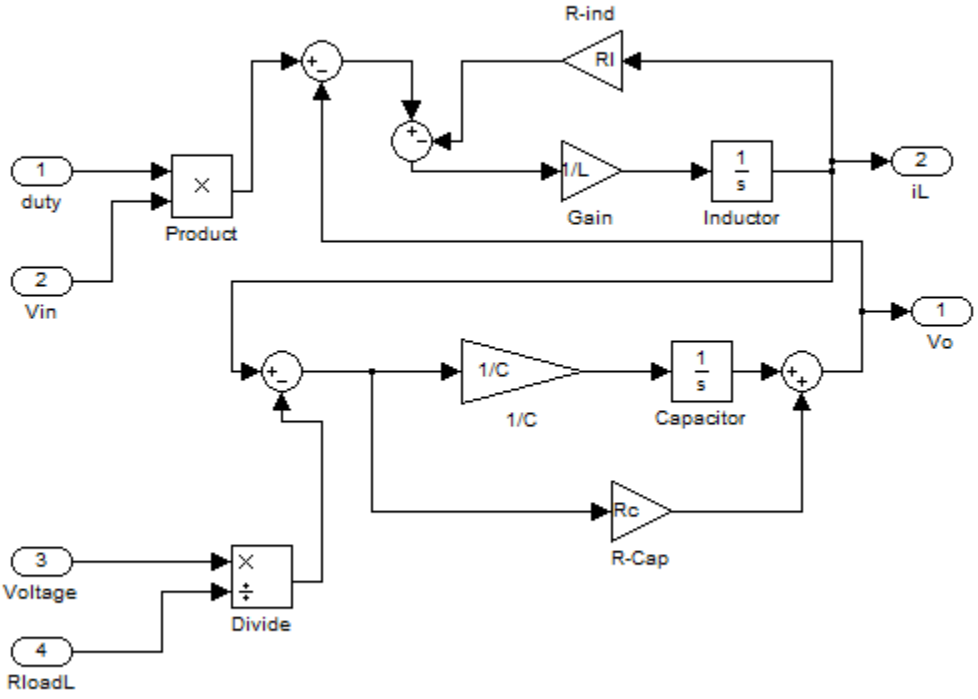

Figure 3.5: Block diagram derived from the equations of the converter

Chapter 4

Simulation

Before the controller is to be implemented on some hardware, a software simulation is needed to help determine what the PID coefficients are. A helpful tool for designing the controller is Matlab's Simulink. First the plant model, the buck converter, will be derived in the continuous time domain. This model's open loop characteristics will then be analyzed. This model will then be incorporated into Simulink where the PID controller will be developed.

4.1 Frequency Analysis Continuous Time

The stability of the converter model and the controller will be analyzed in the frequency domain. To do this, both Bode and checking the poles and zeros will be plotted of the continuous time function's open loop response. The first step is to linearize the block diagram of the model seen in Fig. 4.1.

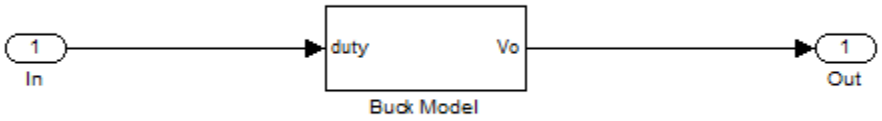

Figure 4.1: Block diagram of the converter model used to linearize the model

The buck converter is a switching circuit driven by a PWM signal, which cannot be linearized by Matlab. To get around this problem, the converter is configured to operate

with a constant input voltage, load and voltage output. This leaves the only input to the model as the duty cycle. The simplification can be seen in Fig. 4.2 which is the block diagram inside the subsystem from the previous figure.

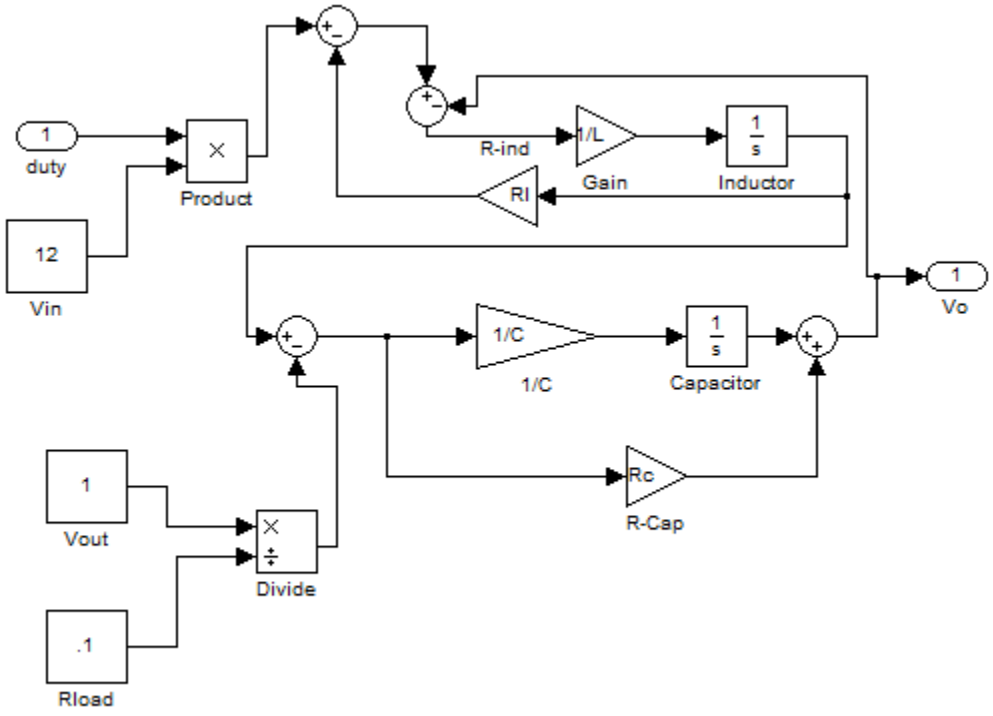

Figure 4.2: Simplified converter model with constants

To linearize the model, the script from Appendix A is run which calls the simulink model. This script first linearizes the model, then calls the A, B, C and D matrices of the state space model. Then uses this information to create a transfer function of the model. It then checks for the stability of the model and generates a Bode plot of the continuous time model seen in Fig 4.3. This plot includes the gain and phase margin of the model. The phase margin is measured at the gain cross over. The gain cross over occurs at the frequency when the magnitude plot crosses 0 dB. The model by itself has a gain margin of infinity and a phase margin of 24.4 degrees which is marked by the black line on the phase plot. Generally, a phase margin of over 60 degrees is desired. The gain margin is measured at the

phase cross over. This is when the phase plot goes below 180 degrees. In this bode plot, you can see that the phase plot never goes below the 180 degree mark, making the gain cross over infinity.

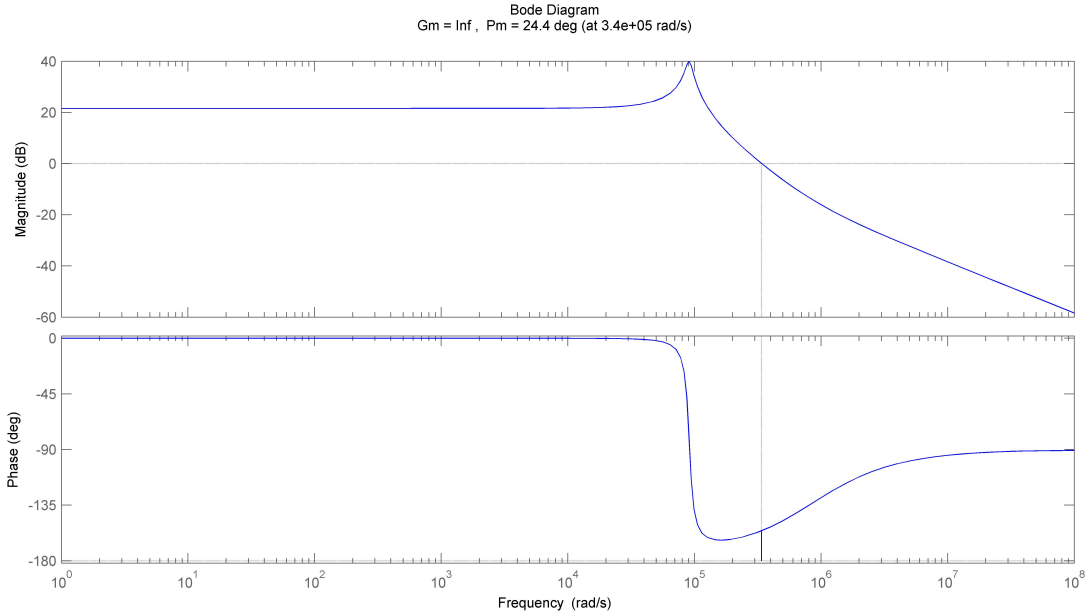

Figure 4.3: Bode plot of the buck converter model only

In a similar fashion to the model, the PID controller is modeled as a stand alone block. This results in the following frequency response seen in Fig 4.4. This diagram also displays the gain and phase margin which are both infinity for the controller by itself. This is good because it can help improve the phase margin of the converter model. The reason the phase margin will increase by using this controller is due to the high frequency gain of this controller. By increasing the high frequency gain, we are shifting the the gain cross over to a higher frequency.

To see the effect on the open loop response of the controller with the converter model, the simulation is repeated with the controller in series with the simplified converter Fig. 4.5. By using this model, the high frequency gain from the controller and the frequency characteristics of the buck model are added together. The results of this combination are seen in

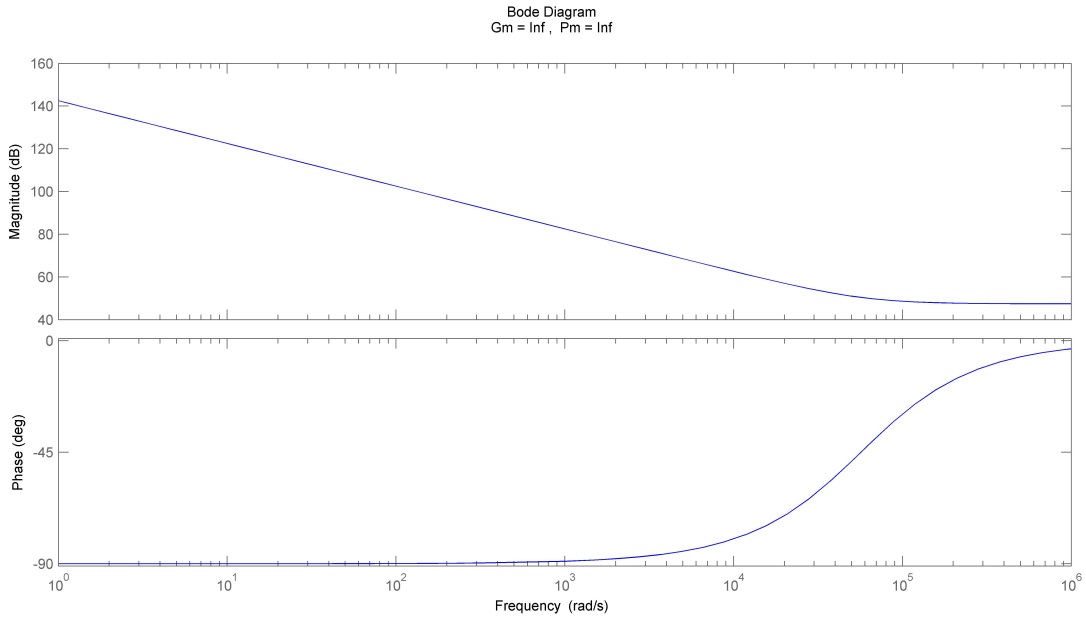

Figure 4.4: Bode plot of the controller model only

Fig. 4.6. This shifts the magnitude curve up higher making the phase cross over occur at a higher frequency. This makes the phase margin increase to 88.3 degrees.

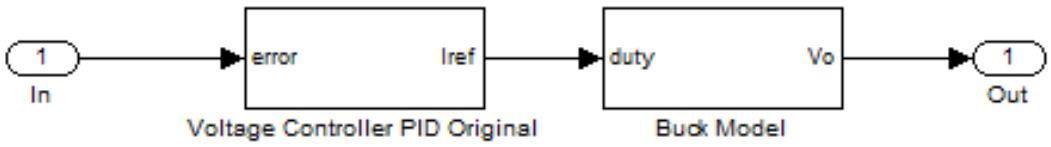

Figure 4.5: Block diagram of the controller and converter model together in series

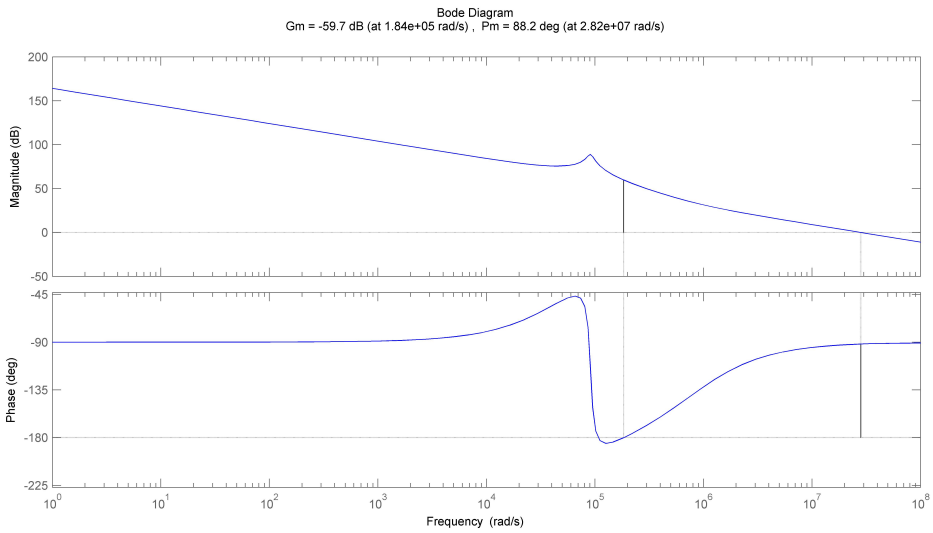

Figure 4.6: Bode plot of the controller and converter model together

4.2 Frequency Analysis Discrete Time

Now that the continuous time model has been analyzed in the frequency domain, the discrete timed model needs to be derived, which can be seen in Appendix A. Matlab has a specific command, called *c2d*, for converting continuous timed transfer functions into discrete timed functions. This command is used in the script where the sampling time is specified as well as the parameter *matched*. This parameter option tries to match the frequency characteristics of the original continuous time transfer function. This is important because we want to keep the desirable frequency response that was seen in the continuous time simulations in the discrete time domain.

First the model is converted to discrete time in the same manner as before to see the effects of the discretization process on the model. One distinct difference in the discrete model is that there is a cut off frequency. Other than this, the general shape of the original curve is maintained, as seen in Fig. 4.7. Some of the minor differences are that the peak value of the magnitude is slightly increased and rolls off much more quickly. This roll off is also present in the phase plot near the cut off point. These two changes together alter the

gain cross over which changes the phase margin to an undesirable 3.64 degrees. This is way below the desired 60 degree margin. The discretization has also reduced the gain margin from infinity to 7.21 dB. Since the gain margin is measured at the phase cross over; this is a result of the phase plots steep roll off.

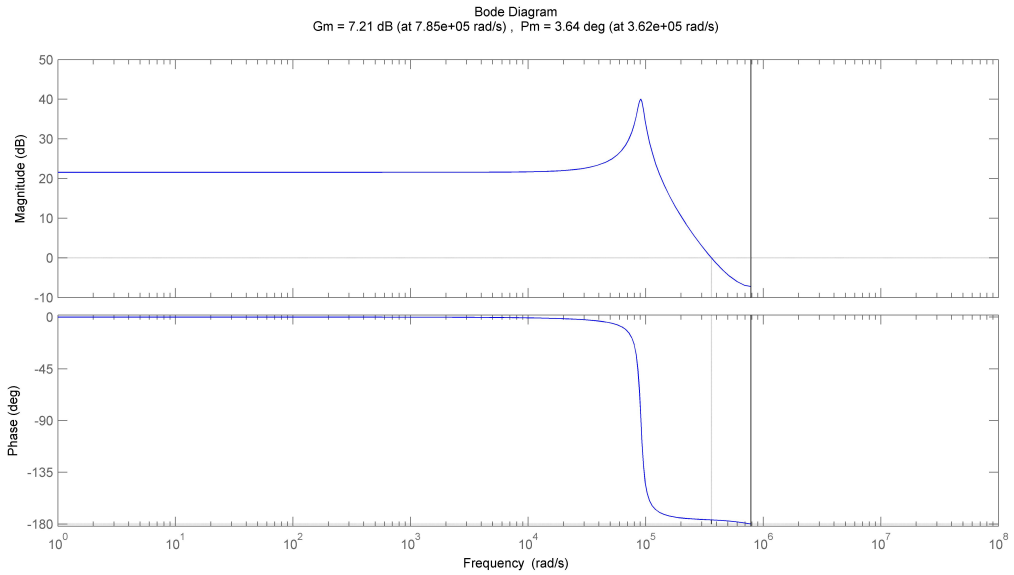

Figure 4.7: Bode plot of the converter model in discrete time

Next, the open loop response of the controller is analyzed. This will provide insight into how a discrete controller would behave when implemented in a digital system, such as a microcontroller.

In both the continuous and discrete timed results, the controller improved the phase margins considerably which is summerized in Table 4.1. The phase margin of the continuous timed results improved from 24.4 degrees to 88.2 degrees satisfying the 60 degree desired margin. In both the continuous and discrete timed results, the gain margin was significantly decreased due to the high gain of the controller. The large negative numbers mean that stability is lost by decreasing the gain.

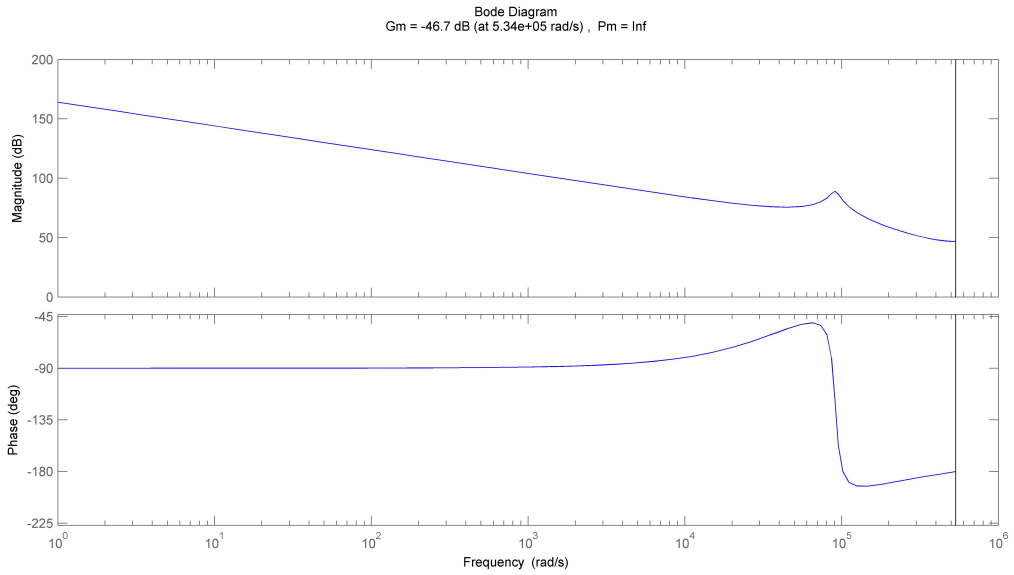

Figure 4.8: Bode plot of the controller in discrete time

	Gain Margin (dB)			Phase Margin (Deg)		
Model	Plant	Controller	Both	Plant	Controller	Both
Continuous	∞	∞	-59.7	24.4	∞	88.2
Discrete	7.21	∞	-46.7	3.64	∞	∞

Table 4.1: Summary of the gain and phase margins for the plant, controller and both together

4.3 Analysis of Single Phase Model

The simulation of the converter monitors the output voltage, the output current, and the PWM input signals. The output voltage, seen in Fig 4.9, shows the converter operating at 10 A. The start up transient settles in approximately 0.15 ms and maintains a constant average value of 1 V for the duration of the simulation, showing the stability of the controller. The output current in Fig 4.10 has an average value of 10 A and switches between approximately 12 A and 8 A. A view of the PWM input signal seen in Fig 4.11 is fed to the buck converter. A close up view of this PWM signal shows that the duty cycle of the converter is aproximetly 8.3% which is required to convert 12 V to 1 V when the system has settled at 1 V and 10 A Fig. 4.12.

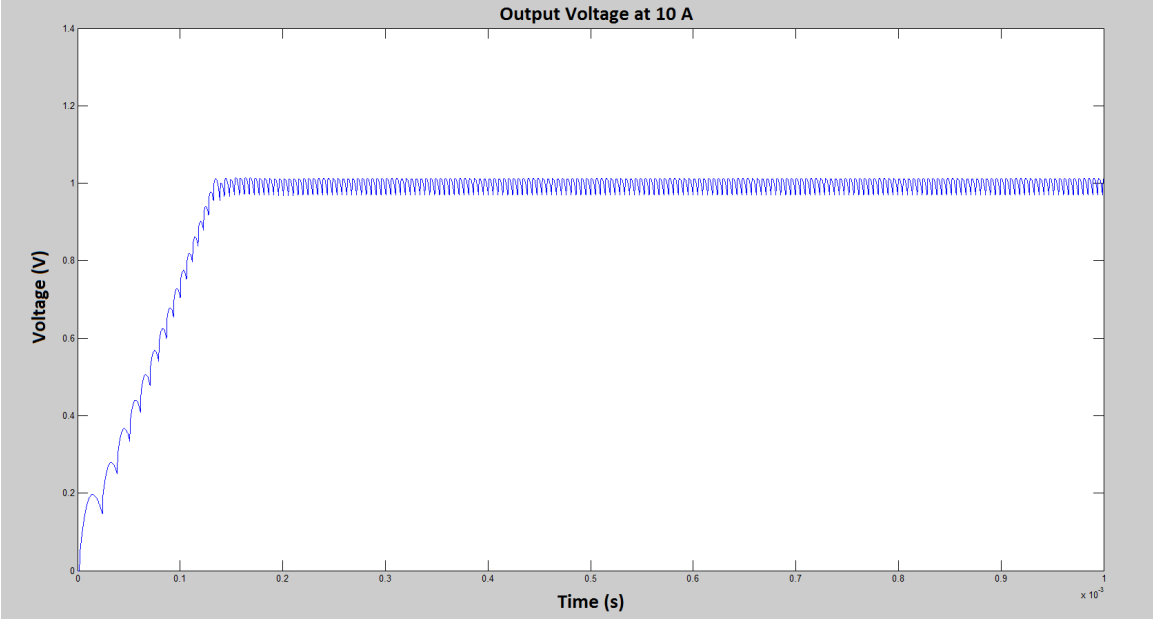

Figure 4.9: Output voltage of a single phase converter to show steady state response

Next the voltage response for a single phase is analyzed under an instantaneous 25% increase in output current. In the simulation, the current increased at approximately 0.3 ms, as seen in the output voltage plot in Fig. 4.13. A close up of the output shows that the voltage only drops approximately 80 mV after the current is increased 25% Fig. 4.14. The corresponding current is shown in Fig. 4.15 where the current increases from an average of 10 A to 12.5 A. The PWM signal only slightly deviates from the steady state operation, showing that the increase in current doesn't affect the system very much.

As a worst case simulation, the output current is instantaneously increased 100% of its original value. This doubles the output current from 10 A to 20 A which is seen in Fig. 4.19. The output voltage response is shown in Fig. 4.17 which settles out in approximately 0.1 ms. A close up of this same voltage response shows that there is approximately 15 mV seen in Fig. 4.18. This is almost double the drop from the 25% load increase where the load increased from 10 A to 12.5 A but in this case the load jumped from 10 A to 20 A. The affect of this larger load increase is seen in the PWM signal as well, in Fig. 4.20. Here around

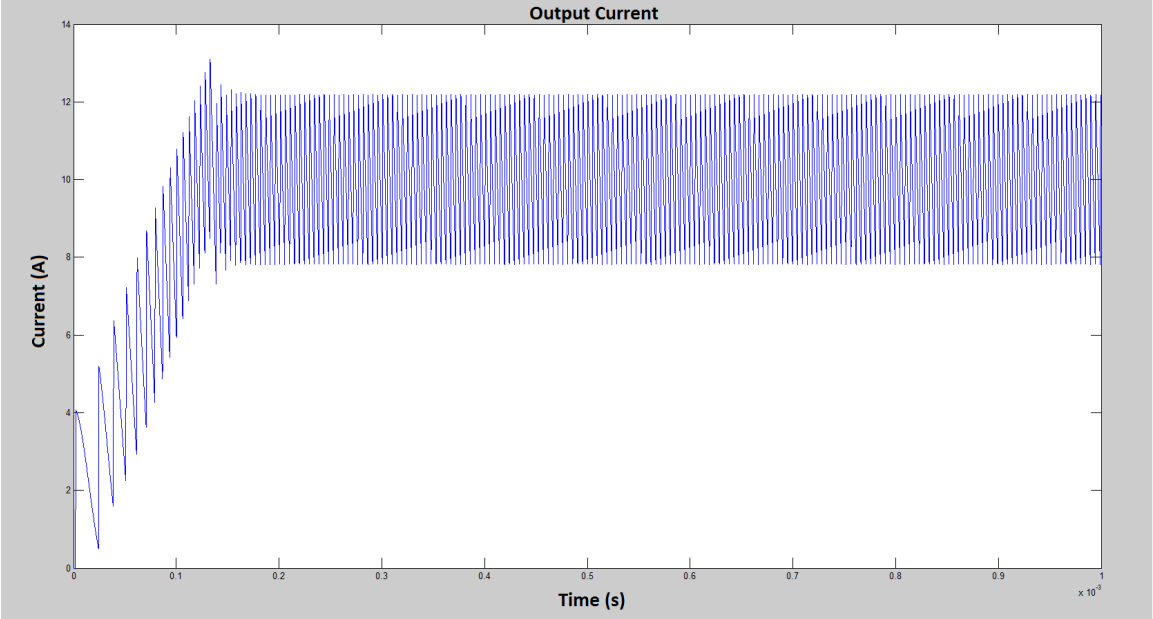

Figure 4.10: Output Current of a single phase converter to show steady state response

the 0.31 ms mark, the system greatly increases the duty cycle to almost double the steady state value of 8.3%. This increase is only for one period before the system settles back out.

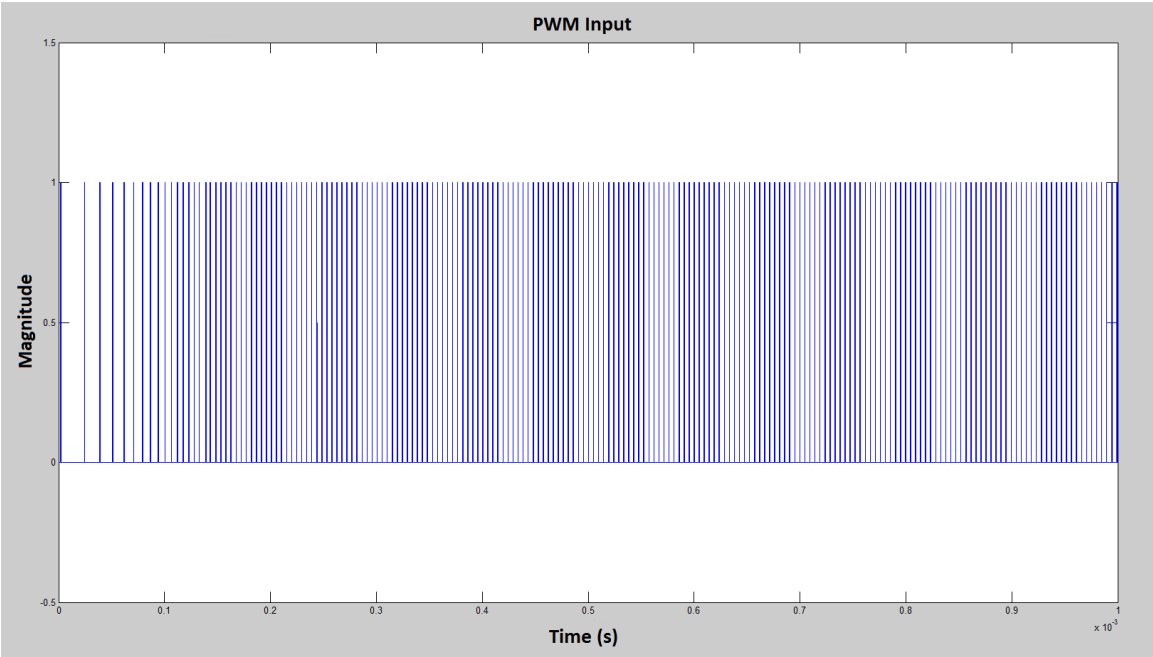

Figure 4.11: PWM input signal over the duration of the simulation of a single phase

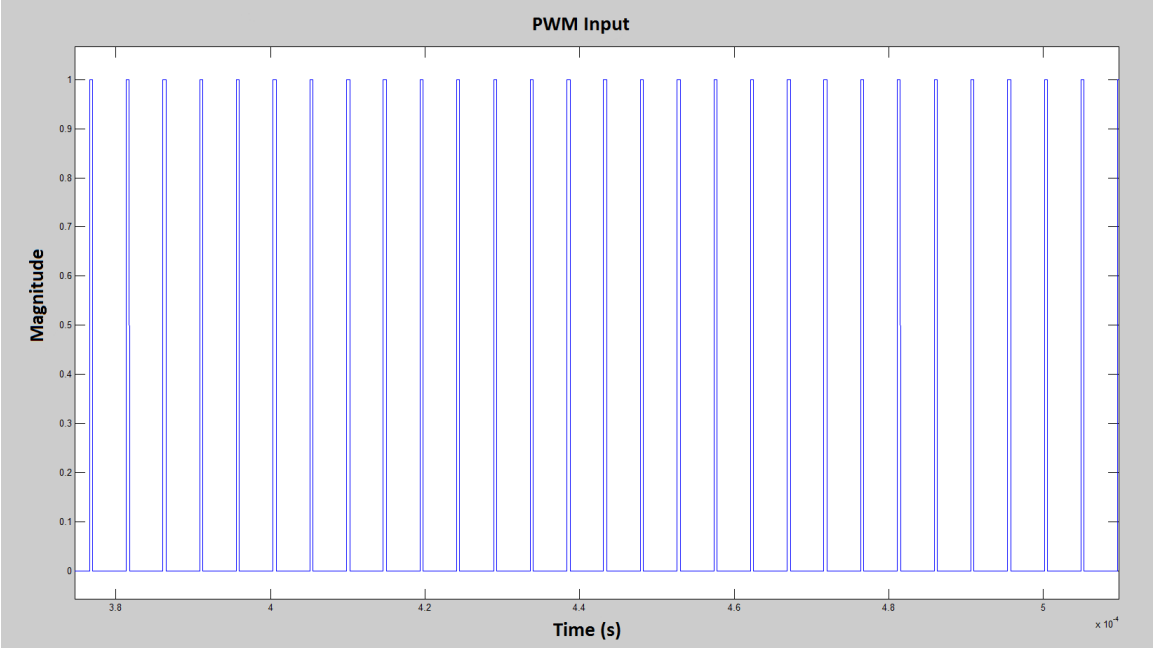

Figure 4.12: Close up of the PWM signal showing a duty cycle of aproximetly 8.3%

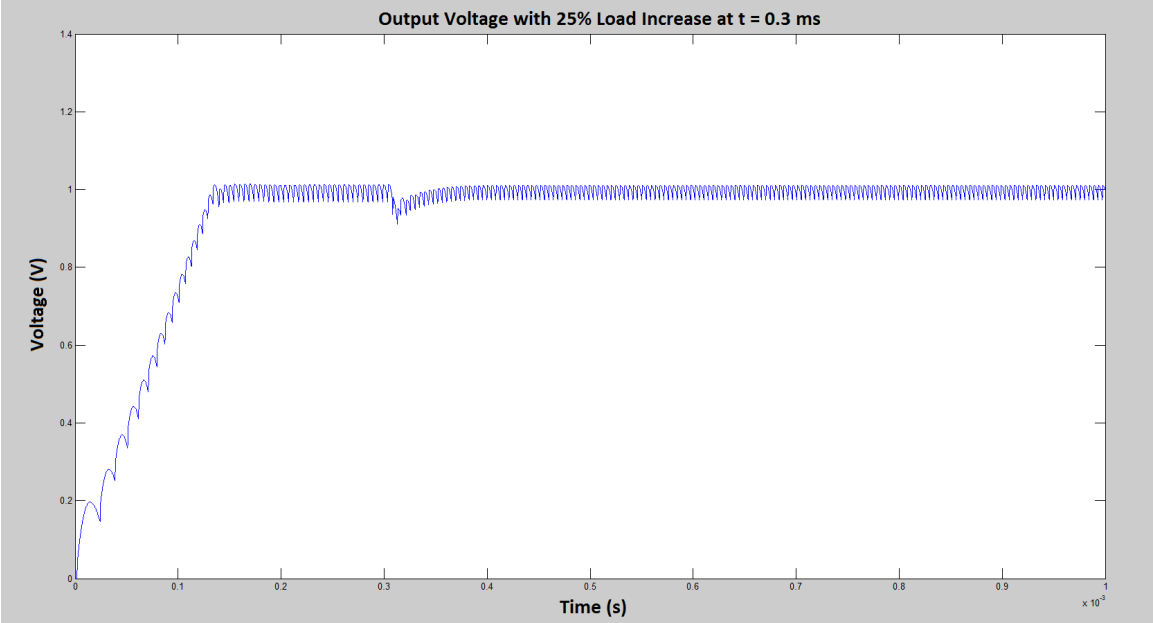

Figure 4.13: Output voltage response to a 25% increase in load

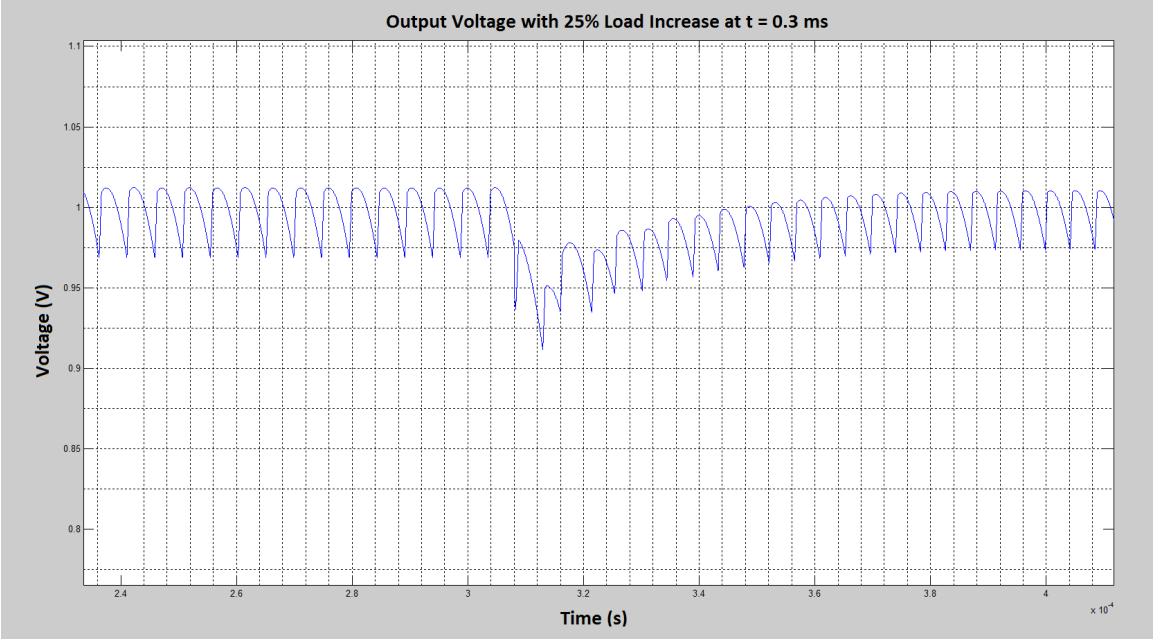

Figure 4.14: Close up of the output voltage drop of 80 mV

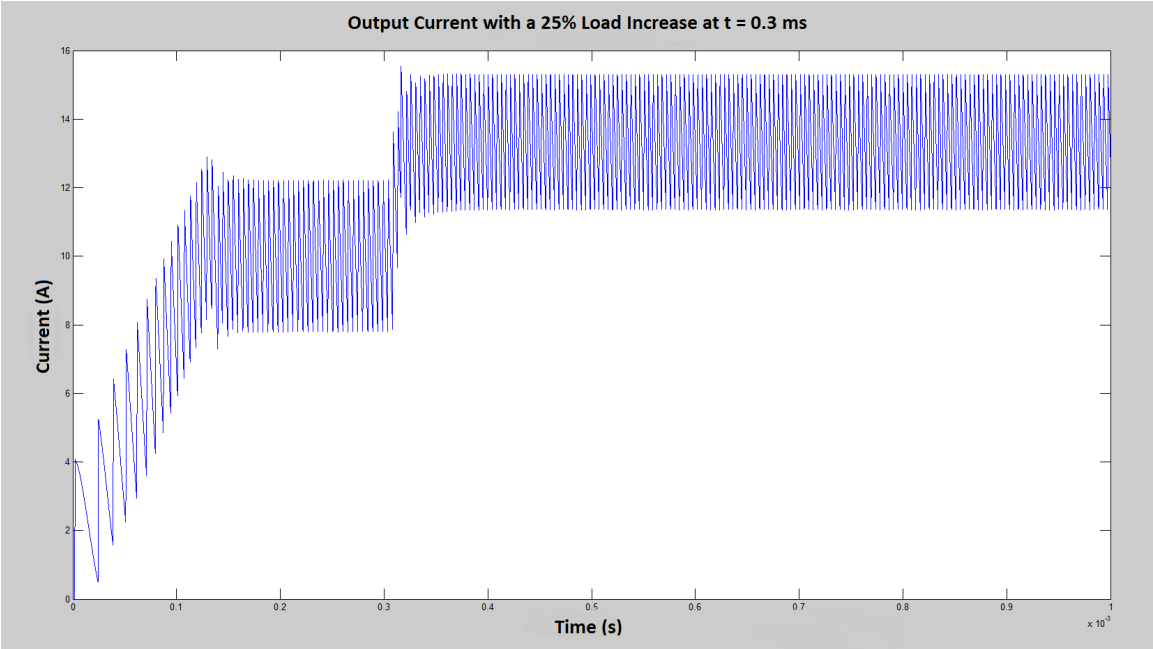

Figure 4.15: Output current of a single phase converter responding to a 25% increase in load

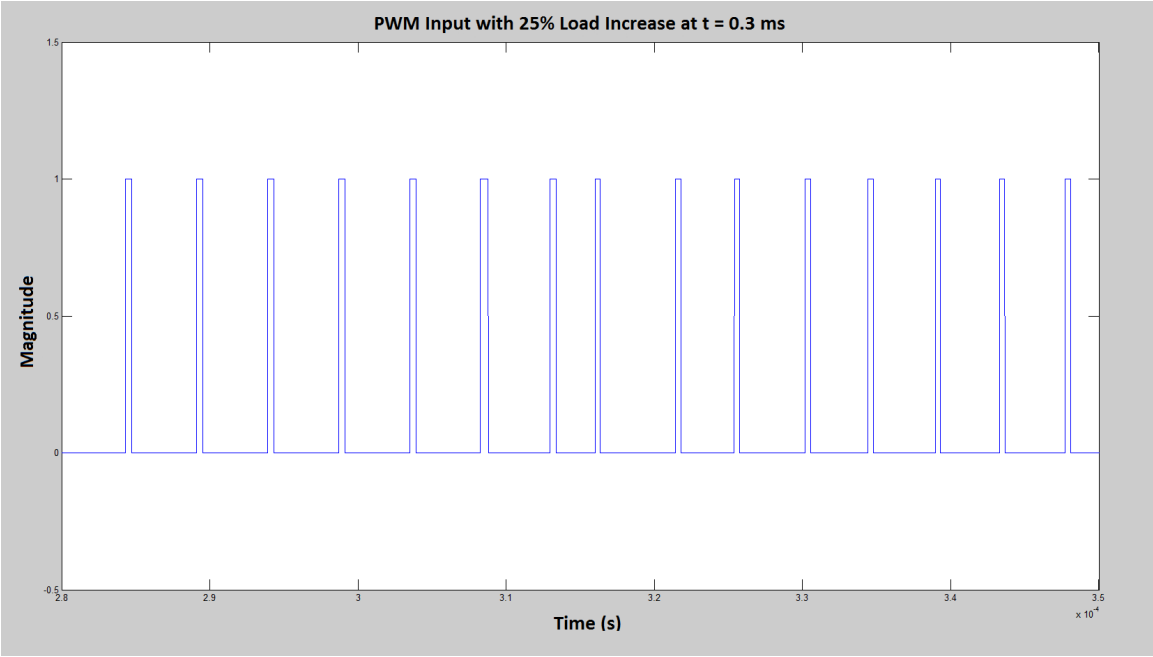

Figure 4.16: PWM signal close up of the 25% increase shows very little deviation from the steady state operation of 8.3% duty cycle

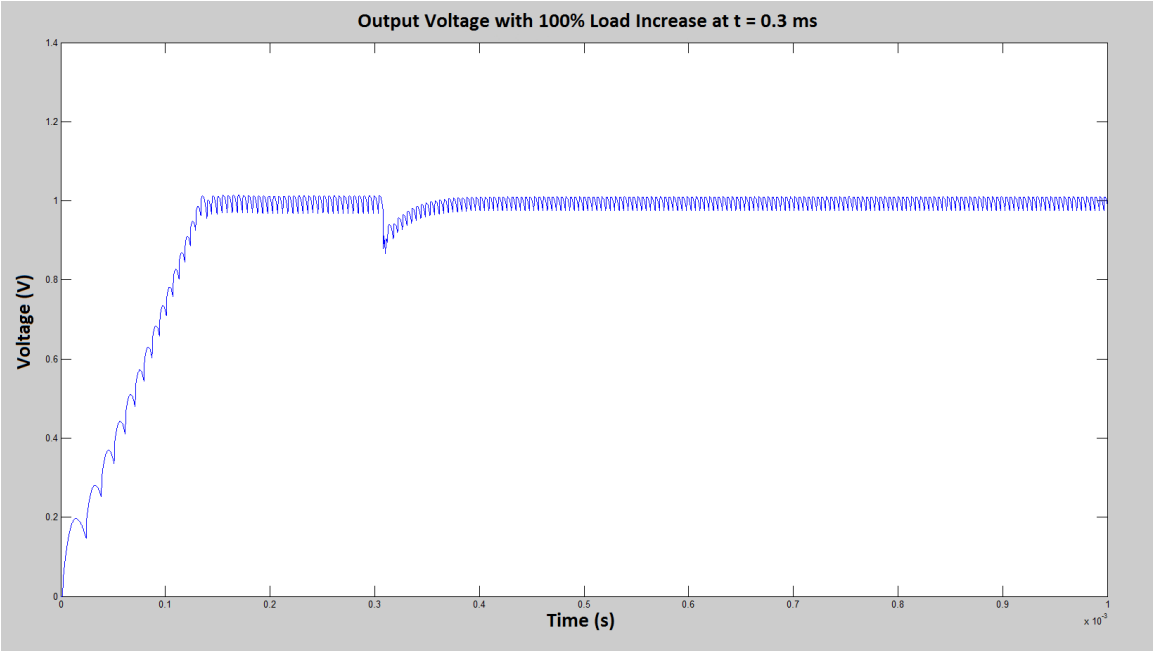

Figure 4.17: Output voltage response to a 100% increase in load

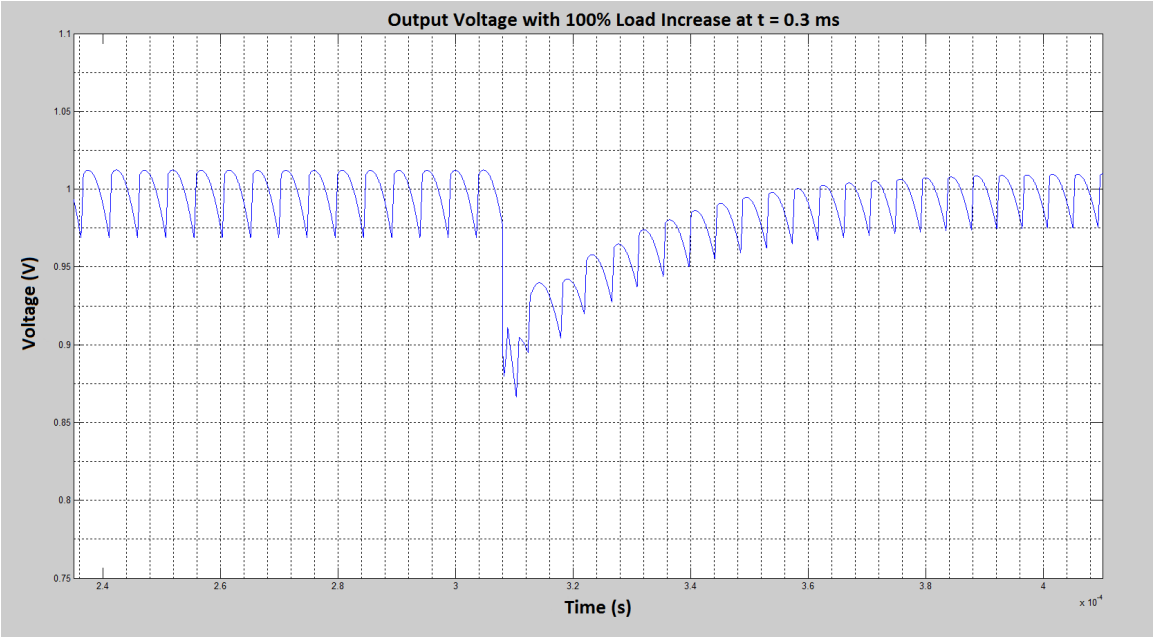

Figure 4.18: Close up of the output voltage drop of 150 mV

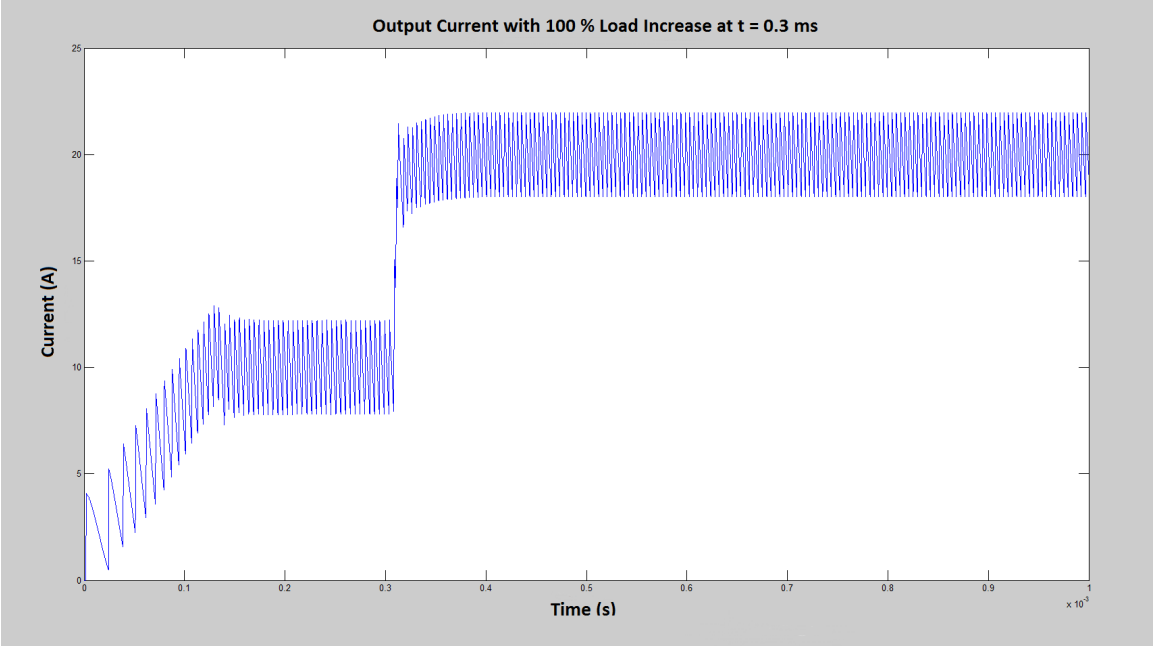

Figure 4.19: Output Current of a single phase converter responding to a 100% increase in load

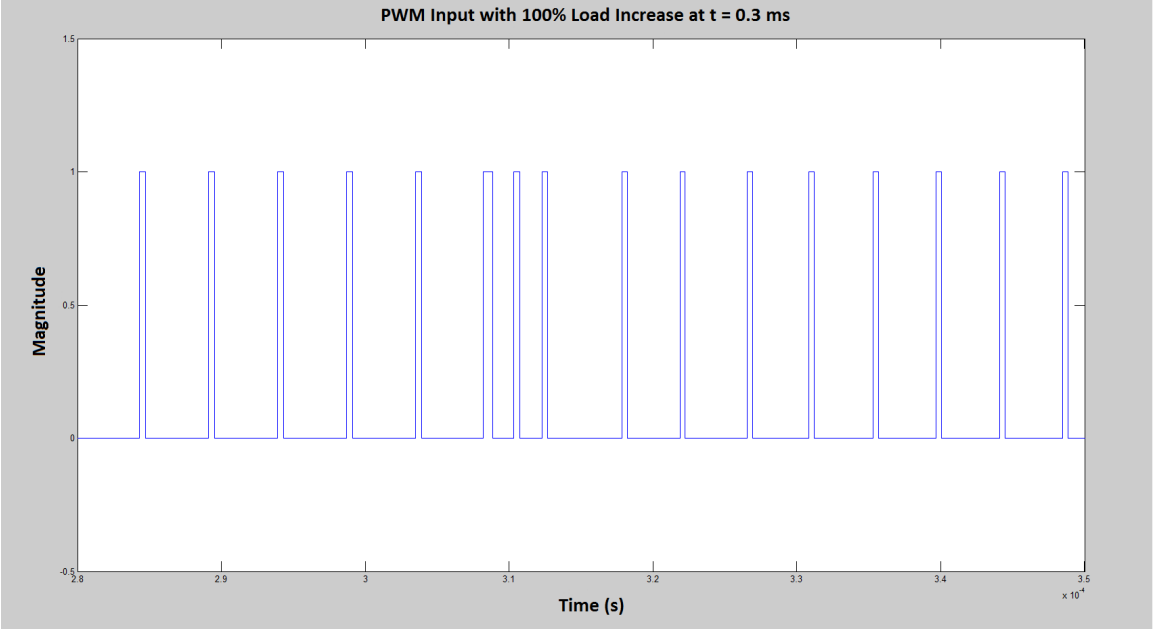

Figure 4.20: PWM signal close up of the 100% increase shows a significant deviation from the steady state operation of 8.3% duty cycle

4.4 Multiple Phase Model

With the single phase converter simulation model working correctly, the expansion to higher phase counts is now possible. This is done by simply running five converters in parallel which allows for equal current sharing among each phase. One of the added benefits of running multiple buck phases with interleaving currents is a reduction in the output voltage ripple. To do this, the PWM driving signals must be evenly spaced. The resulting currents for each phase are evenly spread as seen in Fig. 4.21.

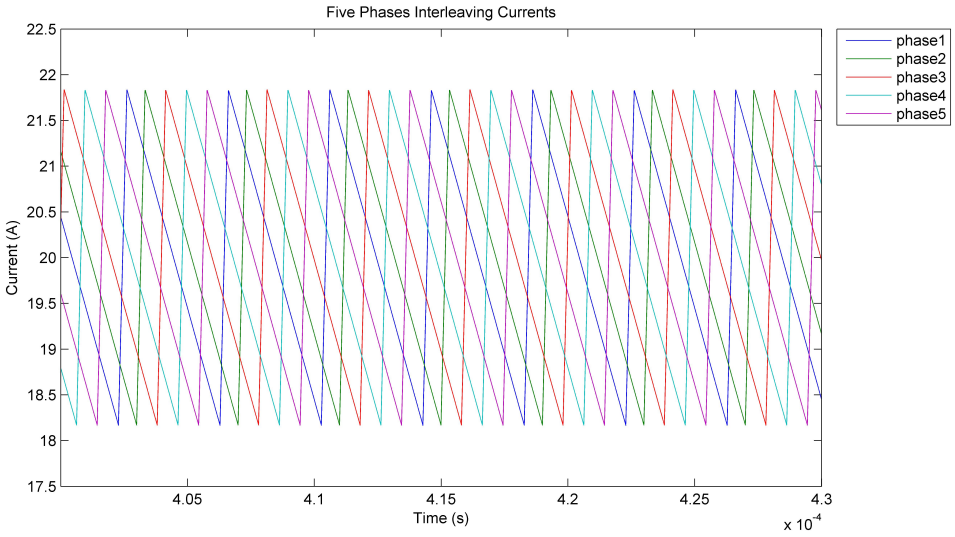

Figure 4.21: Output current of each of the five buck phases evenly spaced to reduce voltage ripple

To see the relationship between the phase count and the output voltage ripple, the same simulation is run with one, two and five active phases. The voltage ripple for a single phase approximately ranges ± 20 mV in Fig. 4.22. When the phase count is increased to two, the voltage ripple is reduced to ± 7.5 mV in Fig. 4.23. When the phase count is increased again to five, the voltage ripple is reduced to ± 2 mV in Fig. 4.24. This validates the assumption that increasing the phase count of a buck converter results in a significantly reduced output voltage ripple.

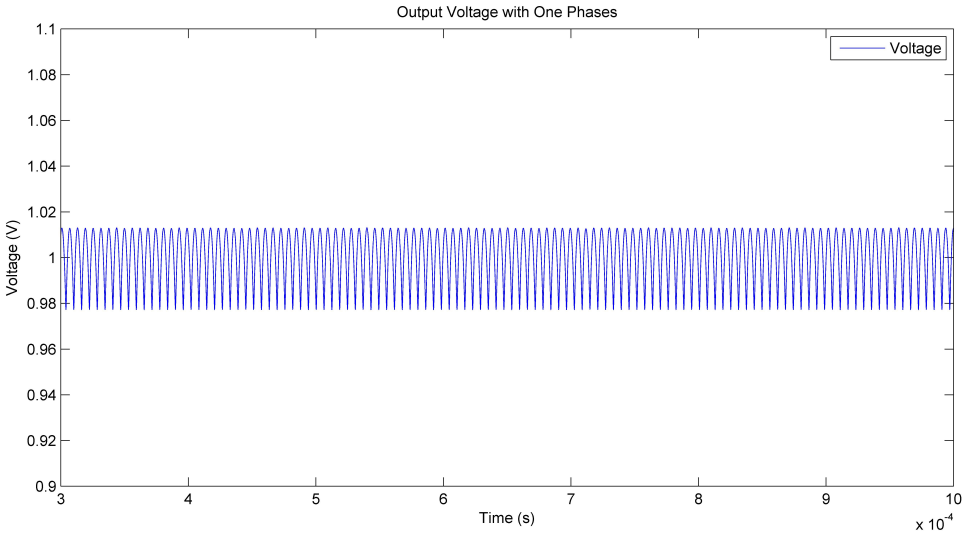

Figure 4.22: Output voltage of one active phase with a ripple of ± 20 mV

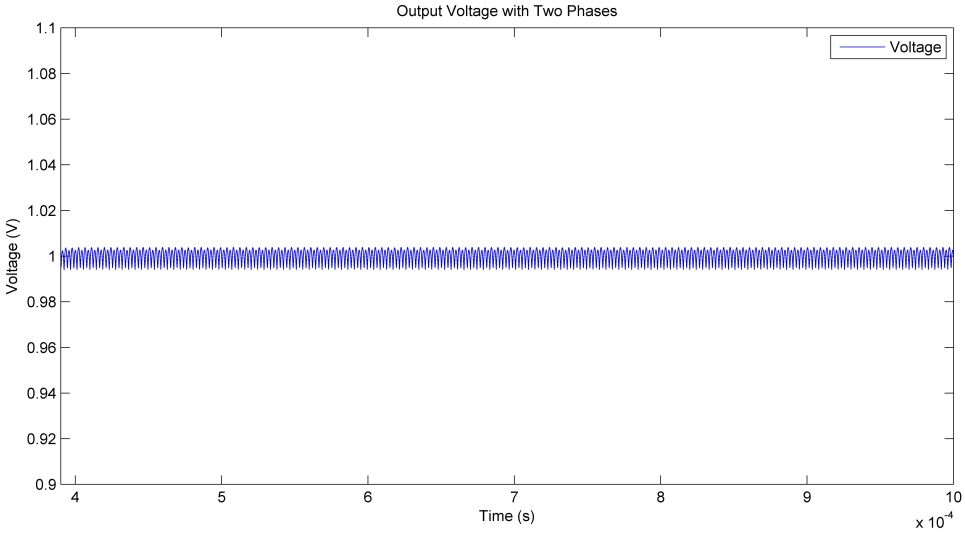

Figure 4.23: Output voltage of two active phases with a ripple of ± 7.5 mV

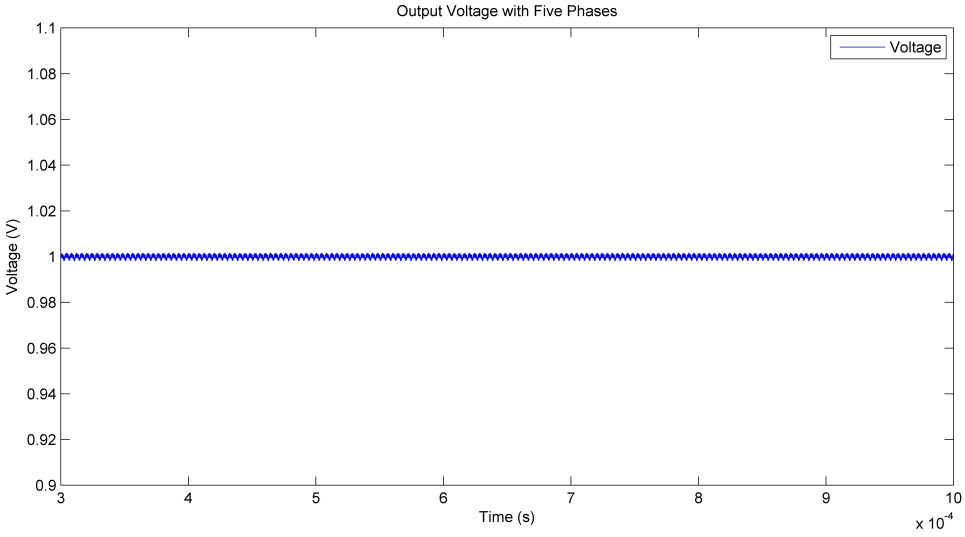

Figure 4.24: Output voltage of five active phases with a ripple of ± 2 mV

Chapter 5

Test Results

The testing setup is designed to measure efficiency for different board layouts and numerous parts. Since the projects primary focus is to maximize efficiency, a large range of frequencies at various loads needs to be tested. A broad range of switching frequencies starting at 150 kHz up to 500 kHz in steps of 10 kHz were tested at 3 A to 20 A for each phase.

This set up required multiple systems working together and was a collaborative effort on the project. The microcontroller was configured to communicate over a serial bus where an interrupt is triggered and a specific action is performed. This action is based on what key on the keyboard of the testing computer is pressed. Initially, this data was collected by manually changing the frequency and stepping the load for each value on the range. The system can be run manually over serial communications; however, this is a tedious process for efficiency data collection. To expedite the process, the data was to be collected autonomously. For the input power supply, the driver power supply and the digital power supply, all of the voltages and currents for the efficiency calculations are read off the power supplies over a general purpose interface bus (GPIB) system which is sent to the testing computer.

The microcontroller chosen for this project is the the Texas Instruments F28335 controlCard as seen in Fig. 5.1. This card was chosen because it has 16 analog-to-digital converter (ADC) channels to allow for plenty of sampling channels of the output voltage of five different buck converter phases as well as their individual currents, as well as the input voltage and current. Also, there are ten PWM channels available so that each converter phase has two separate signals for the switch and the synchronous rectifier. The reason two signals are used is so that there is an adjustable dead band to prevent shoot through current on the

switching device, as discussed previously. Another important aspect of this microcontroller, is that the PWM register is divided in half in two different portions. This allows for only half of the register to be called at once, leaving the other part unaltered. Meaning that the upper half of the register, or the most significant bits, are responsible for the largest change in duty cycle. The lower part of the register, the least significant bits, are responsible for the smallest possible steps in duty cycle. This portion of the register is used to increase the accuracy of the controller.

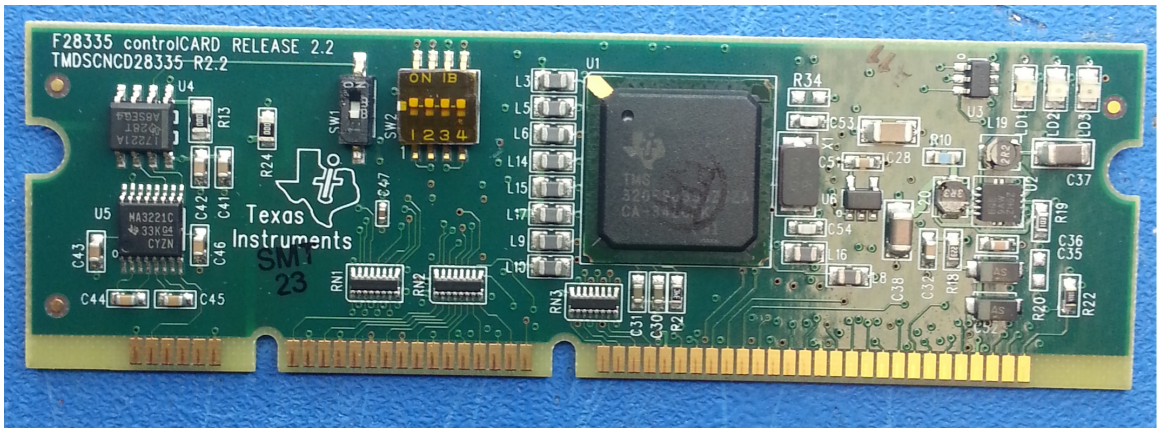

Figure 5.1: TI microcontroller used to control up to five buck phases

5.1 Discrete Time PID

The microcontroller has three specific tasks allocated for data collection. The first task is simply an LED counter which flashes the LED's on the card to provide the user feedback that the card is functioning properly. This is helpful for diagnosing damaged microcontroller cards. Another of these tasks is polling for keyboard presses. This is a serial communication system which allows the user to run the test from a keyboard. The user is capable of changing the frequency, load current, the duty cycle in large and small steps, activating each of the five PWM signals, controlling the dead band, disabling and enabling the controller, and operating a kill switch to turn off the controller and all the PWM signals. The control card measures the output voltage and current for each individual buck phase. It also provides a

unique PWM signal for the switch and the synchronous rectifier which allows for the dead band control for each of the five phases.

The third task contains the control loop and sampling of the output voltage. This task is called every 0.5 ms. With steady state analysis, the most important parameter for efficiency data collection, the controller is set to take four samples and average them together. This is important for a switching converter because the output voltage ripple can range $\pm$ 50 mV. If a single sample was taken at the most negative point then sampled the next time at the most positive point, then this would most likely lead to the controller potentially being unstable and oscillating as it approached the final desired value instead of settling at the desired value in a critically damped manner.

One of the requirements for this project is the ability to test various inductor values, at different loads, different frequencies and to vary the number of active phases. This means the controller needs to be able to handle all of these situations without any modification to the controller algorithm. The controller needs to be able to settle quickly for fast data collection but also needs to have the flexibility to fine tune the output voltage for any situation to maintain the integrity of the efficiency data.

To get both of these behaviors, the control algorithm is divided into two different parts. This is one advantage of the TI microcontroller card since the PWM registers for a large and small step can be accessed separately. For the large steps, the upper portion of the PWM register can be accessed. For fine tuning, the lower portion, called the high resolution portion of the register, can be accessed. The first part of this PID control loop, which will control the large step in duty cycle, which will help reduce the settling time of the controller. This portion of the controller will only be accessed when the sampled voltage is very far away from the reference voltage. Once the sampled voltage is close to the reference value, then the controller switches over to operating only on the high resolution register. This portion of the controller simply increments or decrements the register until the sampled value is equal to the reference value.

This control technique is preferred for this particular set up since it allows for more flexibility. The PID controller can be tuned to a specific inductor, frequency and phase count and will work perfectly for that board. This may not be true when you alter the phase count and increase the switching frequency. In some cases it can lead to instability. The major downside to this approach is that it will increase the settling time. Since the high resolution portion of the controller slowly increments the register, it can take multiple control loop cycles before it settles to the correct value. With efficiency being the primary focus of this project, this sacrifice in settling time to gain flexibility is justified.

5.2 Steady State Error

To make sure the controller is suitable for efficiency analysis, the steady state error of the system needs to be zero. This is especially important since the efficiency testing process requires the current to constantly be cycled at different frequencies. If the controller does not have a steady state error of zero in all of these situations, then the efficiency data might not be valid. The PID controller contains the integral term for this purpose. The integral term takes into account the error accumulation over time to force the steady state error to zero.

First a tolerance range for the output voltage needs to be determined. A common used error tolerance band is 5% of the final value. In this case, 1000 mV is the desired reference which makes the error band $\pm$ 25 mV when looking at the average value of the output voltage. It should be noted that the typical voltage ripple of the switching converter is approximately 950 mV to 1030 mV when a single phase is operating at 5 A. These minimum and maximum values tend to increase at higher loads.

The converter is allowed to settle out at approximately 5 A, 10 A, 15 A and 20 A. The oscilloscope screen captures of the 5 A results can be seen in Fig. 5.2 where the average value of the output voltage is 999 mV. As the load is increased to 10 A in Fig. 5.3 this value is almost the exact same. Increasing the load current futher to 15 A in Fig. 5.4, It can be seen

that the average value of the output voltage has decreased to 988 mV. Again it decreases even further for the 20 A where the voltage is 983.2 mV in Fig. 5.5. This is due to the change in the shape of the output voltage ripple with larger negative peaks at the higher load currents. This is still not a major issue since the average output voltage ripple is still 16.8 mV off the desired value of 1000 mV. The 16.8 mV is less than the error band of ± 25 mV, so it satisfies the 5% tolerance defined earlier. This error is sometimes attributed to insignificant integral gain, however the integral gain was increased to it maximum stable level where the results were the same.

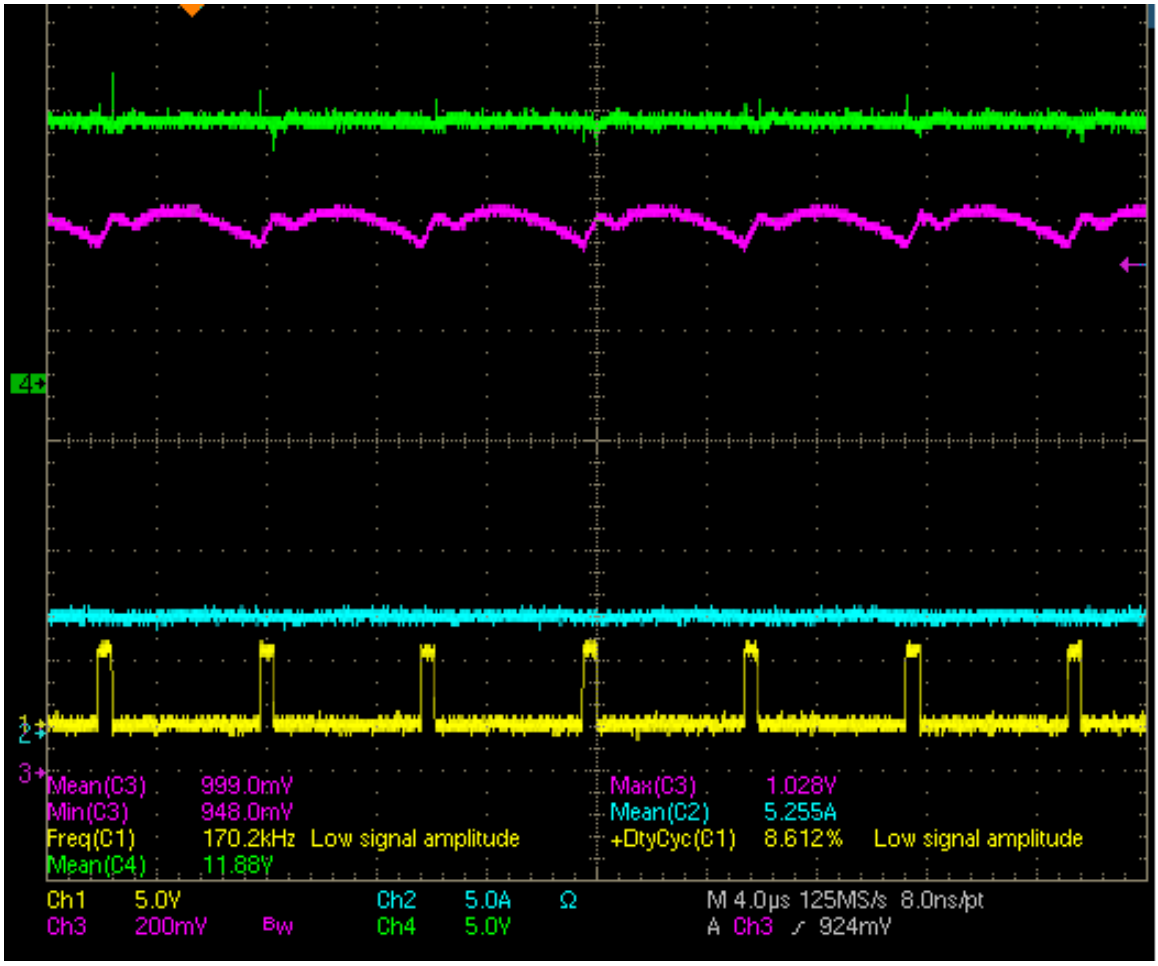

Figure 5.2: Steady state operation at 5 A load current

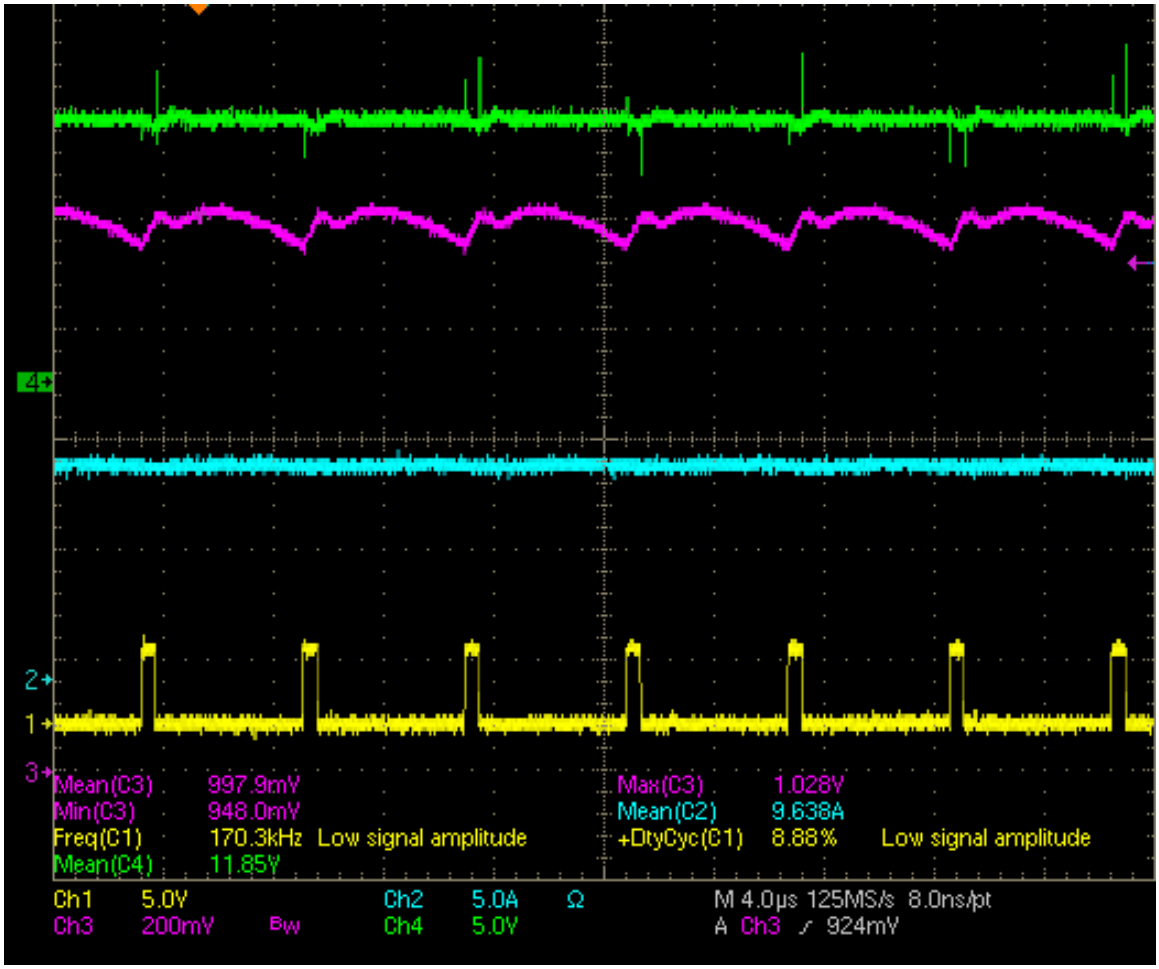

Figure 5.3: Steady state operation at 10 A load current

Figure 5.4: Steady state operation at 15 A load current

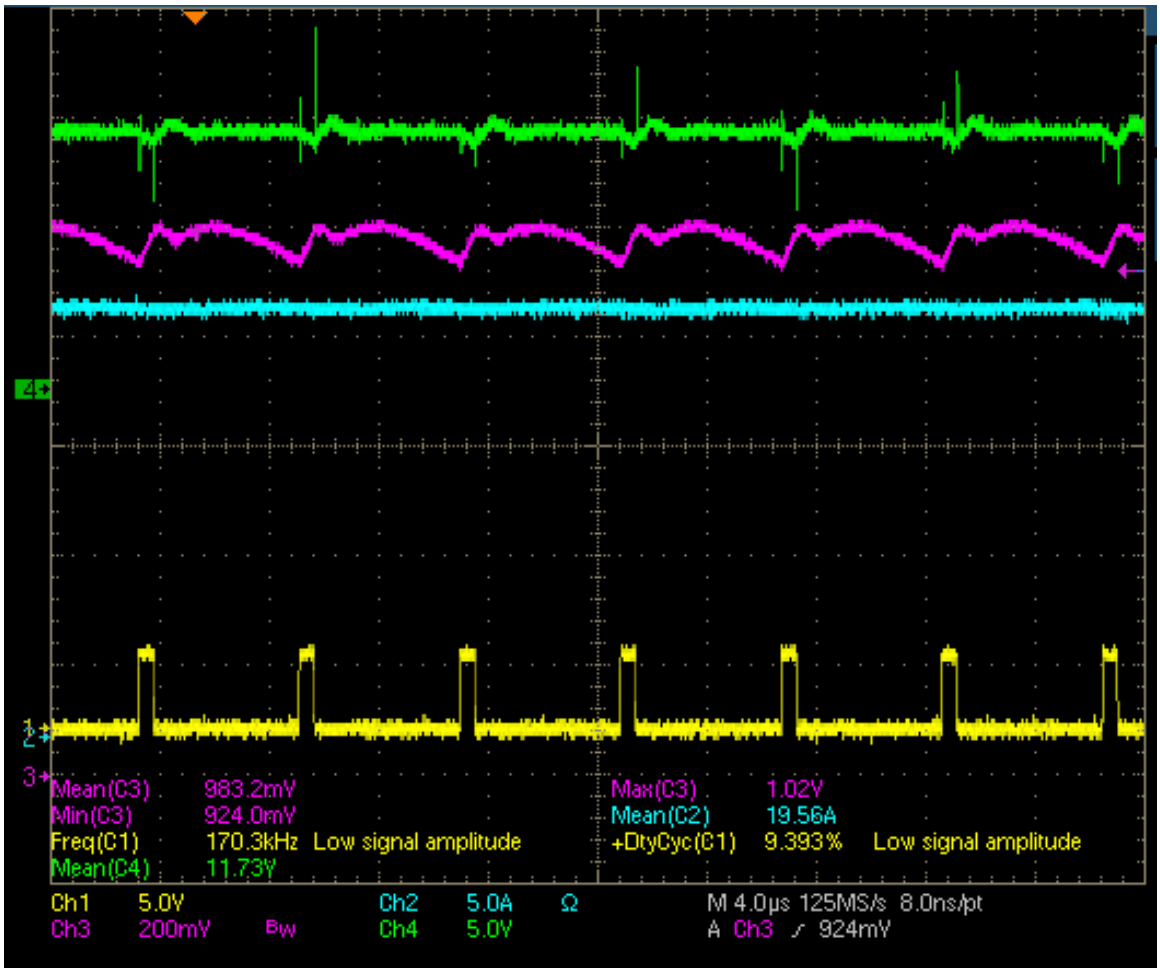

Figure 5.5: Steady state operation at 20 A load current

5.3 Transient Response and Output Capacitance

To determine the dynamic response of the controller, various changes on output current are performed. This is done by increasing or decreasing the output current on a "dummy" resistor load board. The output voltage response is then recorded to get the transient response of the voltage controller.

The majority of this project was centered around maximizing efficiency and maintaining the smallest PCB footprint possible. For this reason, very few output ceramic capacitor pads were populated on some of the top performing boards. This allowed for the impact of different capacitors to be experimentally determined. The effect of ceramic, electrolytic and super capacitors was investigated. All of the boards contain some ceramic output capacitors which equate to approximately 1.44 μF. All of these capacitors have a negligible effect on steady state operation of the converter.

For this comparison, a output load increase from 5 A to 15 A was performed. Increasing the capacitance will increase the initial response time of the converter but at the same time it will decrease the initial voltage drop. It is determined that this assumption is correct up until a certain point where the added capacitance hinders the responsiveness of the controller. A summary of these results can be found in Tab. 5.1. During steady state operation, the typical voltage ripple for this particular board ranges from approximately 0.95 V to 1.03 V peak to peak. The board has a SER1590-202 inductor with one switch and four synchronous rectifiers.

Here it can be seen that when you take the step up from the two 560 μF capacitors to the 2200 μF capacitor, there is a significant increase in initial response time which prevents the controller from correcting the voltage quick enough. This results in a larger initial drop in the output voltage. Another thing to take from this data is that the type of capacitor used is very important. The 1 F super capacitor performance was more comparable to the 560 μF electrolytic. These capacitors are comparable in size, but the super capacitor's

architecture inherently has a higher internal inductance. This makes the super capacitors very application specific.

Initial Response (μs)	Vmin (mV)	Capacitance (μF)	Type
60	428	None Added	-
100	624	1,000,000	Super
100	652	560	Electrolytic
120	708	560 x 2	Electrolytic
200	644	2200	Electrolytic

Table 5.1: Summary of impact of various output capacitance types and values when the output currrent is increased from 5 A to 15 A

To attempt to further improve performance, adding ceramic capacitors was investigated on a different board. The typical ceramic capacitors equate to around 1.44 μF. This is doubled to 2.88 μF by stacking the ceramic capacitors on top of each other. This is not an ideal way to add more capacitance since the solder is not as conductive as copper. This was the only way to add more without changing the board layout and ordering another board run. This very small increase in capacitance had a more significant impact on the output voltage drop than the addition of the electrolytic capacitors. This is most likely due to the lower ESR of the ceramic capacitors.

This time various load increases and load decreases were performed for a more in depth analysis. This board contains a SER1590-301 inductor and one switch and four synchronous rectifiers. For each instance, the initial response time, the maximum voltage and the minimum voltage are recorded. The base test for no extra capacitance value added is found in Tab. 5.2. The initial response time is not the initial response time, but the amount of time it takes to takes for the PID portion of the loop to complete. After this, the incremental controller will fine tune the output which takes significantly longer but allows for the controller to be stable in all situations.

Next the ceramic output capacitors are stacked, effectively doubling the output capacitance of the board up to 2.88 μF, as shown in Tab. 5.3. The additional capacitors made

Load Step (A)					
From	To	Initial Response (μs)	Vmax (mV)	Vmin (V)	Difference (mV)
1	5	80	1.084	756	-244
5	10	80	1.156	746	-252
5	15	100	1.132	660	-340
15	5	120	1.396	876	+396
10	5	120	1.252	852	+252

Table 5.2: Various load steps with corresponding initial response times and voltage maximum and minimum values with no capacitance added

a significant impact on the maximum and minimum voltages with almost no impact on the initial response time. When the load is increased from 5 A to 10 A, there is an approximately 40 mV improvement in the voltage maximum and minimum. Similar improvements across all the load changes can be seen. This is a direct result from adding the ceramic capacitors.

Load Step (A)					
From	To	Initial Response (μs)	Vmax (mV)	Vmin (V)	Difference (mV)
1	5	80	1.06	804	-196
5	10	80	1.164	756	-244
5	15	100	1.148	700	-300
15	5	120	1.348	828	+348
10	5	120	1.19	860	+190

Table 5.3: Various load steps with corresponding initial response times and voltage maximum and minimum values with stacked ceramic capacitance added

Since there is not other practical way to increase the number of ceramic capacitors, a 560 μF electrolytic capacitor added was to the 2.88 μF ceramic capacitors, as shown in Tab. 5.4. The addition of the electrolytic capacitor increases the initial response time but minimizes the voltage drops similar to the previous test. The additional capacitance does improve the performace. However, adding the small number of ceramic capacitors had around the same effect as the large increase of the capacitance from the electrolytic capacitor.

It should be noted that when the current was steped up from 10 A to 20 A, as was done in the simulations for a 100% load increase, the voltage drop is aproximetly 200 mV. This is comparable to the 150 mV drop which was simulated in the Simulations Chapter.

Load Step (A)					
From	To	Initial Response (μs)	Vmax (mV)	Vmin (V)	Difference (mV)
1	5	80	1.028	868	-132
5	10	100	1.028	876	-124
5	15	100	1.012	796	-204
10	20	100	1.024	800	-200
15	5	120	1.216	940	+216
10	5	120	1.16	948	+160
15	1	160	1.348	948	+348

Table 5.4: Various load steps with corresponding initial response times and voltage maximum and minimum values with stacked ceramic capacitors added

When looking at the increase in load from 5 A to 15 A, there is a significant improvement in increasing the output capacitance. With no added capacitance, the voltage difference is 340 mV. When the ceramic capacitors are doubled, the voltage drop is decreased to 300 mV. Finally the addition of the electrolytic capacitor improves this result even more and decreases the drop to 200 mV. This does come at the cost of increasing the initial response time approximately 20 μs in most cases. This also increases the amount of board space required for the circuit. However, the improvements in the voltage performance outweighs the small increase in initial response time.

5.4 Transient Response Single Phase

The transient response of a single converter phase is the amount of time it takes to settle to within 5% of its final value. In this case, the primary PID loop and the amount of capacitance on the output will determine how much of an initial drop in voltage there is for a sudden increase in load current. After the sampled value falls approximately to within 50 mV of the reference value, then the incremental control loop takes over. As stated before, this does increase the settling time of the controller significantly; however, this particular application favors flexibility over settling time. All of the data in this section contains a single phase board with 2.88 μF ceramic capacitors and a single 560 μF electrolytic capacitor. For

this particular project, the customer only requires a step up in load current and to be able to maintain that output current. There is no need for decreasing the current for this specific application, so only load increases were investigated. Another important aspect to consider is that a spike in voltage can be potentially damaging to the load, so overshoot must be minimized. An under voltage is not damaging but is still unwanted.

The output voltage response is recorded for a step in output current from 1 A to 5 A and 5 A to 10 A in Fig 5.6. Here the voltage drops to 892 mV and 876 mV respectively. In both cases, there is approximately a little over a 10% drop in the output voltage from the reference value. The amount of time that it takes both of these situations to completely settle is approximately 2.25 ms. This means it takes 5 full control loop cycles before the controller is completely settled. It should be noted that the controller is within 5% of the reference value after two cycles. In both of these cases, there is a similar drop in voltage and settling time because they are similar steps in current.

The output current increase for the 1 A to 5 A jump can be seen in Fig. 5.7 and the 5 A to 10 A jump can be seen in Fig. 5.8. As seen in the current increases, the ripple of the output current ranges from ± 0.5 A, but gets slightly larger as the load increases.

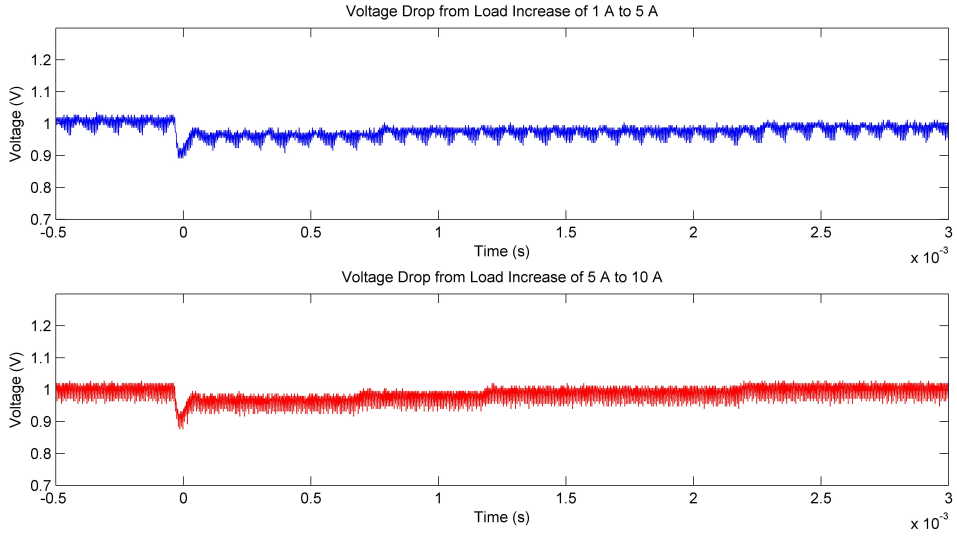

Figure 5.6: Output voltage response to a step up in load of 4 A (Blue) and 5 A (Red)

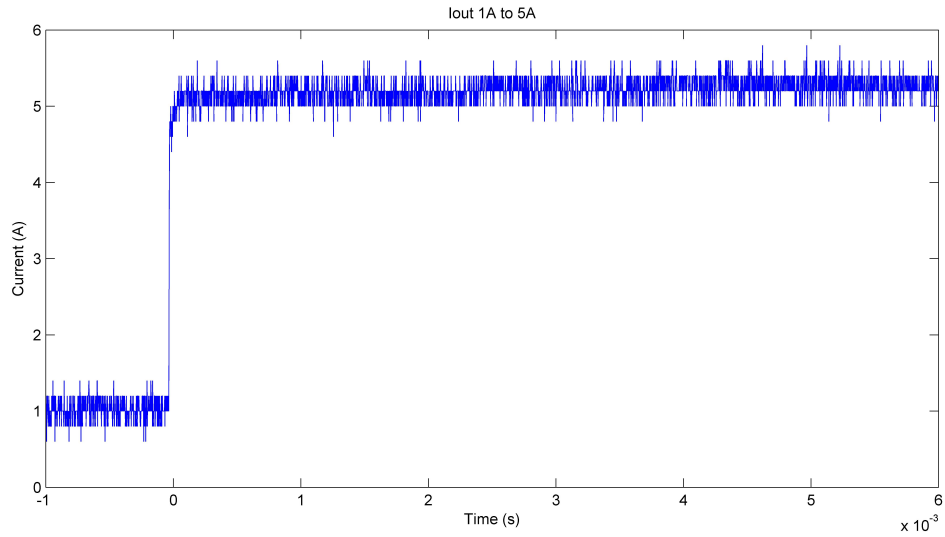

Figure 5.7: Output current response to a step of 4 A

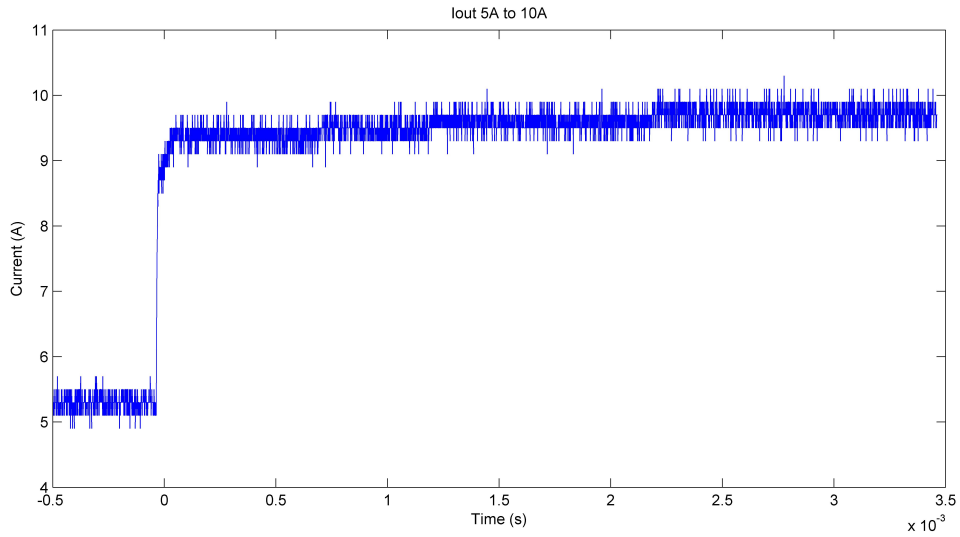

Figure 5.8: Output current response to a step of 5 A

Next the current is stepped up 10 A at a time from two different starting values. The voltage response of a load increase from 5 A to 15 A and 10 A to 20 A can be seen in Fig. 5.9. Once again, the results are similar even though they do not start from the same initial current. This means that the voltage drop is only dependent on how much of a jump in output current. When the load is stepped from 5 to 15 A, the voltage drops down to 780

mV which is 22% of the reference value. When the load is stepped up from 10 to 20 A, the voltage drops down to 828 mV which is 17.2% of the reference value.

The output current increase for the 5 A to 15 A jump can be seen in Fig. 5.10 and the 10 A to 20 A jump can be seen in Fig. 5.11. As seen in the current increases, the ripple of the output current ranges closer to ± 0.5 A, but at 20 A the peak to peak value can be up to ± 1 A.

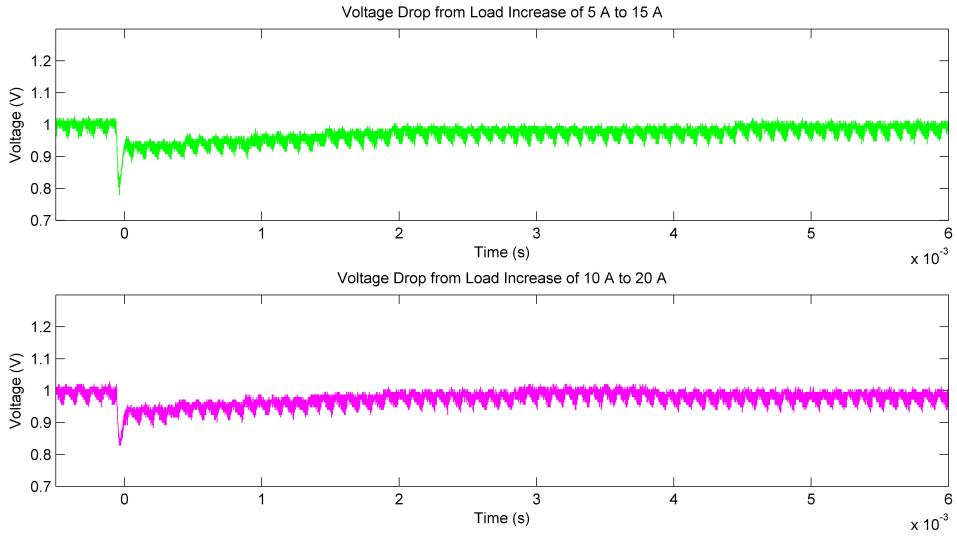

Figure 5.9: Output voltage response to a step up in load of 5 to 15 A (Green) and 10 to 20 A (Magenta)

Included is a summary of all of the single phase transient response results for each load increase, the minimum voltage value and the percentage that value is off the reference value in Tab. 5.5. This leaves room for potential future work for designing a fine tuned PID controller for a single phase board with more aggressive gains.

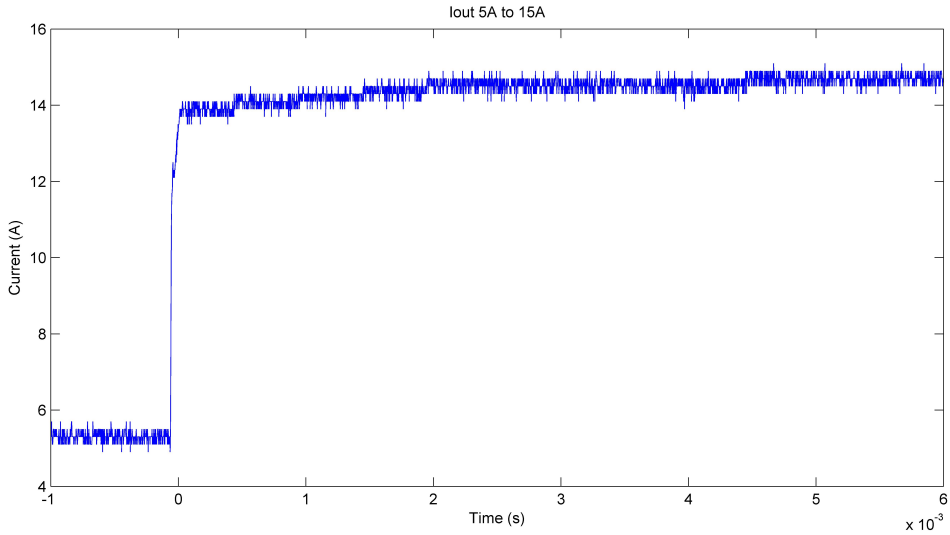

Figure 5.10: Output current response to a step of 10 A

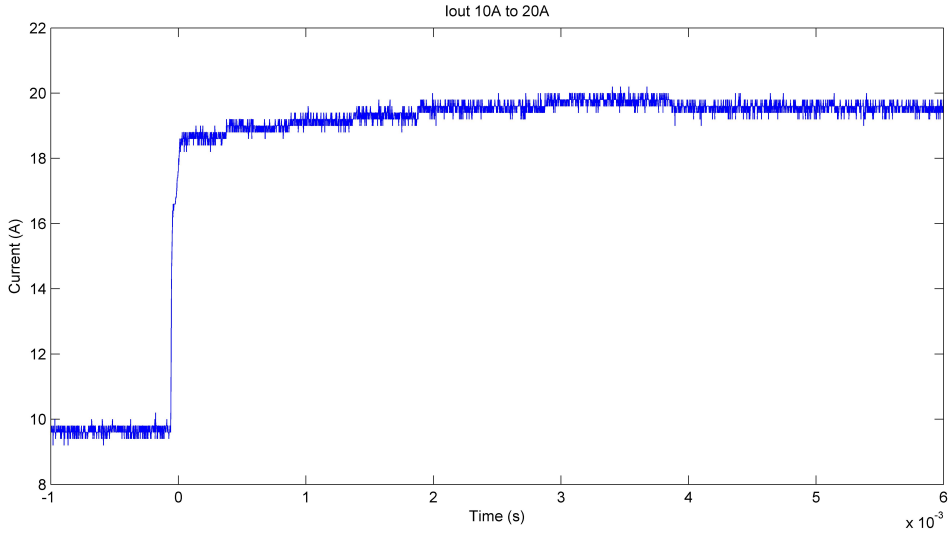

Figure 5.11: Output current response to a step of 10 A

Load Step (A)					
From	To	Step (A)	Vmin (mV)	Vdrop (mV)	Percentage (%)
1	5	4	892	108	10.8
5	10	5	873	124	12.4
5	15	10	780	220	22
10	20	10	828	172	17.2

Table 5.5: Various load steps with the minimum voltage drops and the percentage of change in relation to the reference value

5.5 Transient Response Five Phases

One of the things demanded by the controller is flexibility. This means the controller is going to need to be able to run a single phase all the way up to five phases simultaneously. This is somewhat difficult to achieve since adding more phases requires a reduction in gain of the controller to get similar results. This is due to the controller affecting the duty cycle of each phase simultaneously. For this reason, the controller gain needs to fall within a range where each of these configurations remain stable.

With a higher effective gain for the multiple phase case, the voltage response should be expected to have more oscillatory behavior. Another issue to be aware of is that the multiple phase case has interleaved PWM signals. This means all of the current of each phase would ideally be evenly spaced apart. This is not the case in this application since the PWM is a digitally controlled signal which may not be capable of representing the exact spacing required. This means the output current ripple will sometimes vary in size which could affect the controller stability. The added benefit of interleaving the PWM signals means that the output voltage ripple is significantly reduced.

In a similar manner as the single phase tests, the load current is increased to see the effect on the output voltage. The results for a 5 A to 10 A and a 10 A to 20 A step in load can be seen in Fig. 5.12. For the 5 A increase in load, the output voltage only drops 100 mV. This makes the minimum value of the output voltage 900 mV which correlates to a 10% drop in voltage. This is a slightly better result than the single phase tests where the voltage dropped 124 mV (12.4%). The same test is performed for a 10 A to 20 A step. Here the output voltage drops down to 804 mV which is a 196 mV drop from the reference. This is a 19.6% change in the voltage which is worse than the single phase results of a 17.2% change for the same current step.

One thing that both of the voltage plots show is small perturbations. This ringing behavior was to be expected because of the digital PWM interleaving. The ringing overshoot is minimized to reduce the risk of damaging the load. These oscillations are also present

in the output current as seen in Fig. 5.13 and Fig. 5.14. This is due to the uneven spacing of the PWM interleaved signals which causes an imbalance on each of the five buck phases. Another issue is the tolerance on each of the five inductors as well as the output capacitance. This makes each phase of the converter slightly different.

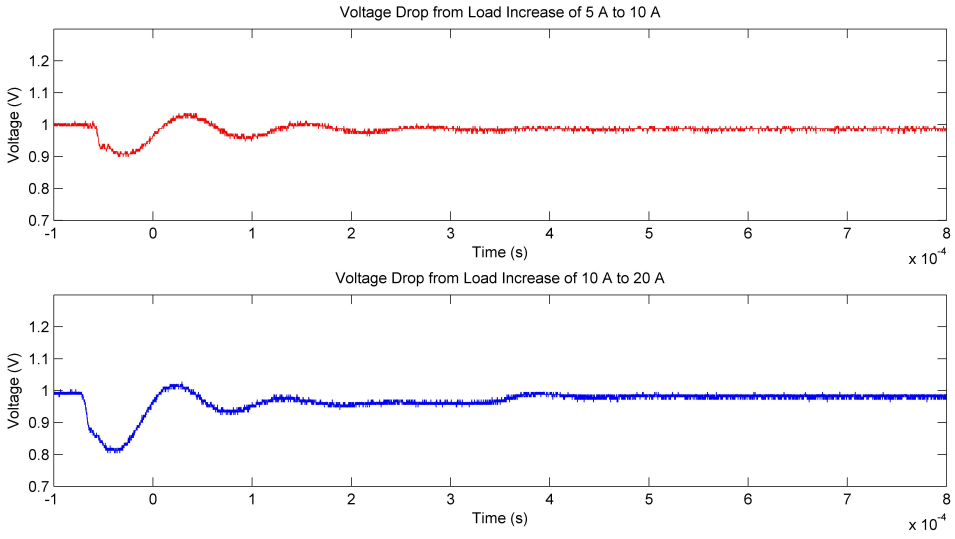

Figure 5.12: Output voltage response to a step up in load of 5 A (Blue) and 10 A (Red)

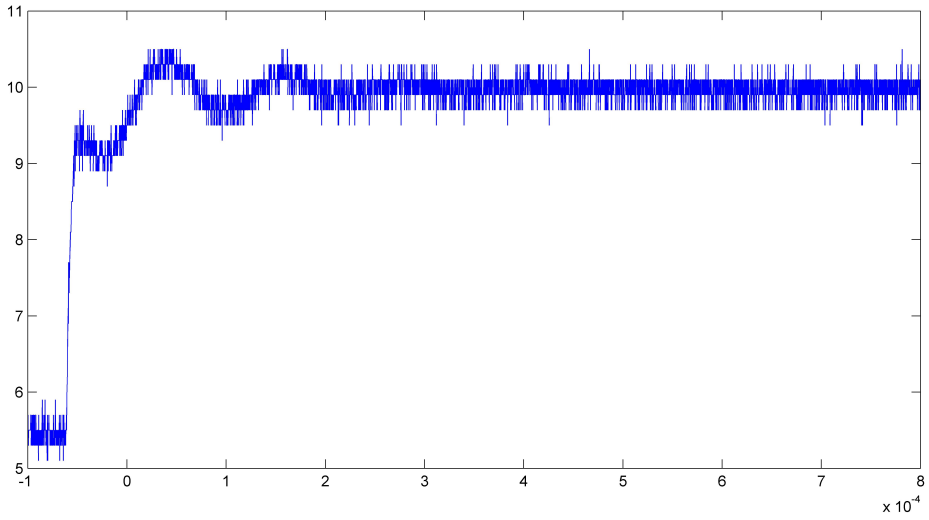

Figure 5.13: Output current response to a step of 5 A

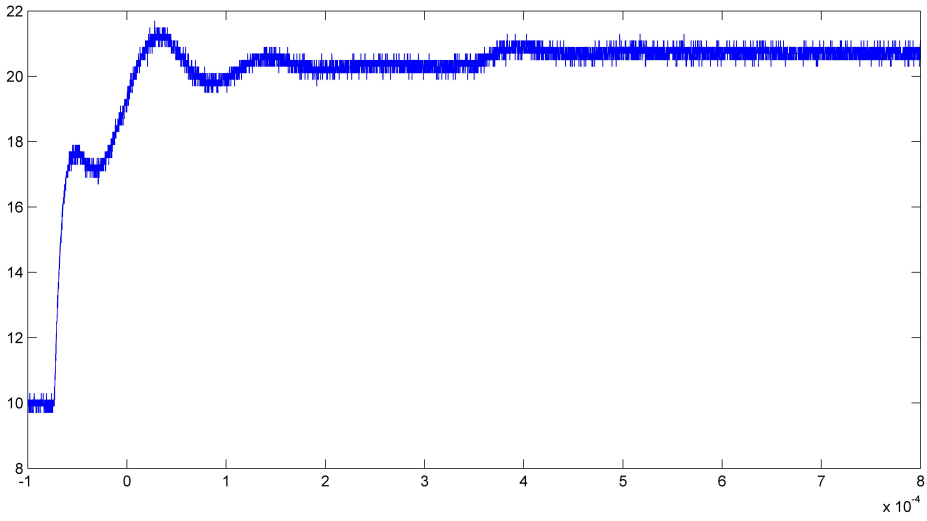

Figure 5.14: Output current response to a step of 10 A

Since the output current is shared between each of the five phases, the converter can operate at higher currents then a single phase can. It also follows the same trend that only the magnitude of the step in load current matters, not the initial and final value. To look at a worst case step in load, a 5 A to 25 A and a 1 A to 25 A step are performed Fig. 5.15. For the 20 A step, the voltage drops down 340 mV down to 660 mV. This is a 34% drop from the reference value for a 20 A increase in load. When the converter is stepped up 24 A, the voltage drops down to 364 mV which results in the lowest voltage being 636 mV. This is a 36.4% drop down form the reference voltage for a 24 A increase in load.

In these two cases, the same minor voltage ringing are present that were not found in the single phase results. These are also present in the load steps of 1 A to 25 A in Fig. 5.16 and the step of 5 A to 25 A in Fig. 5.17. Again the current ripple grows as the output current increases. At 25 A, the peak to peak value ranges $\pm$ 1 A. This is a similar result to the single phase tests. An advantage that the five phase board has over the single phase is the reduced output voltage ripple. The single phase output voltage ripple ranges $\pm$ 8 mV where as the five phase ripple ranges $\pm$ 3 mV. This is one of the desired benefits of interleaving PWM signals to drive multiple buck phases.

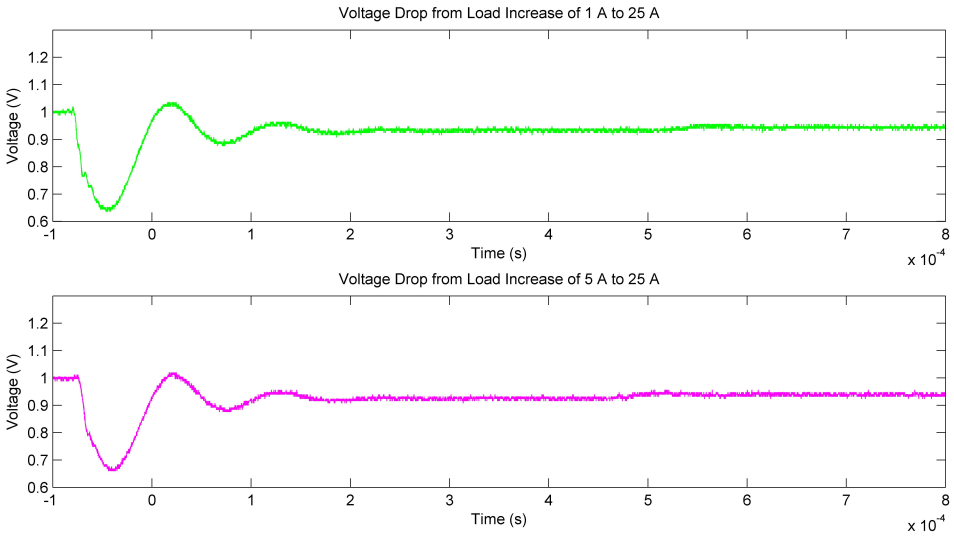

Figure 5.15: Output voltage response to a step up in load of 5 A (Blue) and 10 A (Red)

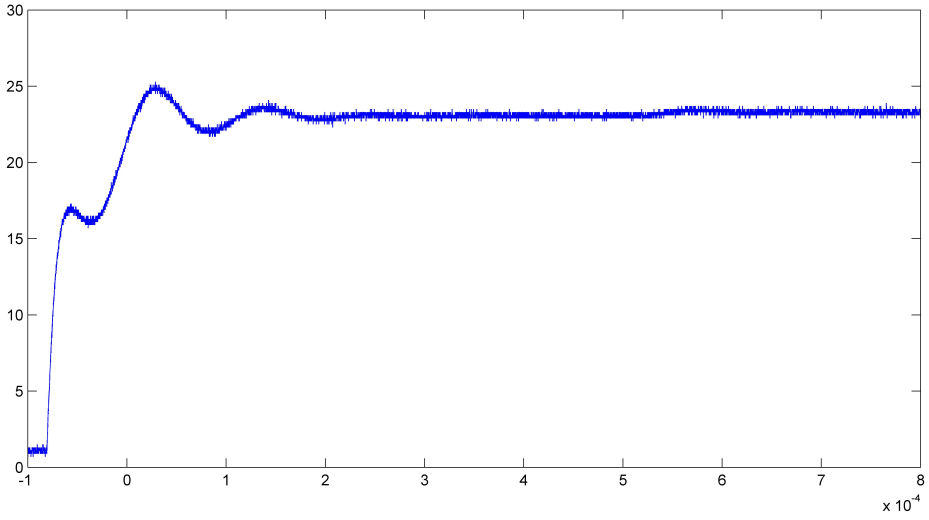

Figure 5.16: Output current response to a step of 24 A

A summary of the five phase transient results can be found in Tab. 5.6. This table includes the magnitude of the step in output current, the minimum voltage, the drop in voltage from the reference value and the percentage drop. The single and five phase results have approximately the same drop in voltage for the 5 A and 10 A increases.

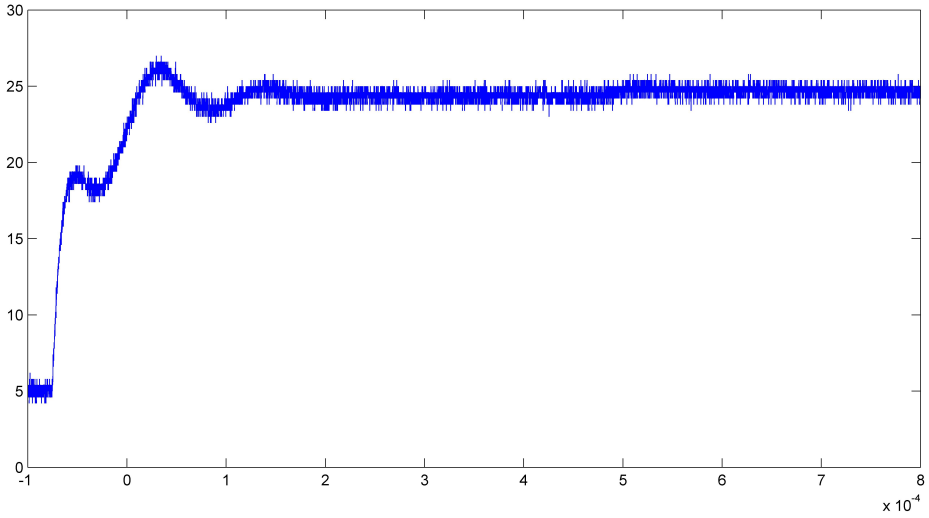

Figure 5.17: Output current response to a step of 20 A

Load Step (A)					
From	To	Step (A)	Vmin (mV)	Vdrop (mV)	Percentage (%)
5	10	5	900	100	10
10	20	10	804	193	19.6
1	25	24	636	364	36.4
5	25	20	663	340	34

Table 5.6: Various load steps with the minimum voltage drops and the percentage of change in relation to the reference value

Chapter 6

Conclusion

This research has explored some of the work on previous buck modeling and controllers. The background and basics behind a digital system lays the foundation for the digital implementation of a PID controller. Before the controller is considered, a model for the switching converter is derived from the governing differential equations. Using this model, a block diagram is derived which allows for a continuous and discrete time controller to be derived. The model, continuous and discrete time controllers are analyzed in the frequency domain using Bode plots to calculate the gain crossovers and phase crossovers. These values allow for the calculation of the gain and phase margins for stability analysis.

The converter is then simulated with the controller in various configurations. The controller is confirmed to be able to handle one to five converter phases with interleaved PWM signals. This discrete time controller is then programmed into a microcontroller to be tested on hardware. The results are presented showing the impact of output capacitance on the voltage drop and initial voltage response. The transient analysis is presented for both the single and five phase converters. The single phase results were able to maintain the voltage within 12.4% of the original value for a step in load of 5 A and within to 17.2% of the reference with a 10 A step in load.

The five phase results produced similar results to the single phase for comparable current steps. For an increase in output current of 5 A, the single phase converter dropped 124 mV whereas the five phase converter dropped 100 mV. With a step of 10 A increase in load, the single phase dropped 172 mV and the five phase board dropped 196 mV. This shows that the controller is interchangeable among the various boards with very similar results.

While the final results are suitable for autonomous efficiency data collection, there is still plenty of room for improvement and future projects. The addition of ceramic output capacitors was shown to improve the initial voltage drop from an increase in load. Due to limited board space, there was not much room to add more ceramic capacitors on the current designs. To fix this, another board design could be made with a long rail of output capacitor pads. The boards could then be assembled without populating the output capacitors to get a baseline test. Then one by one, the capacitors could be placed by hand with a test run after each addition. This would provide a more complete analysis of the effect on the output capacitance as well as determine the maximum number of ceramic capacitors that could be added before performance begins to degrade.

Another aspect that could be addressed is the minor fine tuning incremental controller. This was done to provide flexibility by only requiring a single control card to perform tests for one up to five active phases. To fix this issue, the PID controller could be tuned for a specific scenario, meaning the controller could be fine tuned for all possible numbers of active phases. Then a switch case could be implemented and the user could actively select the desired controlling algorithm via a keyboard input before running a test. This would improve the overall transient response of the controller while still maintaining the desired flexibility of being able to have one control card that is capable of testing all of the board designs.

Bibliography

[1] Zhaoxia Leng; Qingfeng Liu; Jinkun Sun; Jian Liu, *A Research of Efficiency Characteristic for Buck Converter*, ICIMA. 2nd International Conferance vol. 1, p. 232-235, 30-31 May 2010.

[2] S. Deuty, *Optimizing Transistor Performance in Synchronous Rectifier, Buck Converters*, APEC. Fifteenth Annual IEEE Vol. 2 p. 1078-1081, 2000.

[3] S. Deuty, *Analysis and Implementation of a DC-DC Step-Down Converter for Low Output-Voltage and High Output-Current Applications*, APEC. , IEEE International Symposium on Circuits and Systems, 3697-3700, 2010.

[4] Scott, M.J.; Ke Zou; Jin Wang; Chingchi Chen; Ming Su; Lihua Chen, *A Gallium Nitride Switched-Capacitor Circuit Using Synchronous Rectification*, IEEE Transactions vol 49 Iss. 3, p. 1383-1391, 2013.

[5] Villar, G.; Alarcon, E.; Guinjoan, F.; Poveda, A., *Quasi-optimum Efficiency in Output Voltage Hysteretic Control for a Buck Switching Converter with Wide Load Range*, Information and Automation (ICIA), 2010 IEEE International Conference on, p. 555-558, 20-23 Jun 2010.

[6] Jiang You; Hui Wang; Fanrong Meng; Jianwen Cui, *Analysis of Current Sharing and Controller Design Fundamental for Paralleled DC/DC Power Converters*, CSEE, vol. 24, p. 31-36, 2005.

[7] Sehirli, E.; Altinay, M.,*Input-output linearization control of single-phase buck-boost power factor corrector*, UPEC 2012, p. 1-6, 4-7 Sept. 2012.

[8] Il-Oun Lee; Shin-Young Cho; Gun-Woo Moon,*Interleaved buck converter having low switching losses and improved step-down conversion ratio*, ICPE & ECCE, 2011 IEEE 8th International Conference on, p. 2136-2143, 30 May 2011 - 3 June 2011.

[9] Zhe Zhang; Cuk, Slobodan, *A high efficiency 500 W step-up Cuk converter*, IPEMC 2000, vol. 2, p. 909-914, 2000.

[10] Castilla, M.; Garcia de Vicuna, L.; Guerrero, J.M.; Matas, J.; Miret, J, *Design of voltage-mode hysteretic controllers for synchronous buck converters supplying microprocessor loads*, Electrical Power Applications, IEE Procedings, vol. 152, no. 5, p. 1171-1178, 9 Sept. 2005.

[11] Shah, K.; Shenai, K., *Performance Evaluation of Point-of-Load Chip-Scale DC-DC Power Converters Using Silicon Power MOSFETs and GaN Power HEMTs*, Green technologies Conference (IEEE-Green), 2011 IEEE, pp. 1-5, 14-15 Apr. 2011.

[12] Shenai, K.; Scott, R.S.; Baliga, B. Jayant, emphOptimum semiconductors for high-power electronics, Electron Devices, IEEE Transactions on, vol. 36, no. 9, pp. 1811-1823, Sept. 1989.

[13] Shenai, K.; Shah, K.; Huili Xing, emphPerformance evaluation of silicon and gallium nitride power FETs for DC/DC power converter applications Aerospace and Electronics Conference (NAECON), Proceedings of the IEEE 2010 National, pp. 317-321. 14-16 July 2010.

[14] Lambert, W.J.; Ayyanar, R.; Chickamenahalli, S., *Fast Load Transient Regulation of Low-Voltage Converters with the Low-Voltage Transient Processor* Power Electronics, IEEE Transansactions on, VOl. 24, no. 7, pp. 1839-1854. July 2009.

[15] Xunwei Zhou, *Low-Voltage High-Efficiency Fast Transient Voltage Regulator Modules* Virginia Polytechnic Institue and State University, Dissertation submitted to the Faculty of the, July 1999

[16] Redl, R.; Erisman, B.P.; Zansky, Z., *Optimizing the load transient response of the buck converter* Applied Power Electronics Conference and Exposition, 1998. APEC '98, Conference Proceedings 1998., Thirteenth Annual, vol. 1 , pp. 170-176, 15-19 Feb. 1998.

[17] Ordonez, M.; Quaicoe, J.E.; Iqbal, M.T., *Critical Parameters in the Transient Response of Synchronous Buck Converters* Power Electronics Specialist Conference, 2007. PESC 2007. IEEE, pp. 2189-2195, 17-21 June 2007.

[18] Buccella, C.; Cecati, C.; Latafat, H., *Digital Control of Power ConvertersA Survey* Industrial Informatics, IEEE Transactions on, vol. 8, no. 3, pp. 437-447, Aug. 2012.

[19] White, Robert V., *Digital Control For Power Supply Engineers* Applied Power Electronics Conference, APEC 2013, IEEE International Symposium on Circuits and Systems, Technical Session Seminar 2, 17-21 Mar. 2013.

[20] Mujumdar, U.B.; Tutkane, D.R., *Selection of Digital Controller for Power System and Power Electronics Applications* Electronic Systems, Signal Processing and Computing Technologies (ICESC), 2014 International Confernce on, pp. 84-88, 9-11 Jan. 2014.

[21] Zheng Zhang; Kil-To Chong, *Comparison between first-order hold with zero-order hold in discretization of input-delay nonlinear systems*, Control, Automation and Systems 2007. ICCAS '07. International Conference on, p. 2892-2896, 17-20 Oct. 2007.

[22] MadhuKiran, E.R.C.S.; Thota, P.S.; Sridhar, B.; Dileesh, K., *Control of Buck converter by Polynomial, PID and PD controllers* Microelectronics and Electronics (PrimeAsia), 2012 Asia Pacific Conference on Postgraduate Research in, pp. 94-99. 5-7 Dec. 2012.

[23] Basso, Christophe. *Small-signal Modeling and Analytical Analysis of Power Converters* Applied Power Electronics Conference, APEC 2013, IEEE International Symposium on Circuits and Systems, Technical Session Seminar 16, 17-21 Mar. 2013.

[24] Yang Qiu; Ming Xu; Yao, K.; Juanjuan Sun; Lee, F.C., *Multifrequency Small-Signal Model for Buck and Multiphase Buck Converters* Power Electronics, IEEE Transactions on, vol. 21, no. 5, pp. 1185-1192, Sept. 2006.

[25] Yao, K.; Yu Meng; Peng Xu; Lee, F.C., *Design considerations for VRM transient response based on the output impedance* Applied Power Electronics Conference and Exposition, 2002. APEC 2002. Seventheenth Annual IEEE, pp. 14-20. 2002.

Appendices

Appendix A

Matlab Code For Discretization and Simulations Plots

```
%create transfer function from simulink block model

%constants
%HD printing fig size
fmtFig = .8;
pageWidth = 2.3;
T=0.5e-3;
%output directory to save figure
outputDirectory = 'C:\Users\bkr0001\Desktop\BodePlots';
outputDirectory = 'C:\Users\bkr0001\Desktop\digital_project';
%figure handle
h=figure;

%import model
sys=linmod('buck_hand');
% sys=linmod('buck_inter');
%create state space model
TFss=ss(sys.a,sys.b,sys.c,sys.d);
%get transfer funtion data for num and den
[n,d] = tfdata(TFss,'v');
%display trasfer function num and den
TFs=tf(n,d)
%create discrete model and display
TFz = c2d(TFs,T,'zoh')
%Check stability of the transfer functions
%returns one for stable, 0 for unstable
check_s = isstable(TFs);
check_z = isstable(TFz);
%controllability matrix
q=ctrb(sys.a,sys.b);
%display bode plots of both on same plot
bode(TFs,TFz)
%display a single bode plot with margins
margin(TFs)
%display a single bode plot without margins
```

```
bode(TFz)


%HD figure print
% outputFilename = ['bode_cont_only_both'];
outputFilename = ['test'];
outputStr = [outputDirectory '\' outputFilename '.jpg'];
oldSettings3 = fillPage(h, 'margins', [-3 -3 -1 -2.3],'papersize',[2*pageWidth pageWidth
print(h,'-djpeg','-r600', outputStr);
close(h);
%%
%constants
%HD printing fig size
fmtFig = .8;
pageWidth = 2.3;
T=5.8824e-06;
%output directory to save figure
outputDirectory = 'C:\Users\bkr0001\Desktop\Vout';
%figure handle
h=figure;

plot(Vout_2phase)
ylim([0.9,1.06])

%HD figure print
outputFilename = ['Vout_Block_Controller_close'];  %_close
outputStr = [outputDirectory '\' outputFilename '.jpg'];
oldSettings3 = fillPage(h, 'margins', [-3 -3 -1 -2.3],'papersize',[2*pageWidth pageWidth
print(h,'-djpeg','-r600', outputStr);
close(h);


%%
%poles and zeros plot
%constants
%HD printing fig size
fmtFig = .8;
pageWidth = 2.3;
%figure handle
h=figure;
pzplot(TFs)
% xlim([-2.5e6,.5e6])

outputFilename = ['continous_poles_new'];
% outputFilename = ['continous_poles_close'];
```

```
outputStr = [outputDirectory '\' outputFilename '.jpg'];
oldSettings3 = fillPage(h, 'margins', [-3 -3 -2.3 -2.3],'papersize',[2*pageWidth 2*pageW
print(h,'-djpeg','-r600', outputStr);
close(h);

%figure handle
h=figure;
pzplot(TFz)
% xlim([.9994,1.0002])

outputFilename = ['discrete_poles_new'];
% outputFilename = ['discrete_poles_close'];
outputStr = [outputDirectory '\' outputFilename '.jpg'];
oldSettings3 = fillPage(h, 'margins', [-3 -3 -2.3 -2.3],'papersize',[2*pageWidth 2*pageW
print(h,'-djpeg','-r600', outputStr);
close(h);
%%
plot(Vout)
title('Output Voltage with 100% Load Increase at t = 0.3 ms')
ylabel('Voltage (V)')
xlabel('Time (s)')
xlim([.235e-3,0.41e-3])
ylim([0.75,1.1])
%%
plot(Iout)
title('Output Current with 100% Load Increase at t = 0.3 ms')
ylabel('Current (A)')
xlabel('Time (s)')
%%
plot(duty)
title('PWM Input with 100% Load Increase at t = 0.3 ms')
ylabel('Magnitude')
xlabel('Time (s)')
ylim([-0.5,1.5])
xlim([.28e-3,0.35e-3])
```

Appendix B

PID Code

Included is all of the code relavent to this project which is based off of Texas Instruments TMS320F28335 controlCard documentation. Dr. Christopher Wilson configured the ADC, PWM, Interrpts, and serial communications. Benjamin Keaton Rhea implemented the PID controller in the Tasks.c file.

June2013Demo_F28335-Main.c

```
//---------------------------------------------------------------------------
// FILE: June2013Demo_F28335-Main.C
//
// Description: Program for running 5 Buck V3.4j or higher on the BuckCarrierV1
// board. Responsible for the following:
// 1) Regulate output voltage to 1V.
// 2) Keep currents approximately shared.
// 3) Respond to system requests for information via SCI-A
//
// Version:  1.0
//
//  Target:   TMS320F28335 on ControlCard interfaced to BuckCarrierV1
//---------------------------------------------------------------------------
//  Copyright Auburn University 2013
//---------------------------------------------------------------------------
//  Revision History:
//---------------------------------------------------------------------------
//  Date   | Description / Status
//---------------------------------------------------------------------------
// 4 June 2013 - Initial Writing.
//---------------------------------------------------------------------------
//
// PLEASE READ - Useful notes about this Project

// Although this project is made up of several files, the most important ones are:
//  "June2013Demo_F28335-Main.c" - this file
// - Application init and Peripheral config
//  "June2013Demo_F28335-DevInit_F28xxx.C
// - Device related init, e.g. Clock, PLL, WD, GPIO mapping
// - Peripheral clock enables
```

```
// - DevInit file will differ per each F28xxx device series, e.g. F280x, F2833x,
//  "June2013Demo_F28335-Settings.h"
// - Global defines (settings) project selections are found here
// - This file is referenced by both C and ASM files.


#include "June2013Demo_F28335-Settings.h"
#include "PeripheralHeaderIncludes_F28335.h"
#include "Globals.h"
#include "PWMHandler.h"
#include "ADCHandler.h"
#include "Interrupts.h"
#include "SerialHandler.h"
#include "Tasks.h"


//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// FUNCTION PROTOTYPES
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

// ---------------------------------- FRAMEWORK ------------------------------------
void DeviceInit(void);
void SCIA_Init();
void SerialHostComms();
void InitFlash();
void MemCopy();




// ---------------------------------- USER -----------------------------------------



//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// VARIABLE DECLARATIONS - GENERAL
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

// ---------------------------------- FRAMEWORK ------------------------------------

// Used for running BackGround in flash, and ISR in RAM
extern Uint16 *RamfuncsLoadStart, *RamfuncsLoadEnd, *RamfuncsRunStart;
```

```
// ---------------------------------- USER ---------------------------------------


//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// VARIABLE DECLARATIONS - CCS WatchWindow / GUI support
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

// ---------------------------------- FRAMEWORK ----------------------------------

//GUI support variables
// sets a limit on the amount of external GUI controls - increase as necessary
int16  *varSetTxtList[8]; //8 textbox controlled variables
int16  *varSetBtnList[8]; //8 button controlled variables
int16  *varSetSldrList[8]; //8 slider controlled variables
int16  *varGetList[8]; //8 variables sendable to GUI
int16  *arrayGetList[8]; //8 arrays sendable to GUI



//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// MAIN CODE - starts here
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
void main(void)
{
//=============================================================================
// INITIALIZATION - General
//=============================================================================
unsigned char i;
//---------------------------------- FRAMEWORK ----------------------------------

DeviceInit(); // Device Life support & GPIO
InitSciaGpio();
// SCIA_Init();   // Initalize the Serial Comms A peripheral

// Only used if running from FLASH
// Note that the variable FLASH is defined by the compiler with -d FLASH
// (see the project's Build Properties)
#ifdef FLASH
// Copy time critical code and Flash setup code to RAM
// The  RamfuncsLoadStart, RamfuncsLoadEnd, and RamfuncsRunStart
// symbols are created by the linker. Refer to the linker files.
MemCopy(&RamfuncsLoadStart, &RamfuncsLoadEnd, &RamfuncsRunStart);
```

```
// Call Flash Initialization to setup flash waitstates
// This function must reside in RAM
InitFlash(); // Call the flash wrapper init function
#endif //(FLASH)




// Initialize period for each CPU Timer (used by background loops)
// Timer period definitions found in PeripheralHeaderIncludes.h
CpuTimer0Regs.PRD.all =  mSec1; // A tasks
CpuTimer1Regs.PRD.all =  mSec50; // B tasks
CpuTimer2Regs.PRD.all =  mSec1000; // C tasks

// Tasks State-machine init


CommsOKflg = 0;
SerialCommsTimer = 0;

//=======
//  ISR
//=======
InitInterrupts();

//===========================================================================
// INITIALIZATION - Peripherals used
//===========================================================================

// ---------------------------------- USER -----------------------------------
//  Put peripheral initialization here




//===========================================================================
// INITIALIZATION - BUILD OPTIONS - Change in FlashingLeds-Settings.h
//===========================================================================

// ---------------------------------- USER -----------------------------------

//-----------------------------------------------------
#if (INCR_BUILD == 1)
//-----------------------------------------------------
```

```
LedBlinkTimer = 0; //Initialize LedBlinkTimer to 0
Gui_LedPrd_ms = 1000; //Default to 1 blink every second

#endif // (INCR_BUILD == 1)


//-------------------------------------------------
#if (INCR_BUILD == 2)
//-------------------------------------------------

LedBlinkTimer = 0; //Initialize LedBlinkTimer to 0
Gui_LedPrd_ms = 2000;  //Default to 2 blinks each second

#endif // (INCR_BUILD == 2)


GpioDataRegs.GPATOGGLE.bit.GPIO31 = 1; //Initially toggle the pin.

InitPwmModule();
InitADCModule();
scia_fifo_init_async();    // Initialize the SCI FIFO for async comm.
    scia_echoback_init();  // Initalize SCI for comms.

EnableInterrupts();
   EALLOW;
msg = "\r\nWelcome to PowerControl!\0";
    scia_msg_async(msg);

TaskMachineInit();
//===========================================================================
// BACKGROUND (BG) LOOP
//===========================================================================
for(;;)  //infinite loop
{
if(MainFlags.bit.AdcComputeFlag){
for(i = 0; i < 10; i++){
SampleTable[i] = (SampleTable[10+4*i] + SampleTable[11+4*i] + SampleTable[12+4*i] + Samp
}
MainFlags.bit.AdcComputeFlag = 0;
}
// State machine entry & exit point
//===========================================================
(*Alpha_State_Ptr)(); // jump to an Alpha state (A0,B0,...)
//===========================================================
```

```
}
} //END MAIN CODE

Globals.h
#ifndef GLOBALS_H_
#define GLOBALS_H_

#include "PeripheralHeaderIncludes_F28335.h"
//#include "DSP28x_Project.h"     // Device Headerfile and Examples Include File

//extern Uint16 LoopCount;
extern Uint16 ErrorCount;
extern Uint16 ConversionCount;
//extern Uint16 XmitFlag;
//extern Uint16 I2CTXFlag;
//extern Uint16 I2CRXFlag;
//extern Uint16 PhaseStatus;
//extern Uint16 LEDStatus;

extern Uint16 voutRef;
extern Uint16 epsilon;
extern Uint16 epsilonSmall;

struct MAINFLAGBITS{        // bits    description
Uint16 XmitFlag:1;     // 0 Transmit ADC information back
Uint16  I2CTXFlag:1;  // 1 Transmit Something over I2C
Uint16 I2CRXFlag:1;       // 2 Receive something over I2C
Uint16 Timer0Flag:1; // 3 Do a low priority background task on timer0
Uint16  AdcTxFlag:1;  // 4 Reserved
Uint16  AdcComputeFlag:1; // 5 Reserved
Uint16  rsv6:1; // 6 Reserved
Uint16  rsv7:1; // 7 Reserved
Uint16 rsv8:1; // 8 Reserved
Uint16 rsv9:1; // 9 Reserved
Uint16  rsv10:1; // 10 Reserved
Uint16  rsv11:1; // 11 Reserved
Uint16  rsv12:1; // 12 Reserved
Uint16  rsv13:1; // 13 Reserved
Uint16  rsv14:1; // 14 Reserved
Uint16  rsv15:1; // 15 Reserved
};

union MAINFLAGS{
Uint16 all;
struct MAINFLAGBITS bit;
```

```
};

extern union MAINFLAGS MainFlags;

struct PHASESTATUSBITS{     // bits    description
Uint16 Phase1:1;     // 0
Uint16  Phase2:1;  // 1
Uint16 Phase3:1;       // 2
Uint16 Phase4:1; // 3
Uint16  Phase5:1;  // 4
Uint16  Phase6:1; // 5
Uint16  Phase1ADC:1; // 6
Uint16  Phase2ADC:1; // 7
Uint16 Phase3ADC:1; // 8
Uint16 Phase4ADC:1; // 9
Uint16  Phase5ADC:1; // 10
Uint16  Phase6ADC:1; // 11
Uint16  Controller:1; // 12
Uint16  rsv13:1; // 13 Reserved
Uint16  rsv14:1; // 14 Reserved
Uint16  rsv15:1; // 15 Reserved
};

union PHASEFLAGS{
Uint16 all;
struct PHASESTATUSBITS bit;
};

extern union PHASEFLAGS PhaseStatus;

struct BUTTONBITS{        // bits    description
Uint16 PB1:1;     // 0 Transmit ADC information back
Uint16  PB2:1;  // 1 Transmit Something over I2C
Uint16 PB3:1;       // 2 Receive something over I2C
Uint16 PB4:1; // 3 Do a low priority background task on timer0
Uint16  SW1_1:1;  // 4 Reserved
Uint16  SW1_2:1; // 5 Reserved
Uint16  SW1_3:1; // 6 Reserved
Uint16  SW1_4:1; // 7 Reserved
Uint16 rsv8:1; // 8 Reserved
Uint16 rsv9:1; // 9 Reserved
Uint16  rsv10:1; // 10 Reserved
Uint16  rsv11:1; // 11 Reserved
Uint16  rsv12:1; // 12 Reserved
Uint16  rsv13:1; // 13 Reserved
```

```
Uint16  rsv14:1; // 14 Reserved
Uint16  rsv15:1; // 15 Reserved
};
struct BUTTONNIBBLE{        // bits    description
Uint16 PB:4;     // 0 Transmit ADC information back
Uint16  SW1:4;  // 4 Reserved
Uint16 rsv8:1; // 8 Reserved
Uint16 rsv9:1; // 9 Reserved
Uint16  rsv10:1; // 10 Reserved
Uint16  rsv11:1; // 11 Reserved
Uint16  rsv12:1; // 12 Reserved
Uint16  rsv13:1; // 13 Reserved
Uint16  rsv14:1; // 14 Reserved
Uint16  rsv15:1; // 15 Reserved
};
union BUTTONS{
Uint16 all;
struct BUTTONBITS bit;
struct BUTTONNIBBLE nib;
};

extern union BUTTONS ButtonStatus;

struct LEDBITS{        // bits    description
Uint16 LED0:2;     // 0 Transmit ADC information back
Uint16  LED1:2;  // 4 Reserved
Uint16 LED2:2; // 8 Reserved
Uint16 LED3:2; // 9 Reserved
Uint16  LED4:2; // 10 Reserved
Uint16  LED5:2; // 11 Reserved
Uint16  LED6:2; // 12 Reserved
Uint16  LED7:2; // 13 Reserved
};
union LEDS{
Uint16 all;
struct LEDBITS bit;
};

extern union LEDS LEDStatus;
extern union LEDS LEDStatus2;

#define PHASE1ON  0x0001
#define PHASE2ON  0x0002
#define PHASE3ON  0x0004
#define PHASE4ON  0x0008
```

```
#define PHASE5ON  0x0010
#define PHASE6ON  0x0020
#define ALLPHASESON  0x003F
#define PHASE1OFF 0xFFFE
#define PHASE2OFF 0xFFFD
#define PHASE3OFF 0xFFFB
#define PHASE4OFF 0xFFF7
#define PHASE5OFF 0xFFEF
#define PHASE6OFF 0xFFDF
#define ALLPHASESOFF  0xFFC0
#define PHASE1ADCON  0x0040
#define PHASE2ADCON  0x0080
#define PHASE3ADCON  0x0100
#define PHASE4ADCON  0x0200
#define PHASE5ADCON  0x0400
#define PHASE6ADCON  0x0800
#define ALLPHASESADCON  0x0FC0
#define PHASE1ADCOFF 0xFFBF
#define PHASE2ADCOFF 0xFF7F
#define PHASE3ADCOFF 0xFEFF
#define PHASE4ADCOFF 0xFDFF
#define PHASE5ADCOFF 0xFBFF
#define PHASE6ADCOFF 0xF7FF
#define ALLPHASESADCOFF 0xF03F
#define CONTROLLERON 0x1000
#define CONTROLLEROFF 0xEFFF

#endif /*GLOBALS_H_*/
```

FlashingLeds-DevInit_F2833x.c

```
//===========================================================================
//===========================================================================
//
// FILE: FlashingLeds-DevInit_F2833x.c
//
// TITLE: Device initialization for F2833x series
//
// Version: 05 Aug 09
//===========================================================================
//===========================================================================
#include "PeripheralHeaderIncludes.h" // Include all Peripheral Headers

// Functions that will be run from RAM need to be assigned to
// a different section.  This section will then be mapped to a load and
// run address using the linker cmd file.
```

```
#pragma CODE_SECTION(InitFlash, "ramfuncs");

// Function prototypes
void DeviceInit(void);
void PieCntlInit(void);
void PieVectTableInit(void);
void WDogDisable(void);
void PLLset(Uint16);
void ISR_ILLEGAL(void);

//---------------------------------------------------------------------
//  Configure Device for target Application Here
//---------------------------------------------------------------------
void DeviceInit(void)
{
WDogDisable();  // Disable the watchdog initially
DINT; // Global Disable all Interrupts
IER = 0x0000; // Disable CPU interrupts
IFR = 0x0000; // Clear all CPU interrupt flags

// SYSTEM CLOCK speed based on Crystal = 20 MHz (R1.1 controlCARD)
// 0xF =  150   MHz (15)
// 0xE =  140   MHz (14)
// 0xD =  130   MHz (13)
// 0xC =  120   MHz (12)
// 0xB =  110   MHz (11)
// 0xA =  100 MHz (10)
// 0x9 =  90 MHz (9)
// 0x8 =  80 MHz (8)
// 0x7 =  70 MHz (7)
// 0x6 =  60 MHz (6)
// 0x5 =  50 MHz (5)
// 0x4 =  40 MHz (4)
// 0x3 =  30 MHz (3)
// 0x2 =  20 MHz (2)

// SYSTEM CLOCK speed based on Crystal = 25 MHz (R1 controlCARD)
// 0xF =  187.5 MHz (15)
// 0xE =  175   MHz (14)
// 0xD =  162.5 MHz (13)
// 0xC =  150   MHz (12)
// 0xB =  137.5 MHz (11)
// 0xA =  125 MHz (10)
// 0x9 =  112.5 MHz (9)
// 0x8 =  100 MHz (8)
```

```
// 0x7 =  87.5 MHz (7)
// 0x6 =  75 MHz (6)
// 0x5 =  62.5 MHz (5)
// 0x4 =  50 MHz (4)
// 0x3 =  37.5 MHz (3)
// 0x2 =  25 MHz (2)

// SYSTEM CLOCK speed based on Crystal = 30 MHz (R1.2 controlCARD)
// 0xA =  150 MHz (10)
// 0x9 =  135 MHz (9)
// 0x8 =  120 MHz (8)
// 0x7 =  105 MHz (7)
// 0x6 =  90 MHz (6)
// 0x5 =  75 MHz (5)
// 0x4 =  60 MHz (4)
// 0x3 =  45 MHz (3)
// 0x2 =  30 MHz (2)

PLLset(0xA); // choose from options above based on the reference crystal

// Initialise interrupt controller and Vector Table
// to defaults for now. Application ISR mapping done later.
PieCntlInit();
PieVectTableInit();

   EALLOW; // below registers are "protected", allow access.

// HIGH / LOW SPEED CLOCKS prescale register settings
   SysCtrlRegs.HISPCP.all = 0x0002; // Sysclk / 4 (25 MHz)
   SysCtrlRegs.LOSPCP.all = 0x0002; // Sysclk / 4 (25 MHz)


// PERIPHERAL CLOCK ENABLES
//---------------------------------------------------
// If you are not using a peripheral you may want to switch
// the clock off to save power, i.e. set to =0
//
// Note: not all peripherals are available on all 280x derivates.
// Refer to the datasheet for your particular device.

   SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 1;    // ADC
   //-----------------------------------------------
   SysCtrlRegs.PCLKCR0.bit.I2CAENCLK = 1;   // I2C
   //-----------------------------------------------
   SysCtrlRegs.PCLKCR0.bit.SPIAENCLK=1;     // SPI-A
```

```
   //--------------------------------------------------
   SysCtrlRegs.PCLKCR0.bit.SCIAENCLK=1;     // SCI-A
   SysCtrlRegs.PCLKCR0.bit.SCIBENCLK=0;     // SCI-B
   //--------------------------------------------------
   SysCtrlRegs.PCLKCR0.bit.ECANAENCLK=0;    // eCAN-A
   SysCtrlRegs.PCLKCR0.bit.ECANBENCLK=0;    // eCAN-B
   //--------------------------------------------------
   SysCtrlRegs.PCLKCR1.bit.ECAP1ENCLK = 0;  // eCAP1
   SysCtrlRegs.PCLKCR1.bit.ECAP2ENCLK = 0;  // eCAP2
   SysCtrlRegs.PCLKCR1.bit.ECAP3ENCLK = 0;  // eCAP3
   SysCtrlRegs.PCLKCR1.bit.ECAP4ENCLK = 0;  // eCAP4
   //--------------------------------------------------
   SysCtrlRegs.PCLKCR1.bit.EPWM1ENCLK = 1;   // ePWM1
   SysCtrlRegs.PCLKCR1.bit.EPWM2ENCLK = 1;   // ePWM2
   SysCtrlRegs.PCLKCR1.bit.EPWM3ENCLK = 1;   // ePWM3
   SysCtrlRegs.PCLKCR1.bit.EPWM4ENCLK = 1;   // ePWM4
   SysCtrlRegs.PCLKCR1.bit.EPWM5ENCLK = 1;   // ePWM5
   SysCtrlRegs.PCLKCR1.bit.EPWM6ENCLK = 1;   // ePWM6
   //--------------------------------------------------
   SysCtrlRegs.PCLKCR1.bit.EQEP1ENCLK = 0;   // eQEP1
   SysCtrlRegs.PCLKCR1.bit.EQEP2ENCLK = 0;   // eQEP2
   //--------------------------------------------------
   SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;   // Enable TBCLK
   //--------------------------------------------------


// GPIO (GENERAL PURPOSE I/O) CONFIG
//-----------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------
// QUICK NOTES on GPIO CONFIG USAGE:
//----------------------------------
// If GpioCtrlRegs.GP?MUX?bit.GPIO?= 1, 2 or 3 (i.e. Non GPIO func), then leave
// rest of lines commented
// If GpioCtrlRegs.GP?MUX?bit.GPIO?= 0 (i.e. GPIO func), then:
//   1) uncomment GpioCtrlRegs.GP?DIR.bit.GPIO? = ? and choose pin to be IN or OUT direc
//   2) If IN, can leave next two lines commented
//   3) If OUT, uncomment line with ..GPACLEAR.. to force pin LOW or
//      uncomment line with ..GPASET.. to force pin HIGH
//-----------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------
//  GPIO-00 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 0; // 0=GPIO,  1=EPWM1A,  2=Resv,  3=Resv
GpioCtrlRegs.GPADIR.bit.GPIO0 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPACLEAR.bit.GPIO0 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPASET.bit.GPIO0 = 1; // uncomment if --> Set High initially
```

```
//---------------------------------------------------------------------------
//  GPIO-01 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPAMUX1.bit.GPIO1 = 0; // 0=GPIO,  1=EPWM1B,  2=ECAP6,  3=MFSR-B
GpioCtrlRegs.GPADIR.bit.GPIO1 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPACLEAR.bit.GPIO1 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPASET.bit.GPIO1 = 1; // uncomment if --> Set High initially
//---------------------------------------------------------------------------
//  GPIO-02 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPAMUX1.bit.GPIO2 = 0; // 0=GPIO,  1=EPWM2A,  2=Resv,  3=Resv
GpioCtrlRegs.GPADIR.bit.GPIO2 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPACLEAR.bit.GPIO2 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPASET.bit.GPIO2 = 1; // uncomment if --> Set High initially
//---------------------------------------------------------------------------
//  GPIO-03 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPAMUX1.bit.GPIO3 = 0; // 0=GPIO,  1=EPWM2B,  2=ECAP5,  3=MCLKR-B
GpioCtrlRegs.GPADIR.bit.GPIO3 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPACLEAR.bit.GPIO3 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPASET.bit.GPIO3 = 1; // uncomment if --> Set High initially
//---------------------------------------------------------------------------
//  GPIO-04 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPAMUX1.bit.GPIO4 = 0; // 0=GPIO,  1=EPWM3A,  2=Resv,  3=Resv
GpioCtrlRegs.GPADIR.bit.GPIO4 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPACLEAR.bit.GPIO4 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPASET.bit.GPIO4 = 1; // uncomment if --> Set High initially
//---------------------------------------------------------------------------
//  GPIO-05 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPAMUX1.bit.GPIO5 = 0; // 0=GPIO,  1=EPWM3B,  2=MFSR-A,  3=ECAP1
GpioCtrlRegs.GPADIR.bit.GPIO5 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPACLEAR.bit.GPIO5 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPASET.bit.GPIO5 = 1; // uncomment if --> Set High initially
//---------------------------------------------------------------------------
//  GPIO-06 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPAMUX1.bit.GPIO6 = 0; // 0=GPIO,  1=EPWM4A,  2=SYNCI,  3=SYNCO
GpioCtrlRegs.GPADIR.bit.GPIO6 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPACLEAR.bit.GPIO6 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPASET.bit.GPIO6 = 1; // uncomment if --> Set High initially
//---------------------------------------------------------------------------
//  GPIO-07 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPAMUX1.bit.GPIO7 = 0; // 0=GPIO,  1=EPWM4B,  2=MCLKR-A,  3=ECAP2
GpioCtrlRegs.GPADIR.bit.GPIO7 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPACLEAR.bit.GPIO7 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPASET.bit.GPIO7 = 1; // uncomment if --> Set High initially
//---------------------------------------------------------------------------
//  GPIO-08 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPAMUX1.bit.GPIO8 = 0; // 0=GPIO,  1=EPWM5A,  2=CANTX-B,  3=ADCSOC-A
```

```
GpioCtrlRegs.GPADIR.bit.GPIO8 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPACLEAR.bit.GPIO8 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPASET.bit.GPIO8 = 1; // uncomment if --> Set High initially
//-----------------------------------------------------------------------------
//  GPIO-09 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPAMUX1.bit.GPIO9 = 0; // 0=GPIO,  1=EPWM5B,  2=SCITX-B,  3=ECAP3
GpioCtrlRegs.GPADIR.bit.GPIO9 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPACLEAR.bit.GPIO9 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPASET.bit.GPIO9 = 1; // uncomment if --> Set High initially
//-----------------------------------------------------------------------------
//  GPIO-10 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPAMUX1.bit.GPIO10 = 0; // 0=GPIO,  1=EPWM6A,  2=CANRX-B,  3=ADCSOC-B
GpioCtrlRegs.GPADIR.bit.GPIO10 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPACLEAR.bit.GPIO10 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPASET.bit.GPIO10 = 1; // uncomment if --> Set High initially
//-----------------------------------------------------------------------------
//  GPIO-11 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPAMUX1.bit.GPIO11 = 0; // 0=GPIO,  1=EPWM6B,  2=SCIRX-B,  3=ECAP4
GpioCtrlRegs.GPADIR.bit.GPIO11 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPACLEAR.bit.GPIO11 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPASET.bit.GPIO11 = 1; // uncomment if --> Set High initially
//-----------------------------------------------------------------------------
//  GPIO-12 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPAMUX1.bit.GPIO12 = 0; // 0=GPIO,  1=TZ1,  2=CANTX-B,  3=MDX-B
GpioCtrlRegs.GPADIR.bit.GPIO12 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPACLEAR.bit.GPIO12 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPASET.bit.GPIO12 = 1; // uncomment if --> Set High initially
//-----------------------------------------------------------------------------
//  GPIO-13 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPAMUX1.bit.GPIO13 = 0; // 0=GPIO,  1=TZ2,  2=CANRX-B,  3=MDR-B
GpioCtrlRegs.GPADIR.bit.GPIO13 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPACLEAR.bit.GPIO13 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPASET.bit.GPIO13 = 1; // uncomment if --> Set High initially
//-----------------------------------------------------------------------------
//  GPIO-14 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPAMUX1.bit.GPIO14 = 0; // 0=GPIO,  1=TZ3,  2=SCITX-B,  3=MCLKX-B
GpioCtrlRegs.GPADIR.bit.GPIO14 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPACLEAR.bit.GPIO14 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPASET.bit.GPIO14 = 1; // uncomment if --> Set High initially
//-----------------------------------------------------------------------------
//  GPIO-15 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPAMUX1.bit.GPIO15 = 0; // 0=GPIO,  1=TZ4,  2=SCIRX-B,  3=MFSX-B
GpioCtrlRegs.GPADIR.bit.GPIO15 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPACLEAR.bit.GPIO15 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPASET.bit.GPIO15 = 1; // uncomment if --> Set High initially
```

```
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
//  GPIO-16 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPAMUX2.bit.GPIO16 = 0; // 0=GPIO,  1=SPISIMO-A,  2=CANTX-B,  3=TZ5
GpioCtrlRegs.GPADIR.bit.GPIO16 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPACLEAR.bit.GPIO16 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPASET.bit.GPIO16 = 1; // uncomment if --> Set High initially
//---------------------------------------------------------------------------
//  GPIO-17 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPAMUX2.bit.GPIO17 = 0; // 0=GPIO,  1=SPISOMI-A,  2=CANRX-B,  3=TZ6
GpioCtrlRegs.GPADIR.bit.GPIO17 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPACLEAR.bit.GPIO17 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPASET.bit.GPIO17 = 1; // uncomment if --> Set High initially
//---------------------------------------------------------------------------
//  GPIO-18 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPAMUX2.bit.GPIO18 = 0; // 0=GPIO,  1=SPICLK-A,  2=SCITX-B,  3=CANRX-A
GpioCtrlRegs.GPADIR.bit.GPIO18 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPACLEAR.bit.GPIO18 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPASET.bit.GPIO18 = 1; // uncomment if --> Set High initially
//---------------------------------------------------------------------------
//  GPIO-19 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPAMUX2.bit.GPIO19 = 0; // 0=GPIO,  1=SPISTE-A,  2=SCIRX-B,  3=CANTX-A
GpioCtrlRegs.GPADIR.bit.GPIO19 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPACLEAR.bit.GPIO19 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPASET.bit.GPIO19 = 1; // uncomment if --> Set High initially
//---------------------------------------------------------------------------
//  GPIO-20 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPAMUX2.bit.GPIO20 = 0; // 0=GPIO,  1=EQEPA-1,  2=MDX-A,  3=CANTX-B
GpioCtrlRegs.GPADIR.bit.GPIO20 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPACLEAR.bit.GPIO20 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPASET.bit.GPIO20 = 1; // uncomment if --> Set High initially
//---------------------------------------------------------------------------
//  GPIO-21 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPAMUX2.bit.GPIO21 = 0; // 0=GPIO,  1=EQEPB-1,  2=MDR-A,  3=CANRX-B
GpioCtrlRegs.GPADIR.bit.GPIO21 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPACLEAR.bit.GPIO21 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPASET.bit.GPIO21 = 1; // uncomment if --> Set High initially
//---------------------------------------------------------------------------
//  GPIO-22 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPAMUX2.bit.GPIO22 = 0; // 0=GPIO,  1=EQEPS-1,  2=MCLKX-A,  3=SCITX-B
GpioCtrlRegs.GPADIR.bit.GPIO22 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPACLEAR.bit.GPIO22 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPASET.bit.GPIO22 = 1; // uncomment if --> Set High initially
//---------------------------------------------------------------------------
//  GPIO-23 - PIN FUNCTION = --Spare--
```

```
GpioCtrlRegs.GPAMUX2.bit.GPIO23 = 0; // 0=GPIO,  1=EQEPI-1,  2=MFSX-A,  3=SCIRX-B
GpioCtrlRegs.GPADIR.bit.GPIO23 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPACLEAR.bit.GPIO23 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPASET.bit.GPIO23 = 1; // uncomment if --> Set High initially
//-----------------------------------------------------------------------------
//  GPIO-24 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPAMUX2.bit.GPIO24 = 0; // 0=GPIO,  1=ECAP1,  2=EQEPA-2,  3=MDX-B
GpioCtrlRegs.GPADIR.bit.GPIO24 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPACLEAR.bit.GPIO24 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPASET.bit.GPIO24 = 1; // uncomment if --> Set High initially
//-----------------------------------------------------------------------------
//  GPIO-25 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPAMUX2.bit.GPIO25 = 0; // 0=GPIO,  1=ECAP2,  2=EQEPB-2,  3=MDR-B
GpioCtrlRegs.GPADIR.bit.GPIO25 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPACLEAR.bit.GPIO25 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPASET.bit.GPIO25 = 1; // uncomment if --> Set High initially
//-----------------------------------------------------------------------------
//  GPIO-26 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPAMUX2.bit.GPIO26 = 0; // 0=GPIO,  1=ECAP3,  2=EQEPI-2,  3=MCLKX-B
GpioCtrlRegs.GPADIR.bit.GPIO26 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPACLEAR.bit.GPIO26 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPASET.bit.GPIO26 = 1; // uncomment if --> Set High initially
//-----------------------------------------------------------------------------
//  GPIO-27 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPAMUX2.bit.GPIO27 = 0; // 0=GPIO,  1=ECAP4,  2=EQEPS-2,  3=MFSX-B
GpioCtrlRegs.GPADIR.bit.GPIO27 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPACLEAR.bit.GPIO27 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPASET.bit.GPIO27 = 1; // uncomment if --> Set High initially
//-----------------------------------------------------------------------------
//  GPIO-28 - PIN FUNCTION = SCI-RX
GpioCtrlRegs.GPAMUX2.bit.GPIO28 = 1; // 0=GPIO,  1=SCIRX-A,  2=Resv,  3=Resv
// GpioCtrlRegs.GPADIR.bit.GPIO28 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPACLEAR.bit.GPIO28 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPASET.bit.GPIO28 = 1; // uncomment if --> Set High initially
//-----------------------------------------------------------------------------
//  GPIO-29 - PIN FUNCTION = SCI-TX
GpioCtrlRegs.GPAMUX2.bit.GPIO29 = 1; // 0=GPIO,  1=SCITXD-A,  2=XA19,  3=Resv
// GpioCtrlRegs.GPADIR.bit.GPIO29 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPACLEAR.bit.GPIO29 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPASET.bit.GPIO29 = 1; // uncomment if --> Set High initially
//-----------------------------------------------------------------------------
//  GPIO-30 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPAMUX2.bit.GPIO30 = 0; // 0=GPIO,  1=CANRX-A,  2=XA18,  3=Resv
GpioCtrlRegs.GPADIR.bit.GPIO30 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPACLEAR.bit.GPIO30 = 1; // uncomment if --> Set Low initially
```

```
// GpioDataRegs.GPASET.bit.GPIO30 = 1; // uncomment if --> Set High initially
//---------------------------------------------------------------------------
//  GPIO-31 - PIN FUNCTION = LED2 (for Release 1.1 and up F2833x controlCARDs)
GpioCtrlRegs.GPAMUX2.bit.GPIO31 = 0; // 0=GPIO,  1=CANTX-A,  2=XA17,  3=Resv
GpioCtrlRegs.GPADIR.bit.GPIO31 = 1; // 1=OUTput,  0=INput
// GpioDataRegs.GPACLEAR.bit.GPIO31 = 1; // uncomment if --> Set Low initially
GpioDataRegs.GPASET.bit.GPIO31 = 1; // uncomment if --> Set High initially
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
//  GPIO-32 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPBMUX1.bit.GPIO32 = 0; // 0=GPIO,  1=I2C-SDA,  2=SYNCI,  3=ADCSOCA
GpioCtrlRegs.GPBDIR.bit.GPIO32 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPBCLEAR.bit.GPIO32 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPBSET.bit.GPIO32 = 1; // uncomment if --> Set High initially
//---------------------------------------------------------------------------
//  GPIO-33 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPBMUX1.bit.GPIO33 = 0; // 0=GPIO,  1=I2C-SCL,  2=SYNCO,  3=ADCSOCB
GpioCtrlRegs.GPBDIR.bit.GPIO33 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPBCLEAR.bit.GPIO33 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPBSET.bit.GPIO33 = 1; // uncomment if --> Set High initially
//---------------------------------------------------------------------------
//  GPIO-34 - PIN FUNCTION = LED3 (for Release 1.1 and up F2833x controlCARDs)
GpioCtrlRegs.GPBMUX1.bit.GPIO34 = 0; // 0=GPIO,  1=ECAP1,  2=Resv,  3=Resv
GpioCtrlRegs.GPBDIR.bit.GPIO34 = 1; // 1=OUTput,  0=INput
// GpioDataRegs.GPBCLEAR.bit.GPIO34 = 1; // uncomment if --> Set Low initially
GpioDataRegs.GPBSET.bit.GPIO34 = 1; // uncomment if --> Set High initially
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
//  GPIO-35 - PIN FUNCTION = --Spare-- (SCI-TX on R1 F2833x controlCARD)
GpioCtrlRegs.GPBMUX1.bit.GPIO35 = 0; // 0=GPIO,  1=SCIA-TX,  2=Resv,  3=Resv
GpioCtrlRegs.GPBDIR.bit.GPIO35 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPBCLEAR.bit.GPIO35 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPBSET.bit.GPIO35 = 1; // uncomment if --> Set High initially
//---------------------------------------------------------------------------
//  GPIO-36 - PIN FUNCTION = --Spare-- (SCI-RX on R1 F2833x controlCARD)
GpioCtrlRegs.GPBMUX1.bit.GPIO36 = 0; // 0=GPIO,  1=SCIA-RX,  2=Resv,  3=Resv
GpioCtrlRegs.GPBDIR.bit.GPIO36 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPBCLEAR.bit.GPIO36 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPBSET.bit.GPIO36 = 1; // uncomment if --> Set High initially
//---------------------------------------------------------------------------
//  GPIO-38 - PIN FUNCTION = LED2 (for Release 1 F2833x controlCARDs)
GpioCtrlRegs.GPBMUX1.bit.GPIO38 = 0; // 0=GPIO,  1=Resv,  2=Resv,  3=Resv
GpioCtrlRegs.GPBDIR.bit.GPIO38 = 1; // 1=OUTput,  0=INput
// GpioDataRegs.GPBCLEAR.bit.GPIO38 = 1; // uncomment if --> Set Low initially
GpioDataRegs.GPBSET.bit.GPIO38 = 1; // uncomment if --> Set High initially
```

```
//---------------------------------------------------------------------------
//  GPIO-39 - PIN FUNCTION = LED3 (for Release 1 F2833x controlCARDs)
GpioCtrlRegs.GPBMUX1.bit.GPIO39 = 0; // 0=GPIO,  1=Resv,  2=XA16,  3=Resv
GpioCtrlRegs.GPBDIR.bit.GPIO39 = 1; // 1=OUTput,  0=INput
// GpioDataRegs.GPBCLEAR.bit.GPIO39 = 1; // uncomment if --> Set Low initially
GpioDataRegs.GPBSET.bit.GPIO39 = 1; // uncomment if --> Set High initially
//---------------------------------------------------------------------------

//  GPIO-48 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPBMUX2.bit.GPIO48 = 0; // 0=GPIO,  1=ECAP5,  2=XD31,  3=Resv
GpioCtrlRegs.GPBDIR.bit.GPIO48 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPBCLEAR.bit.GPIO48 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPBSET.bit.GPIO48 = 1; // uncomment if --> Set High initially
//---------------------------------------------------------------------------
//  GPIO-49 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPBMUX2.bit.GPIO49 = 0; // 0=GPIO,  1=ECAP6,  2=XD30,  3=Resv
GpioCtrlRegs.GPBDIR.bit.GPIO49 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPBCLEAR.bit.GPIO49 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPBSET.bit.GPIO49 = 1; // uncomment if --> Set High initially
//---------------------------------------------------------------------------

//  GPIO-58 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPBMUX2.bit.GPIO58 = 0; // 0=GPIO,  1=MCLKR-A,  2=XD21,  3=Resv
GpioCtrlRegs.GPBDIR.bit.GPIO58 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPBCLEAR.bit.GPIO58 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPBSET.bit.GPIO58 = 1; // uncomment if --> Set High initially
//---------------------------------------------------------------------------
//  GPIO-59 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPBMUX2.bit.GPIO59 = 0; // 0=GPIO,  1=MFSR-A,  2=XD20,  3=Resv
GpioCtrlRegs.GPBDIR.bit.GPIO59 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPBCLEAR.bit.GPIO59 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPBSET.bit.GPIO59 = 1; // uncomment if --> Set High initially
//---------------------------------------------------------------------------
//  GPIO-60 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPBMUX2.bit.GPIO60 = 0; // 0=GPIO,  1=MCLKR-B,  2=XD19,  3=Resv
GpioCtrlRegs.GPBDIR.bit.GPIO60 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPBCLEAR.bit.GPIO60 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPBSET.bit.GPIO60 = 1; // uncomment if --> Set High initially
//---------------------------------------------------------------------------
//  GPIO-61 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPBMUX2.bit.GPIO61 = 0; // 0=GPIO,  1=MFSR-B,  2=XD18,  3=Resv
GpioCtrlRegs.GPBDIR.bit.GPIO61 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPBCLEAR.bit.GPIO61 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPBSET.bit.GPIO61 = 1; // uncomment if --> Set High initially
//---------------------------------------------------------------------------
```

```
//  GPIO-62 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPBMUX2.bit.GPIO62 = 0; // 0=GPIO,  1=SCIRX-C,  2=XD17,  3=Resv
GpioCtrlRegs.GPBDIR.bit.GPIO62 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPBCLEAR.bit.GPIO62 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPBSET.bit.GPIO62 = 1; // uncomment if --> Set High initially
//-----------------------------------------------------------------------------
//  GPIO-63 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPBMUX2.bit.GPIO63 = 0; // 0=GPIO,  1=SCITX-C,  2=XD16,  3=Resv
GpioCtrlRegs.GPBDIR.bit.GPIO63 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPBCLEAR.bit.GPIO63 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPBSET.bit.GPIO63 = 1; // uncomment if --> Set High initially
//-----------------------------------------------------------------------------

//  GPIO-84 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPCMUX2.bit.GPIO84 = 0; // 0=GPIO,  1=GPIO,  2=XA12,  3=Resv
GpioCtrlRegs.GPCDIR.bit.GPIO84 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPCCLEAR.bit.GPIO84 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPCSET.bit.GPIO84 = 1; // uncomment if --> Set High initially
//-----------------------------------------------------------------------------
//  GPIO-85 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPCMUX2.bit.GPIO85 = 0; // 0=GPIO,  1=GPIO,  2=XA13,  3=Resv
GpioCtrlRegs.GPCDIR.bit.GPIO85 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPCCLEAR.bit.GPIO85 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPCSET.bit.GPIO85 = 1; // uncomment if --> Set High initially
//-----------------------------------------------------------------------------
//  GPIO-86 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPCMUX2.bit.GPIO86 = 0; // 0=GPIO,  1=GPIO,  2=XA14,  3=Resv
GpioCtrlRegs.GPCDIR.bit.GPIO86 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPCCLEAR.bit.GPIO86 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPCSET.bit.GPIO86 = 1; // uncomment if --> Set High initially
//-----------------------------------------------------------------------------
//  GPIO-87 - PIN FUNCTION = --Spare--
GpioCtrlRegs.GPCMUX2.bit.GPIO87 = 0; // 0=GPIO,  1=GPIO,  2=XA15,  3=Resv
GpioCtrlRegs.GPCDIR.bit.GPIO87 = 0; // 1=OUTput,  0=INput
// GpioDataRegs.GPCCLEAR.bit.GPIO87 = 1; // uncomment if --> Set Low initially
// GpioDataRegs.GPCSET.bit.GPIO87 = 1; // uncomment if --> Set High initially
//-----------------------------------------------------------------------------

// EDIS; // Disable register access
}

//===========================================================================
// NOTE:
// IN MOST APPLICATIONS THE FUNCTIONS AFTER THIS POINT CAN BE LEFT UNCHANGED
// THE USER NEED NOT REALLY UNDERSTAND THE BELOW CODE TO SUCCESSFULLY RUN THIS
```

```
// APPLICATION.
//===========================================================================

void WDogDisable(void)
{
    EALLOW;
    SysCtrlRegs.WDCR= 0x0068;
//    EDIS;
}

// This function initializes the PLLCR register.
//void InitPll(Uint16 val, Uint16 clkindiv)
void PLLset(Uint16 val)
{
   volatile Uint16 iVol;

   // Make sure the PLL is not running in limp mode
   if (SysCtrlRegs.PLLSTS.bit.MCLKSTS != 0)
   {
      // Missing external clock has been detected
      // Replace this line with a call to an appropriate
      // SystemShutdown(); function.
      asm("        ESTOP0");
   }

   // CLKINDIV MUST be 0 before PLLCR can be changed from
   // 0x0000. It is set to 0 by an external reset XRSn

   // Change the PLLCR
   if (SysCtrlRegs.PLLCR.bit.DIV != val)
   {

      EALLOW;
      // Before setting PLLCR turn off missing clock detect logic
      SysCtrlRegs.PLLSTS.bit.MCLKOFF = 1;
      SysCtrlRegs.PLLCR.bit.DIV = val;
//      EDIS;

      // Optional: Wait for PLL to lock.
      // During this time the CPU will switch to OSCCLK/2 until
      // the PLL is stable.  Once the PLL is stable the CPU will
      // switch to the new PLL value.
      //
      // This time-to-lock is monitored by a PLL lock counter.
      //
```

```
      // Code is not required to sit and wait for the PLL to lock.
      // However, if the code does anything that is timing critical,
      // and requires the correct clock be locked, then it is best to
      // wait until this switching has completed.

      // Wait for the PLL lock bit to be set.
      // The watchdog should be disabled before this loop, or fed within
      // the loop via ServiceDog().

  // Uncomment to disable the watchdog
      WDogDisable();

      EALLOW;
      SysCtrlRegs.PLLSTS.bit.MCLKOFF = 0;
//      SysCtrlRegs.PLLSTS.bit.CLKINDIV != clkindiv;

//      EDIS;
    }

//divide down SysClk by 2 to increase stability
EALLOW;
    SysCtrlRegs.PLLSTS.bit.DIVSEL = 2;
//    EDIS;
}


// This function initializes the PIE control registers to a known state.
//
void PieCntlInit(void)
{
    // Disable Interrupts at the CPU level:
    DINT;

    // Disable the PIE
    PieCtrlRegs.PIECTRL.bit.ENPIE = 0;

// Clear all PIEIER registers:
PieCtrlRegs.PIEIER1.all = 0;
PieCtrlRegs.PIEIER2.all = 0;
PieCtrlRegs.PIEIER3.all = 0;
PieCtrlRegs.PIEIER4.all = 0;
PieCtrlRegs.PIEIER5.all = 0;
PieCtrlRegs.PIEIER6.all = 0;
PieCtrlRegs.PIEIER7.all = 0;
PieCtrlRegs.PIEIER8.all = 0;
```

```
PieCtrlRegs.PIEIER9.all = 0;
PieCtrlRegs.PIEIER10.all = 0;
PieCtrlRegs.PIEIER11.all = 0;
PieCtrlRegs.PIEIER12.all = 0;

// Clear all PIEIFR registers:
PieCtrlRegs.PIEIFR1.all = 0;
PieCtrlRegs.PIEIFR2.all = 0;
PieCtrlRegs.PIEIFR3.all = 0;
PieCtrlRegs.PIEIFR4.all = 0;
PieCtrlRegs.PIEIFR5.all = 0;
PieCtrlRegs.PIEIFR6.all = 0;
PieCtrlRegs.PIEIFR7.all = 0;
PieCtrlRegs.PIEIFR8.all = 0;
PieCtrlRegs.PIEIFR9.all = 0;
PieCtrlRegs.PIEIFR10.all = 0;
PieCtrlRegs.PIEIFR11.all = 0;
PieCtrlRegs.PIEIFR12.all = 0;
}


void PieVectTableInit(void)
{
int16 i;
Uint32 *Source = (void *) &ISR_ILLEGAL;
Uint32 *Dest = (void *) &PieVectTable;

EALLOW;
for(i=0; i < 128; i++)
*Dest++ = *Source;
// EDIS;

// Enable the PIE Vector Table
PieCtrlRegs.PIECTRL.bit.ENPIE = 1;
}

interrupt void ISR_ILLEGAL(void)   // Illegal operation TRAP
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm("          ESTOP0");
  for(;;);
```

```
}

// This function initializes the Flash Control registers

//                   CAUTION
// This function MUST be executed out of RAM. Executing it
// out of OTP/Flash will yield unpredictable results

void InitFlash(void)
{
   EALLOW;
   //Enable Flash Pipeline mode to improve performance
   //of code executed from Flash.
   FlashRegs.FOPT.bit.ENPIPE = 1;

   //                CAUTION
   //Minimum waitstates required for the flash operating
   //at a given CPU rate must be characterized by TI.
   //Refer to the datasheet for the latest information.

   //Set the Paged Waitstate for the Flash
   FlashRegs.FBANKWAIT.bit.PAGEWAIT = 3;

   //Set the Random Waitstate for the Flash
   FlashRegs.FBANKWAIT.bit.RANDWAIT = 3;

   //Set the Waitstate for the OTP
   FlashRegs.FOTPWAIT.bit.OTPWAIT = 5;

   //                CAUTION
   //ONLY THE DEFAULT VALUE FOR THESE 2 REGISTERS SHOULD BE USED
   FlashRegs.FSTDBYWAIT.bit.STDBYWAIT = 0x01FF;
   FlashRegs.FACTIVEWAIT.bit.ACTIVEWAIT = 0x01FF;
//   EDIS;

   //Force a pipeline flush to ensure that the write to
   //the last register configured occurs before returning.

   asm(" RPT #7 || NOP");
}


// This function will copy the specified memory contents from
// one location to another.
//
```

```
// Uint16 *SourceAddr        Pointer to the first word to be moved
//                          SourceAddr < SourceEndAddr
// Uint16* SourceEndAddr     Pointer to the last word to be moved
// Uint16* DestAddr          Pointer to the first destination word
//
// No checks are made for invalid memory locations or that the
// end address is > then the first start address.

void MemCopy(Uint16 *SourceAddr, Uint16* SourceEndAddr, Uint16* DestAddr)
{
    while(SourceAddr < SourceEndAddr)
    {
       *DestAddr++ = *SourceAddr++;
    }
    return;
}

//===========================================================================
// End of file.
//===========================================================================

FlashingLeds-Settings.h
//---------------------------------------------------------------------------
// FILE: FlashingLeds-Settings.h
//
// Description:    This file contains the definitions for this project, and is
// linked to both {ProjectName}-Main.c and {ProjectName}-DPL-ISR.asm .
//
// Type:  Device Independent
//
//---------------------------------------------------------------------------
//  Copyright Texas Instruments  2010
//---------------------------------------------------------------------------
//  Revision History:
//---------------------------------------------------------------------------
//  Date   | Description / Status
//---------------------------------------------------------------------------
// 9 April 2010 - MB
//---------------------------------------------------------------------------
#ifndef _PROJSETTINGS_H
#define _PROJSETTINGS_H

//***************************************************************************
//  NOTE: WHEN CHANGING THIS FILE PLEASE REBUILD ALL
//***************************************************************************
```

```
//===========================================================================
// Incremental Build options for System check-out
//===========================================================================
#define INCR_BUILD 1 //1 - Blink LD3 every 1s
//2 - Blink LD3 every 2s
//3 -
//4 -
//===========================================================================
// System Settings
//---------------------------------------------------------------------------
//Add any system specific setting below


//===========================================================================
// Interrupt Framework options
//===========================================================================
// If using TI's Digital Power Library add interrupt settings here


#endif //_PROJSETTINGS_H

PeripheralHeaderIncludes_F28335.h
//===========================================================================
//===========================================================================
//
// FILE:    PeripheralHeaderIncludes_F28335.h (2833x version)
//
// DESCRIPTION: Contains F283xx device specific definitions and includes
//
// VERSION: 04 Apr 2008 - (BRL)
//===========================================================================
//===========================================================================
#ifndef DSP2833x_DEVICE_H
#define DSP2833x_DEVICE_H 1

#ifdef __cplusplus
extern "C" {
#endif

#define CPU_FRQ_150MHZ 1
//---------------------------------------------------------------------------
// CPU Timer Definitions:
// Timer definitions are based on 150MHz System Clock
//  if not using a 150MHz clock define a different set of constants elsewhere
```

```
#define mSec0_5 75000 // 0.5 mS
#define mSec1 150000 // 1.0 mS
#define mSec2 300000 // 2.0 mS
#define mSec5 750000 // 5.0 mS
#define mSec7_5 1125000 // 7.5 mS
#define mSec10 1500000 // 10 mS
#define mSec20 3000000 // 20 mS
#define mSec50 7500000 // 50 mS
#define mSec100 15000000 // 100 mS
#define mSec500 75000000 // 500 mS
#define mSec1000 150000000 // 1000 mS
#define mSec5000 750000000 // 5000 mS

//---------------------------------------------------------------------------
// Common CPU Definitions:
//
extern cregister volatile unsigned int IFR;
extern cregister volatile unsigned int IER;

#define  EINT   asm(" clrc INTM")
#define  DINT   asm(" setc INTM")
#define  ERTM   asm(" clrc DBGM")
#define  DRTM   asm(" setc DBGM")
#define  EALLOW asm(" EALLOW")
#define  EDIS   asm(" EDIS")
#define  ESTOP0 asm(" ESTOP0")

#define M_INT1  0x0001
#define M_INT2  0x0002
#define M_INT3  0x0004
#define M_INT4  0x0008
#define M_INT5  0x0010
#define M_INT6  0x0020
#define M_INT7  0x0040
#define M_INT8  0x0080
#define M_INT9  0x0100
#define M_INT10 0x0200
#define M_INT11 0x0400
#define M_INT12 0x0800
#define M_INT13 0x1000
#define M_INT14 0x2000
#define M_DLOG  0x4000
#define M_RTOS  0x8000

#define BIT0    0x0001
```

```
#define BIT1    0x0002
#define BIT2    0x0004
#define BIT3    0x0008
#define BIT4    0x0010
#define BIT5    0x0020
#define BIT6    0x0040
#define BIT7    0x0080
#define BIT8    0x0100
#define BIT9    0x0200
#define BIT10   0x0400
#define BIT11   0x0800
#define BIT12   0x1000
#define BIT13   0x2000
#define BIT14   0x4000
#define BIT15   0x8000


//---------------------------------------------------------------------------
// For Portability, User Is Recommended To Use Following Data Type Size
// Definitions For 16-bit and 32-Bit Signed/Unsigned Integers:
//
#ifndef DSP28_DATA_TYPES
#define DSP28_DATA_TYPES
typedef char int8;
typedef int             int16;
typedef long            int32;
typedef unsigned char   Uint8;
typedef unsigned int    Uint16;
typedef unsigned long   Uint32;
typedef float           float32;
typedef long double     float64;
#endif


//---------------------------------------------------------------------------
// Include All Peripheral Header Files:
//
#include "DSP2833x_Adc.h"                // ADC Registers
#include "DSP2833x_DevEmu.h"             // Device Emulation Registers
#include "DSP2833x_CpuTimers.h"          // 32-bit CPU Timers
#include "DSP2833x_ECan.h"               // Enhanced eCAN Registers
#include "DSP2833x_ECap.h"               // Enhanced Capture
#include "DSP2833x_DMA.h"                // DMA Registers
#include "DSP2833x_EPwm.h"               // Enhanced PWM
#include "DSP2833x_EQep.h"               // Enhanced QEP
```

```
#include "DSP2833x_Gpio.h"              // General Purpose I/O Registers
#include "DSP2833x_I2c.h"               // I2C Registers
#include "DSP2833x_McBSP.h"             // McBSP
#include "DSP2833x_PieCtrl.h"           // PIE Control Registers
#include "DSP2833x_PieVect.h"           // PIE Vector Table
#include "DSP2833x_Spi.h"               // SPI Registers
#include "DSP2833x_Sci.h"               // SCI Registers
#include "DSP2833x_SysCtrl.h"           // System Control/Power Modes
#include "DSP2833x_XIntrupt.h"          // External Interrupts
#include "DSP2833x_Xintf.h"             // XINTF External Interface


#ifdef __cplusplus
}
#endif /* extern "C" */

#endif  // end of DSP2833x_DEVICE_H definition

//===========================================================================
// End of file.
//===========================================================================

DSP2833x_PieVect.c
// TI File $Revision: /main/1 $
// Checkin $Date: August 18, 2006   13:46:38 $
//###########################################################################
//
// FILE: DSP2833x_PieVect.c
//
// TITLE: DSP2833x Devices PIE Vector Table Initialization Functions.
//
//###########################################################################
// $TI Release: 2833x/2823x Header Files V1.32 $
// $Release Date: June 28, 2010 $
//###########################################################################

#include "PeripheralHeaderIncludes_F28335.h"
#include "F28335_DefaultIsr.h"

//
//const struct PIE_VECT_TABLE PieVectTableInit = {
//
//      PIE_RESERVED,  // 0  Reserved space
//      PIE_RESERVED,  // 1  Reserved space
//      PIE_RESERVED,  // 2  Reserved space
```

```
//      PIE_RESERVED,  // 3  Reserved space
//      PIE_RESERVED,  // 4  Reserved space
//      PIE_RESERVED,  // 5  Reserved space
//      PIE_RESERVED,  // 6  Reserved space
//      PIE_RESERVED,  // 7  Reserved space
//      PIE_RESERVED,  // 8  Reserved space
//      PIE_RESERVED,  // 9  Reserved space
//      PIE_RESERVED,  // 10 Reserved space
//      PIE_RESERVED,  // 11 Reserved space
//      PIE_RESERVED,  // 12 Reserved space
//
//
//// Non-Peripheral Interrupts
//      INT13_ISR,     // XINT13 or CPU-Timer 1
//      INT14_ISR,     // CPU-Timer2
//      DATALOG_ISR,   // Datalogging interrupt
//      RTOSINT_ISR,   // RTOS interrupt
//      EMUINT_ISR,    // Emulation interrupt
//      NMI_ISR,       // Non-maskable interrupt
//      ILLEGAL_ISR,   // Illegal operation TRAP
//      USER1_ISR,     // User Defined trap 1
//      USER2_ISR,     // User Defined trap 2
//      USER3_ISR,     // User Defined trap 3
//      USER4_ISR,     // User Defined trap 4
//      USER5_ISR,     // User Defined trap 5
//      USER6_ISR,     // User Defined trap 6
//      USER7_ISR,     // User Defined trap 7
//      USER8_ISR,     // User Defined trap 8
//      USER9_ISR,     // User Defined trap 9
//      USER10_ISR,    // User Defined trap 10
//      USER11_ISR,    // User Defined trap 11
//      USER12_ISR,    // User Defined trap 12
//
//// Group 1 PIE Vectors
//      SEQ1INT_ISR,     // 1.1 ADC
//      SEQ2INT_ISR,     // 1.2 ADC
//      rsvd_ISR,        // 1.3
//      XINT1_ISR,       // 1.4
//      XINT2_ISR,       // 1.5
//      ADCINT_ISR,      // 1.6 ADC
//      TINT0_ISR,       // 1.7 Timer 0
//      WAKEINT_ISR,     // 1.8 WD, Low Power
//
//// Group 2 PIE Vectors
//      EPWM1_TZINT_ISR, // 2.1 EPWM-1 Trip Zone
```

```
//      EPWM2_TZINT_ISR, // 2.2 EPWM-2 Trip Zone
//      EPWM3_TZINT_ISR, // 2.3 EPWM-3 Trip Zone
//      EPWM4_TZINT_ISR, // 2.4 EPWM-4 Trip Zone
//      EPWM5_TZINT_ISR, // 2.5 EPWM-5 Trip Zone
//      EPWM6_TZINT_ISR, // 2.6 EPWM-6 Trip Zone
//      rsvd_ISR,        // 2.7
//      rsvd_ISR,        // 2.8
//
//// Group 3 PIE Vectors
//      EPWM1_INT_ISR,   // 3.1 EPWM-1 Interrupt
//      EPWM2_INT_ISR,   // 3.2 EPWM-2 Interrupt
//      EPWM3_INT_ISR,   // 3.3 EPWM-3 Interrupt
//      EPWM4_INT_ISR,   // 3.4 EPWM-4 Interrupt
//      EPWM5_INT_ISR,   // 3.5 EPWM-5 Interrupt
//      EPWM6_INT_ISR,   // 3.6 EPWM-6 Interrupt
//      rsvd_ISR,        // 3.7
//      rsvd_ISR,        // 3.8
//
//// Group 4 PIE Vectors
//      ECAP1_INT_ISR,   // 4.1 ECAP-1
//      ECAP2_INT_ISR,   // 4.2 ECAP-2
//      ECAP3_INT_ISR,   // 4.3 ECAP-3
//      ECAP4_INT_ISR,   // 4.4 ECAP-4
//      ECAP5_INT_ISR,   // 4.5 ECAP-5
//      ECAP6_INT_ISR,   // 4.6 ECAP-6
//      rsvd_ISR,        // 4.7
//      rsvd_ISR,        // 4.8
//
//// Group 5 PIE Vectors
//      EQEP1_INT_ISR,   // 5.1 EQEP-1
//      EQEP2_INT_ISR,   // 5.2 EQEP-2
//      rsvd_ISR,        // 5.3
//      rsvd_ISR,        // 5.4
//      rsvd_ISR,        // 5.5
//      rsvd_ISR,        // 5.6
//      rsvd_ISR,        // 5.7
//      rsvd_ISR,        // 5.8
//
//
//// Group 6 PIE Vectors
//      SPIRXINTA_ISR,   // 6.1 SPI-A
//      SPITXINTA_ISR,   // 6.2 SPI-A
//      MRINTA_ISR,      // 6.3 McBSP-A
//      MXINTA_ISR,      // 6.4 McBSP-A
//      MRINTB_ISR,      // 6.5 McBSP-B
```

```
//      MXINTB_ISR,      // 6.6 McBSP-B
//      rsvd_ISR,        // 6.7
//      rsvd_ISR,        // 6.8
//
//
//// Group 7 PIE Vectors
//      DINTCH1_ISR,     // 7.1  DMA channel 1
//      DINTCH2_ISR,     // 7.2  DMA channel 2
//      DINTCH3_ISR,     // 7.3  DMA channel 3
//      DINTCH4_ISR,     // 7.4  DMA channel 4
//      DINTCH5_ISR,     // 7.5  DMA channel 5
//      DINTCH6_ISR,     // 7.6  DMA channel 6
//      rsvd_ISR,        // 7.7
//      rsvd_ISR,        // 7.8
//
//// Group 8 PIE Vectors
//      I2CINT1A_ISR,    // 8.1  I2C
//      I2CINT2A_ISR,    // 8.2  I2C
//      rsvd_ISR,        // 8.3
//      rsvd_ISR,        // 8.4
//      SCIRXINTC_ISR,   // 8.5  SCI-C
//      SCITXINTC_ISR,   // 8.6  SCI-C
//      rsvd_ISR,        // 8.7
//      rsvd_ISR,        // 8.8
//
//// Group 9 PIE Vectors
//      SCIRXINTA_ISR,   // 9.1 SCI-A
//      SCITXINTA_ISR,   // 9.2 SCI-A
//      SCIRXINTB_ISR,   // 9.3 SCI-B
//      SCITXINTB_ISR,   // 9.4 SCI-B
//      ECAN0INTA_ISR,   // 9.5 eCAN-A
//      ECAN1INTA_ISR,   // 9.6 eCAN-A
//      ECAN0INTB_ISR,   // 9.7 eCAN-B
//      ECAN1INTB_ISR,   // 9.8 eCAN-B
//
//// Group 10 PIE Vectors
//      rsvd_ISR,        // 10.1
//      rsvd_ISR,        // 10.2
//      rsvd_ISR,        // 10.3
//      rsvd_ISR,        // 10.4
//      rsvd_ISR,        // 10.5
//      rsvd_ISR,        // 10.6
//      rsvd_ISR,        // 10.7
//      rsvd_ISR,        // 10.8
//
```

```
//// Group 11 PIE Vectors
//      rsvd_ISR,        // 11.1
//      rsvd_ISR,        // 11.2
//      rsvd_ISR,        // 11.3
//      rsvd_ISR,        // 11.4
//      rsvd_ISR,        // 11.5
//      rsvd_ISR,        // 11.6
//      rsvd_ISR,        // 11.7
//      rsvd_ISR,        // 11.8
//
//// Group 12 PIE Vectors
//      XINT3_ISR,       // 12.1
//      XINT4_ISR,       // 12.2
//      XINT5_ISR,       // 12.3
//      XINT6_ISR,       // 12.4
//      XINT7_ISR,       // 12.5
//      rsvd_ISR,        // 12.6
//      LVF_ISR,         // 12.7
//      LUF_ISR,         // 12.8
//};


//---------------------------------------------------------------------------
// InitPieVectTable:
//---------------------------------------------------------------------------
// This function initializes the PIE vector table to a known state.
// This function must be executed after boot time.
//

//void InitPieVectTable(void)
//{
// int16 i;
// Uint32 *Source = (void *) &PieVectTableInit;
// Uint32 *Dest = (void *) &PieVectTable;
//
// EALLOW;
// for(i=0; i < 128; i++)
// *Dest++ = *Source++;
// EDIS;
//
// // Enable the PIE Vector Table
// PieCtrlRegs.PIECTRL.bit.ENPIE = 1;
//
//}
```

```
//===========================================================================
// End of file.
//===========================================================================

DSP2833x_GlobalVariableDefs.c
// TI File $Revision: /main/4 $
// Checkin $Date: June 2, 2008   11:12:33 $
//###########################################################################
//
// FILE: DSP2833x_GlobalVariableDefs.c
//
// TITLE: DSP2833x Global Variables and Data Section Pragmas.
//
//###########################################################################
// $TI Release: 2833x/2823x Header Files V1.32 $
// $Release Date: June 28, 2010 $
//###########################################################################

#include "DSP2833x_Device.h"     // DSP2833x Headerfile Include File

//---------------------------------------------------------------------------
// Define Global Peripheral Variables:
//
//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("AdcRegsFile")
#else
#pragma DATA_SECTION(AdcRegs,"AdcRegsFile");
#endif
volatile struct ADC_REGS AdcRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("AdcMirrorFile")
#else
#pragma DATA_SECTION(AdcMirror,"AdcMirrorFile");
#endif
volatile struct ADC_RESULT_MIRROR_REGS AdcMirror;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("CpuTimer0RegsFile")
#else
#pragma DATA_SECTION(CpuTimer0Regs,"CpuTimer0RegsFile");
#endif
```

```
volatile struct CPUTIMER_REGS CpuTimer0Regs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("CpuTimer1RegsFile")
#else
#pragma DATA_SECTION(CpuTimer1Regs,"CpuTimer1RegsFile");
#endif
volatile struct CPUTIMER_REGS CpuTimer1Regs;


//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("CpuTimer2RegsFile")
#else
#pragma DATA_SECTION(CpuTimer2Regs,"CpuTimer2RegsFile");
#endif
volatile struct CPUTIMER_REGS CpuTimer2Regs;


//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("CsmPwlFile")
#else
#pragma DATA_SECTION(CsmPwl,"CsmPwlFile");
#endif
volatile struct CSM_PWL CsmPwl;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("CsmRegsFile")
#else
#pragma DATA_SECTION(CsmRegs,"CsmRegsFile");
#endif
volatile struct CSM_REGS CsmRegs;



//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("DevEmuRegsFile")
#else
#pragma DATA_SECTION(DevEmuRegs,"DevEmuRegsFile");
#endif
volatile struct DEV_EMU_REGS DevEmuRegs;
```

```
//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("DmaRegsFile")
#else
#pragma DATA_SECTION(DmaRegs,"DmaRegsFile");
#endif
volatile struct DMA_REGS DmaRegs;


//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ECanaRegsFile")
#else
#pragma DATA_SECTION(ECanaRegs,"ECanaRegsFile");
#endif
volatile struct ECAN_REGS ECanaRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ECanaMboxesFile")
#else
#pragma DATA_SECTION(ECanaMboxes,"ECanaMboxesFile");
#endif
volatile struct ECAN_MBOXES ECanaMboxes;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ECanaLAMRegsFile")
#else
#pragma DATA_SECTION(ECanaLAMRegs,"ECanaLAMRegsFile");
#endif
volatile struct LAM_REGS ECanaLAMRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ECanaMOTSRegsFile")
#else
#pragma DATA_SECTION(ECanaMOTSRegs,"ECanaMOTSRegsFile");
#endif
volatile struct MOTS_REGS ECanaMOTSRegs;


//---------------------------------------
#ifdef __cplusplus
```

```
#pragma DATA_SECTION("ECanaMOTORegsFile")
#else
#pragma DATA_SECTION(ECanaMOTORegs,"ECanaMOTORegsFile");
#endif
volatile struct MOTO_REGS ECanaMOTORegs;


//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ECanbRegsFile")
#else
#pragma DATA_SECTION(ECanbRegs,"ECanbRegsFile");
#endif
volatile struct ECAN_REGS ECanbRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ECanbMboxesFile")
#else
#pragma DATA_SECTION(ECanbMboxes,"ECanbMboxesFile");
#endif
volatile struct ECAN_MBOXES ECanbMboxes;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ECanbLAMRegsFile")
#else
#pragma DATA_SECTION(ECanbLAMRegs,"ECanbLAMRegsFile");
#endif
volatile struct LAM_REGS ECanbLAMRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ECanbMOTSRegsFile")
#else
#pragma DATA_SECTION(ECanbMOTSRegs,"ECanbMOTSRegsFile");
#endif
volatile struct MOTS_REGS ECanbMOTSRegs;


//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ECanbMOTORegsFile")
#else
#pragma DATA_SECTION(ECanbMOTORegs,"ECanbMOTORegsFile");
```

```
#endif
volatile struct MOTO_REGS ECanbMOTORegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("EPwm1RegsFile")
#else
#pragma DATA_SECTION(EPwm1Regs,"EPwm1RegsFile");
#endif
volatile struct EPWM_REGS EPwm1Regs;


//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("EPwm2RegsFile")
#else
#pragma DATA_SECTION(EPwm2Regs,"EPwm2RegsFile");
#endif
volatile struct EPWM_REGS EPwm2Regs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("EPwm3RegsFile")
#else
#pragma DATA_SECTION(EPwm3Regs,"EPwm3RegsFile");
#endif
volatile struct EPWM_REGS EPwm3Regs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("EPwm4RegsFile")
#else
#pragma DATA_SECTION(EPwm4Regs,"EPwm4RegsFile");
#endif
volatile struct EPWM_REGS EPwm4Regs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("EPwm5RegsFile")
#else
#pragma DATA_SECTION(EPwm5Regs,"EPwm5RegsFile");
#endif
volatile struct EPWM_REGS EPwm5Regs;

//---------------------------------------
```

```
#ifdef __cplusplus
#pragma DATA_SECTION("EPwm6RegsFile")
#else
#pragma DATA_SECTION(EPwm6Regs,"EPwm6RegsFile");
#endif
volatile struct EPWM_REGS EPwm6Regs;


//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ECap1RegsFile")
#else
#pragma DATA_SECTION(ECap1Regs,"ECap1RegsFile");
#endif
volatile struct ECAP_REGS ECap1Regs;


//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ECap2RegsFile")
#else
#pragma DATA_SECTION(ECap2Regs,"ECap2RegsFile");
#endif
volatile struct ECAP_REGS ECap2Regs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ECap3RegsFile")
#else
#pragma DATA_SECTION(ECap3Regs,"ECap3RegsFile");
#endif
volatile struct ECAP_REGS ECap3Regs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ECap4RegsFile")
#else
#pragma DATA_SECTION(ECap4Regs,"ECap4RegsFile");
#endif
volatile struct ECAP_REGS ECap4Regs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ECap5RegsFile")
#else
```

```
#pragma DATA_SECTION(ECap5Regs,"ECap5RegsFile");
#endif
volatile struct ECAP_REGS ECap5Regs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ECap6RegsFile")
#else
#pragma DATA_SECTION(ECap6Regs,"ECap6RegsFile");
#endif
volatile struct ECAP_REGS ECap6Regs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("EQep1RegsFile")
#else
#pragma DATA_SECTION(EQep1Regs,"EQep1RegsFile");
#endif
volatile struct EQEP_REGS EQep1Regs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("EQep2RegsFile")
#else
#pragma DATA_SECTION(EQep2Regs,"EQep2RegsFile");
#endif
volatile struct EQEP_REGS EQep2Regs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("GpioCtrlRegsFile")
#else
#pragma DATA_SECTION(GpioCtrlRegs,"GpioCtrlRegsFile");
#endif
volatile struct GPIO_CTRL_REGS GpioCtrlRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("GpioDataRegsFile")
#else
#pragma DATA_SECTION(GpioDataRegs,"GpioDataRegsFile");
#endif
volatile struct GPIO_DATA_REGS GpioDataRegs;

//---------------------------------------
```

```
#ifdef __cplusplus
#pragma DATA_SECTION("GpioIntRegsFile")
#else
#pragma DATA_SECTION(GpioIntRegs,"GpioIntRegsFile");
#endif
volatile struct GPIO_INT_REGS GpioIntRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("I2caRegsFile")
#else
#pragma DATA_SECTION(I2caRegs,"I2caRegsFile");
#endif
volatile struct I2C_REGS I2caRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("McbspaRegsFile")
#else
#pragma DATA_SECTION(McbspaRegs,"McbspaRegsFile");
#endif
volatile struct MCBSP_REGS McbspaRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("McbspbRegsFile")
#else
#pragma DATA_SECTION(McbspbRegs,"McbspbRegsFile");
#endif
volatile struct MCBSP_REGS McbspbRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("PartIdRegsFile")
#else
#pragma DATA_SECTION(PartIdRegs,"PartIdRegsFile");
#endif
volatile struct PARTID_REGS PartIdRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("PieCtrlRegsFile")
#else
#pragma DATA_SECTION(PieCtrlRegs,"PieCtrlRegsFile");
#endif
```

```
volatile struct PIE_CTRL_REGS PieCtrlRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("PieVectTableFile")
#else
#pragma DATA_SECTION(PieVectTable,"PieVectTableFile");
#endif
struct PIE_VECT_TABLE PieVectTable;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("SciaRegsFile")
#else
#pragma DATA_SECTION(SciaRegs,"SciaRegsFile");
#endif
volatile struct SCI_REGS SciaRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ScibRegsFile")
#else
#pragma DATA_SECTION(ScibRegs,"ScibRegsFile");
#endif
volatile struct SCI_REGS ScibRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("ScicRegsFile")
#else
#pragma DATA_SECTION(ScicRegs,"ScicRegsFile");
#endif
volatile struct SCI_REGS ScicRegs;


//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("SpiaRegsFile")
#else
#pragma DATA_SECTION(SpiaRegs,"SpiaRegsFile");
#endif
volatile struct SPI_REGS SpiaRegs;

//---------------------------------------
#ifdef __cplusplus
```

```
#pragma DATA_SECTION("SysCtrlRegsFile")
#else
#pragma DATA_SECTION(SysCtrlRegs,"SysCtrlRegsFile");
#endif
volatile struct SYS_CTRL_REGS SysCtrlRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("FlashRegsFile")
#else
#pragma DATA_SECTION(FlashRegs,"FlashRegsFile");
#endif
volatile struct FLASH_REGS FlashRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("XIntruptRegsFile")
#else
#pragma DATA_SECTION(XIntruptRegs,"XIntruptRegsFile");
#endif
volatile struct XINTRUPT_REGS XIntruptRegs;

//---------------------------------------
#ifdef __cplusplus
#pragma DATA_SECTION("XintfRegsFile")
#else
#pragma DATA_SECTION(XintfRegs,"XintfRegsFile");
#endif
volatile struct XINTF_REGS XintfRegs;



//===========================================================================
// End of file.
//===========================================================================

F28335_EPwm_Int.h
#ifndef F28335_EPWM_INIT_H_
#define F28335_EPWM_INIT_H_

void InitEPwm1Gpio(void);
void InitEPwm2Gpio(void);
void InitEPwm3Gpio(void);
void InitEPwm4Gpio(void);
void InitEPwm5Gpio(void);
```

```
void InitEPwm6Gpio(void);

void InitEPwmGpio(void);

#endif /*F28335_EPWM_INIT_H_*/
```

F28335_EPwm_Int.c

```
#include "PeripheralHeaderIncludes_F28335.h"

void InitEPwm(void)
{
   // Initialize ePWM1/2/3/4/5/6

   //tbd...

}

//---------------------------------------------------------------------------
// Example: InitEPwmGpio:
//---------------------------------------------------------------------------
// This function initializes GPIO pins to function as ePWM pins
//
// Each GPIO pin can be configured as a GPIO pin or up to 3 different
// peripheral functional pins. By default all pins come up as GPIO
// inputs after reset.
//


void InitEPwm1Gpio(void)
{
   EALLOW;

/* Enable internal pull-up for the selected pins */
// Pull-ups can be enabled or disabled by the user.
// This will enable the pullups for the specified pins.
// Comment out other unwanted lines.

    GpioCtrlRegs.GPAPUD.bit.GPIO0 = 0;    // Enable pull-up on GPIO0 (EPWM1A)
    GpioCtrlRegs.GPAPUD.bit.GPIO1 = 0;    // Enable pull-up on GPIO1 (EPWM1B)

/* Configure ePWM-1 pins using GPIO regs*/
// This specifies which of the possible GPIO pins will be ePWM1 functional pins.
// Comment out other unwanted lines.
```

```
    GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 1;   // Configure GPIO0 as EPWM1A
    GpioCtrlRegs.GPAMUX1.bit.GPIO1 = 1;   // Configure GPIO1 as EPWM1B

    EDIS;
}

void InitEPwm2Gpio(void)
{
   EALLOW;

/* Enable internal pull-up for the selected pins */
// Pull-ups can be enabled or disabled by the user.
// This will enable the pullups for the specified pins.
// Comment out other unwanted lines.

    GpioCtrlRegs.GPAPUD.bit.GPIO2 = 0;    // Enable pull-up on GPIO2 (EPWM2A)
    GpioCtrlRegs.GPAPUD.bit.GPIO3 = 0;    // Enable pull-up on GPIO3 (EPWM3B)

/* Configure ePWM-2 pins using GPIO regs*/
// This specifies which of the possible GPIO pins will be ePWM2 functional pins.
// Comment out other unwanted lines.

    GpioCtrlRegs.GPAMUX1.bit.GPIO2 = 1;   // Configure GPIO2 as EPWM2A
    GpioCtrlRegs.GPAMUX1.bit.GPIO3 = 1;   // Configure GPIO3 as EPWM2B

    EDIS;
}

void InitEPwm3Gpio(void)
{
   EALLOW;

/* Enable internal pull-up for the selected pins */
// Pull-ups can be enabled or disabled by the user.
// This will enable the pullups for the specified pins.
// Comment out other unwanted lines.

    GpioCtrlRegs.GPAPUD.bit.GPIO4 = 0;    // Enable pull-up on GPIO4 (EPWM3A)
    GpioCtrlRegs.GPAPUD.bit.GPIO5 = 0;    // Enable pull-up on GPIO5 (EPWM3B)

/* Configure ePWM-3 pins using GPIO regs*/
// This specifies which of the possible GPIO pins will be ePWM3 functional pins.
// Comment out other unwanted lines.

    GpioCtrlRegs.GPAMUX1.bit.GPIO4 = 1;   // Configure GPIO4 as EPWM3A
```

```
    GpioCtrlRegs.GPAMUX1.bit.GPIO5 = 1;   // Configure GPIO5 as EPWM3B

    EDIS;
}

void InitEPwm4Gpio(void)
{
   EALLOW;
/* Enable internal pull-up for the selected pins */
// Pull-ups can be enabled or disabled by the user.
// This will enable the pullups for the specified pins.
// Comment out other unwanted lines.

    GpioCtrlRegs.GPAPUD.bit.GPIO6 = 0;    // Enable pull-up on GPIO6 (EPWM4A)
    GpioCtrlRegs.GPAPUD.bit.GPIO7 = 0;    // Enable pull-up on GPIO7 (EPWM4B)

/* Configure ePWM-4 pins using GPIO regs*/
// This specifies which of the possible GPIO pins will be ePWM4 functional pins.
// Comment out other unwanted lines.

    GpioCtrlRegs.GPAMUX1.bit.GPIO6 = 1;   // Configure GPIO6 as EPWM4A
    GpioCtrlRegs.GPAMUX1.bit.GPIO7 = 1;   // Configure GPIO7 as EPWM4B

    EDIS;
}

void InitEPwm5Gpio(void)
{
   EALLOW;
/* Enable internal pull-up for the selected pins */
// Pull-ups can be enabled or disabled by the user.
// This will enable the pullups for the specified pins.
// Comment out other unwanted lines.

    GpioCtrlRegs.GPAPUD.bit.GPIO8 = 0;    // Enable pull-up on GPIO8 (EPWM5A)
    GpioCtrlRegs.GPAPUD.bit.GPIO9 = 0;    // Enable pull-up on GPIO9 (EPWM5B)

/* Configure ePWM-5 pins using GPIO regs*/
// This specifies which of the possible GPIO pins will be ePWM5 functional pins.
// Comment out other unwanted lines.

    GpioCtrlRegs.GPAMUX1.bit.GPIO8 = 1;   // Configure GPIO8 as EPWM5A
    GpioCtrlRegs.GPAMUX1.bit.GPIO9 = 1;   // Configure GPIO9 as EPWM5B

    EDIS;
```

```
}

void InitEPwm6Gpio(void)
{
   EALLOW;

/* Enable internal pull-up for the selected pins */
// Pull-ups can be enabled or disabled by the user.
// This will enable the pullups for the specified pins.
// Comment out other unwanted lines.

    GpioCtrlRegs.GPAPUD.bit.GPIO10 = 0;    // Enable pull-up on GPIO10 (EPWM6A)
    GpioCtrlRegs.GPAPUD.bit.GPIO11 = 0;    // Enable pull-up on GPIO11 (EPWM6B)

/* Configure ePWM-6 pins using GPIO regs*/
// This specifies which of the possible GPIO pins will be ePWM6 functional pins.
// Comment out other unwanted lines.

    GpioCtrlRegs.GPAMUX1.bit.GPIO10 = 1;   // Configure GPIO10 as EPWM6A
    GpioCtrlRegs.GPAMUX1.bit.GPIO11 = 1;   // Configure GPIO11 as EPWM6B

    EDIS;
}


void InitEPwmGpio(void)
{
   InitEPwm1Gpio();
   InitEPwm2Gpio();
   InitEPwm3Gpio();
   InitEPwm4Gpio();
   InitEPwm5Gpio();
   InitEPwm6Gpio();
}

F283335_DefaultIsr.h
#ifndef F28335_DEFAULTISR_H_
#define F28335_DEFAULTISR_H_

// Connected to INT13 of CPU (use MINT13 mask):
// Note CPU-Timer1 is reserved for TI use, however XINT13
// ISR can be used by the user.
interrupt void INT13_ISR(void);     // INT13 or CPU-Timer1

// Note CPU-Timer2 is reserved for TI use.
```

```
interrupt void INT14_ISR(void);     // CPU-Timer2

interrupt void DATALOG_ISR(void);   // Datalogging interrupt

interrupt void RTOSINT_ISR(void);   // RTOS interrupt

interrupt void EMUINT_ISR(void);    // Emulation interrupt

interrupt void NMI_ISR(void);       // Non-maskable interrupt

interrupt void ILLEGAL_ISR(void);   // Illegal operation TRAP

interrupt void USER1_ISR(void);     // User Defined trap 1

interrupt void USER2_ISR(void);     // User Defined trap 2

interrupt void USER3_ISR(void);     // User Defined trap 3

interrupt void USER4_ISR(void);     // User Defined trap 4

interrupt void USER5_ISR(void);     // User Defined trap 5

interrupt void USER6_ISR(void);     // User Defined trap 6

interrupt void USER7_ISR(void);     // User Defined trap 7

interrupt void USER8_ISR(void);     // User Defined trap 8

interrupt void USER9_ISR(void);     // User Defined trap 9

interrupt void USER10_ISR(void);    // User Defined trap 10

interrupt void USER11_ISR(void);    // User Defined trap 11

interrupt void USER12_ISR(void);     // User Defined trap 12

// ---------------------------------------------------------
// PIE Group 1 - MUXed into CPU INT1
// ---------------------------------------------------------

// INT1.1
interrupt void SEQ1INT_ISR(void);   //SEQ1 ADC

// INT1.2
interrupt void SEQ2INT_ISR(void);  //SEQ2 ADC
```

```
// INT1.3 - Reserved

// INT1.4
interrupt void  XINT1_ISR(void);

// INT1.5
interrupt void  XINT2_ISR(void);

// INT1.6
interrupt void  ADCINT_ISR(void);     // ADC

// INT1.7
interrupt void  TINT0_ISR(void);      // CPU-Timer 0

// INT1.8
interrupt void  WAKEINT_ISR(void);    // WD, LOW Power

// ------------------------------------------------------------
// PIE Group 2 - MUXed into CPU INT2
// ------------------------------------------------------------

// INT2.1
interrupt void EPWM1_TZINT_ISR(void);    // EPWM-1

// INT2.2
interrupt void EPWM2_TZINT_ISR(void);    // EPWM-2

// INT2.3
interrupt void EPWM3_TZINT_ISR(void);    // EPWM-3

// INT2.4
interrupt void EPWM4_TZINT_ISR(void);    // EPWM-4

// INT2.5
interrupt void EPWM5_TZINT_ISR(void);    // EPWM-5

// INT2.6
interrupt void EPWM6_TZINT_ISR(void);   // EPWM-6

// INT2.7 - Reserved
// INT2.8 - Reserved

// ------------------------------------------------------------
// PIE Group 3 - MUXed into CPU INT3
// ------------------------------------------------------------
```

```
// INT 3.1
interrupt void EPWM1_INT_ISR(void);     // EPWM-1

// INT3.2
interrupt void EPWM2_INT_ISR(void);     // EPWM-2

// INT3.3
interrupt void EPWM3_INT_ISR(void);    // EPWM-3

// INT3.4
interrupt void EPWM4_INT_ISR(void);    // EPWM-4

// INT3.5
interrupt void EPWM5_INT_ISR(void);    // EPWM-5

// INT3.6
interrupt void EPWM6_INT_ISR(void);    // EPWM-6

// INT3.7 - Reserved
// INT3.8 - Reserved


// ------------------------------------------------------------
// PIE Group 4 - MUXed into CPU INT4
// ------------------------------------------------------------

// INT 4.1
interrupt void ECAP1_INT_ISR(void);    // ECAP-1

// INT4.2
interrupt void ECAP2_INT_ISR(void);    // ECAP-2

// INT4.3
interrupt void ECAP3_INT_ISR(void);    // ECAP-3

// INT4.4
interrupt void ECAP4_INT_ISR(void);     // ECAP-4

// INT4.5
interrupt void ECAP5_INT_ISR(void);     // ECAP-5

// INT4.6
interrupt void ECAP6_INT_ISR(void);     // ECAP-6
// INT4.7 - Reserved
```

```
// INT4.8 - Reserved

// ---------------------------------------------------------
// PIE Group 5 - MUXed into CPU INT5
// ---------------------------------------------------------

// INT 5.1
interrupt void EQEP1_INT_ISR(void);    // EQEP-1

// INT5.2
interrupt void EQEP2_INT_ISR(void);    // EQEP-2

// INT5.3 - Reserved
// INT5.4 - Reserved
// INT5.5 - Reserved
// INT5.6 - Reserved
// INT5.7 - Reserved
// INT5.8 - Reserved

// ---------------------------------------------------------
// PIE Group 6 - MUXed into CPU INT6
// ---------------------------------------------------------

// INT6.1
interrupt void SPIRXINTA_ISR(void);    // SPI-A

// INT6.2
interrupt void SPITXINTA_ISR(void);     // SPI-A

// INT6.3
interrupt void MRINTB_ISR(void);     // McBSP-B

// INT6.4
interrupt void MXINTB_ISR(void);    // McBSP-B

// INT6.5
interrupt void MRINTA_ISR(void);    // McBSP-A

// INT6.6
interrupt void MXINTA_ISR(void);    // McBSP-A

// INT6.7 - Reserved
// INT6.8 - Reserved
```

```
// ---------------------------------------------------------
// PIE Group 7 - MUXed into CPU INT7
// ---------------------------------------------------------

// INT7.1
interrupt void DINTCH1_ISR(void);    // DMA

// INT7.2
interrupt void DINTCH2_ISR(void);    // DMA

// INT7.3
interrupt void DINTCH3_ISR(void);    // DMA

// INT7.4
interrupt void DINTCH4_ISR(void);    // DMA

// INT7.5
interrupt void DINTCH5_ISR(void);    // DMA

// INT7.6
interrupt void DINTCH6_ISR(void);    // DMA

// INT7.7 - Reserved
// INT7.8 - Reserved

// ---------------------------------------------------------
// PIE Group 8 - MUXed into CPU INT8
// ---------------------------------------------------------

// INT8.1
interrupt void I2CINT1A_ISR(void);     // I2C-A

// INT8.2
interrupt void I2CINT2A_ISR(void);     // I2C-A

// INT8.3 - Reserved
// INT8.4 - Reserved

// INT8.5
interrupt void SCIRXINTC_ISR(void);     // SCI-C

// INT8.6
interrupt void SCITXINTC_ISR(void);     // SCI-C
```

```
// INT8.7 - Reserved
// INT8.8 - Reserved


// ---------------------------------------------------------
// PIE Group 9 - MUXed into CPU INT9
// ---------------------------------------------------------

// INT9.1
interrupt void SCIRXINTA_ISR(void);     // SCI-A

// INT9.2
interrupt void SCITXINTA_ISR(void);     // SCI-A

// INT9.3
interrupt void SCIRXINTB_ISR(void);     // SCI-B

// INT9.4
interrupt void SCITXINTB_ISR(void);     // SCI-B

// INT9.5
interrupt void ECAN0INTA_ISR(void);  // eCAN-A

// INT9.6
interrupt void ECAN1INTA_ISR(void);  // eCAN-A

// INT9.7
interrupt void ECAN0INTB_ISR(void);  // eCAN-B

// INT9.8
interrupt void ECAN1INTB_ISR(void);  // eCAN-B

// ---------------------------------------------------------
// PIE Group 10 - MUXed into CPU INT10
// ---------------------------------------------------------

// INT10.1 - Reserved
// INT10.2 - Reserved
// INT10.3 - Reserved
// INT10.4 - Reserved
// INT10.5 - Reserved
// INT10.6 - Reserved
// INT10.7 - Reserved
// INT10.8 - Reserved
```

```
// ---------------------------------------------------------
// PIE Group 11 - MUXed into CPU INT11
// ---------------------------------------------------------

// INT11.1 - Reserved
// INT11.2 - Reserved
// INT11.3 - Reserved
// INT11.4 - Reserved
// INT11.5 - Reserved
// INT11.6 - Reserved
// INT11.7 - Reserved
// INT11.8 - Reserved

// ---------------------------------------------------------
// PIE Group 12 - MUXed into CPU INT12
// ---------------------------------------------------------

// INT12.1
interrupt void XINT3_ISR(void);  // External Interrupt

// INT12.2
interrupt void XINT4_ISR(void);  // External Interrupt

// INT12.3
interrupt void XINT5_ISR(void);  // External Interrupt

// INT12.4
interrupt void XINT6_ISR(void);  // External Interrupt

// INT12.5
interrupt void XINT7_ISR(void);  // External Interrupt

// INT12.6 - Reserved
// INT12.7
interrupt void LVF_ISR(void);  // Latched overflow
// INT12.8
interrupt void LUF_ISR(void);  // Latched underflow


interrupt void PIE_RESERVED(void);  // Reserved space.  For test.
interrupt void rsvd_ISR(void);      // For test

#endif /*F28335_DEFAULTISR_H_*/
```

```
F28335_DefaultIsr.c
// TI File $Revision: /main/2 $
// Checkin $Date: January 14, 2008   11:17:46 $
//###########################################################################
//
// FILE: DSP2833x_DefaultIsr.c
//
// TITLE: DSP2833x Device Default Interrupt Service Routines.
//
// This file contains shell ISR routines for the 2833x PIE vector table.
// Typically these shell ISR routines can be used to populate the entire PIE
// vector table during device debug.  In this manner if an interrupt is taken
// during firmware development, there will always be an ISR to catch it.
//
// As develpment progresses, these ISR rotuines can be eliminated and replaced
// with the user's own ISR routines for each interrupt.  Since these shell ISRs
// include infinite loops they will typically not be included as-is in the final
// production firmware.
//
//###########################################################################
// $TI Release: 2833x/2823x Header Files V1.32 $
// $Release Date: June 28, 2010 $
//###########################################################################

#include "PeripheralHeaderIncludes_F28335.h"


// Connected to INT13 of CPU (use MINT13 mask):
// Note CPU-Timer1 is reserved for TI use, however XINT13
// ISR can be used by the user.
interrupt void INT13_ISR(void)     // INT13 or CPU-Timer1
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// Note CPU-Timer2 is reserved for TI use.
interrupt void INT14_ISR(void)     // CPU-Timer2
{
  // Insert ISR Code here
```

```
  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

interrupt void DATALOG_ISR(void)   // Datalogging interrupt
{
   // Insert ISR Code here

   // Next two lines for debug only to halt the processor here
   // Remove after inserting ISR Code
   asm ("      ESTOP0");
   for(;;);
}

interrupt void RTOSINT_ISR(void)   // RTOS interrupt
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

interrupt void EMUINT_ISR(void)    // Emulation interrupt
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

interrupt void NMI_ISR(void)       // Non-maskable interrupt
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
```

```
}

interrupt void ILLEGAL_ISR(void)   // Illegal operation TRAP
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm("          ESTOP0");
  for(;;);

}


interrupt void USER1_ISR(void)     // User Defined trap 1
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}

interrupt void USER2_ISR(void)     // User Defined trap 2
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);


}

interrupt void USER3_ISR(void)     // User Defined trap 3
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
```

```
}

interrupt void USER4_ISR(void)     // User Defined trap 4
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

interrupt void USER5_ISR(void)     // User Defined trap 5
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

interrupt void USER6_ISR(void)     // User Defined trap 6
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

interrupt void USER7_ISR(void)     // User Defined trap 7
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

interrupt void USER8_ISR(void)     // User Defined trap 8
{
  // Insert ISR Code here
```

```
  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

interrupt void USER9_ISR(void)     // User Defined trap 9
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

interrupt void USER10_ISR(void)    // User Defined trap 10
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

interrupt void USER11_ISR(void)    // User Defined trap 11
{
  // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

interrupt void USER12_ISR(void)     // User Defined trap 12
{
 // Insert ISR Code here

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
```

```
}

// ---------------------------------------------------------
// PIE Group 1 - MUXed into CPU INT1
// ---------------------------------------------------------

// INT1.1
interrupt void SEQ1INT_ISR(void)   //SEQ1 ADC
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code

  asm ("      ESTOP0");
  for(;;);

}

// INT1.2
interrupt void SEQ2INT_ISR(void)  //SEQ2 ADC
{

  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code

  asm("   ESTOP0");
  for(;;);

}
// INT1.3 - Reserved

// INT1.4
interrupt void  XINT1_ISR(void)
{
  // Insert ISR Code here
```

```
  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}

// INT1.5
interrupt void  XINT2_ISR(void)
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}

// INT1.6
interrupt void  ADCINT_ISR(void)     // ADC
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT1.7
interrupt void  TINT0_ISR(void)      // CPU-Timer 0
{
  // Insert ISR Code here
```

```
  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}


// INT1.8
interrupt void  WAKEINT_ISR(void)    // WD, LOW Power
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}


// ---------------------------------------------------------
// PIE Group 2 - MUXed into CPU INT2
// ---------------------------------------------------------

// INT2.1
interrupt void EPWM1_TZINT_ISR(void)    // EPWM-1
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP2;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT2.2
interrupt void EPWM2_TZINT_ISR(void)    // EPWM-2
```

```
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP2;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT2.3
interrupt void EPWM3_TZINT_ISR(void)    // EPWM-3
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP2;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}


// INT2.4
interrupt void EPWM4_TZINT_ISR(void)    // EPWM-4
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP2;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}


// INT2.5
interrupt void EPWM5_TZINT_ISR(void)    // EPWM-5
{
```

```
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP2;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT2.6
interrupt void EPWM6_TZINT_ISR(void)   // EPWM-6
{
  // Insert ISR Code here


  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP2;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT2.7 - Reserved
// INT2.8 - Reserved

// ------------------------------------------------------------
// PIE Group 3 - MUXed into CPU INT3
// ------------------------------------------------------------

// INT 3.1
interrupt void EPWM1_INT_ISR(void)     // EPWM-1
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
```

```
}

// INT3.2
interrupt void EPWM2_INT_ISR(void)     // EPWM-2
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT3.3
interrupt void EPWM3_INT_ISR(void)    // EPWM-3
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT3.4
interrupt void EPWM4_INT_ISR(void)    // EPWM-4
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT3.5
```

```
interrupt void EPWM5_INT_ISR(void)    // EPWM-5
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT3.6
interrupt void EPWM6_INT_ISR(void)    // EPWM-6
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT3.7 - Reserved
// INT3.8 - Reserved


// ------------------------------------------------------------
// PIE Group 4 - MUXed into CPU INT4
// ------------------------------------------------------------

// INT 4.1
interrupt void ECAP1_INT_ISR(void)    // ECAP-1
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP4;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
```

```
  asm ("      ESTOP0");
  for(;;);
}

// INT4.2
interrupt void ECAP2_INT_ISR(void)    // ECAP-2
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP4;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT4.3
interrupt void ECAP3_INT_ISR(void)    // ECAP-3
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP4;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT4.4
interrupt void ECAP4_INT_ISR(void)     // ECAP-4
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP4;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}
```

```
// INT4.5
interrupt void ECAP5_INT_ISR(void)     // ECAP-5
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP4;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}
// INT4.6
interrupt void ECAP6_INT_ISR(void)     // ECAP-6
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP4;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}
// INT4.7 - Reserved
// INT4.8 - Reserved

// ---------------------------------------------------------
// PIE Group 5 - MUXed into CPU INT5
// ---------------------------------------------------------

// INT 5.1
interrupt void EQEP1_INT_ISR(void)    // EQEP-1
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP5;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
```

```
  for(;;);
}

// INT5.2
interrupt void EQEP2_INT_ISR(void)    // EQEP-2
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP5;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT5.3 - Reserved
// INT5.4 - Reserved
// INT5.5 - Reserved
// INT5.6 - Reserved
// INT5.7 - Reserved
// INT5.8 - Reserved

// ---------------------------------------------------------
// PIE Group 6 - MUXed into CPU INT6
// ---------------------------------------------------------

// INT6.1
interrupt void SPIRXINTA_ISR(void)    // SPI-A
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP6;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
   asm ("      ESTOP0");
   for(;;);
}

// INT6.2
interrupt void SPITXINTA_ISR(void)     // SPI-A
{
```

```
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP6;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
   asm ("      ESTOP0");
   for(;;);
}

// INT6.3
interrupt void MRINTB_ISR(void)     // McBSP-B
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP6;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT6.4
interrupt void MXINTB_ISR(void)     // McBSP-B
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP6;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT6.5
interrupt void MRINTA_ISR(void)     // McBSP-A
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
```

```
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP6;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
   asm ("      ESTOP0");
   for(;;);
}

// INT6.6
interrupt void MXINTA_ISR(void)     // McBSP-A
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP6;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
   asm ("      ESTOP0");
   for(;;);
}

// INT6.7 - Reserved
// INT6.8 - Reserved




// ------------------------------------------------------------
// PIE Group 7 - MUXed into CPU INT7
// ------------------------------------------------------------

// INT7.1
interrupt void DINTCH1_ISR(void)     // DMA
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP7;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
   asm ("      ESTOP0");
   for(;;);
}
```

```
// INT7.2
interrupt void DINTCH2_ISR(void)     // DMA
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP7;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
   asm ("      ESTOP0");
   for(;;);
}

// INT7.3
interrupt void DINTCH3_ISR(void)     // DMA
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP7;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
   asm ("      ESTOP0");
   for(;;);
}

// INT7.4
interrupt void DINTCH4_ISR(void)     // DMA
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP7;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
   asm ("      ESTOP0");
   for(;;);
}

// INT7.5
interrupt void DINTCH5_ISR(void)     // DMA
```

```
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP7;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
   asm ("      ESTOP0");
   for(;;);
}

// INT7.6
interrupt void DINTCH6_ISR(void)     // DMA
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP7;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
   asm ("      ESTOP0");
   for(;;);
}

// INT7.7 - Reserved
// INT7.8 - Reserved

// ---------------------------------------------------------
// PIE Group 8 - MUXed into CPU INT8
// ---------------------------------------------------------

// INT8.1
interrupt void I2CINT1A_ISR(void)     // I2C-A
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP8;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
```

```
}

// INT8.2
interrupt void I2CINT2A_ISR(void)     // I2C-A
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP8;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);
}

// INT8.3 - Reserved
// INT8.4 - Reserved

// INT8.5
interrupt void SCIRXINTC_ISR(void)     // SCI-C
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP8;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}

// INT8.6
interrupt void SCITXINTC_ISR(void)     // SCI-C
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP8;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
```

```
  for(;;);

}

// INT8.7 - Reserved
// INT8.8 - Reserved


// ---------------------------------------------------------
// PIE Group 9 - MUXed into CPU INT9
// ---------------------------------------------------------

// INT9.1
interrupt void SCIRXINTA_ISR(void)     // SCI-A
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}

// INT9.2
interrupt void SCITXINTA_ISR(void)     // SCI-A
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}


// INT9.3
interrupt void SCIRXINTB_ISR(void)     // SCI-B
```

```
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}

// INT9.4
interrupt void SCITXINTB_ISR(void)     // SCI-B
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}

// INT9.5
interrupt void ECAN0INTA_ISR(void)  // eCAN-A
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}

// INT9.6
interrupt void ECAN1INTA_ISR(void)  // eCAN-A
```

```
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}

// INT9.7
interrupt void ECAN0INTB_ISR(void)  // eCAN-B
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}

// INT9.8
interrupt void ECAN1INTB_ISR(void)  // eCAN-B
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}

// ---------------------------------------------------------
// PIE Group 10 - MUXed into CPU INT10
```

```
// ---------------------------------------------------------

// INT10.1 - Reserved
// INT10.2 - Reserved
// INT10.3 - Reserved
// INT10.4 - Reserved
// INT10.5 - Reserved
// INT10.6 - Reserved
// INT10.7 - Reserved
// INT10.8 - Reserved


// ---------------------------------------------------------
// PIE Group 11 - MUXed into CPU INT11
// ---------------------------------------------------------

// INT11.1 - Reserved
// INT11.2 - Reserved
// INT11.3 - Reserved
// INT11.4 - Reserved
// INT11.5 - Reserved
// INT11.6 - Reserved
// INT11.7 - Reserved
// INT11.8 - Reserved

// ---------------------------------------------------------
// PIE Group 12 - MUXed into CPU INT12
// ---------------------------------------------------------

// INT12.1
interrupt void XINT3_ISR(void)  // External Interrupt
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP12;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}

// INT12.2
```

```
interrupt void XINT4_ISR(void)  // External Interrupt
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP12;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}

// INT12.3
interrupt void XINT5_ISR(void)  // External Interrupt
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP12;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}
// INT12.4
interrupt void XINT6_ISR(void)  // External Interrupt
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP12;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}

// INT12.5
interrupt void XINT7_ISR(void)  // External Interrupt
```

```
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP12;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}

// INT12.6 - Reserved
// INT12.7
interrupt void LVF_ISR(void)  // Latched overflow
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP12;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}
// INT12.8
interrupt void LUF_ISR(void)  // Latched underflow
{
  // Insert ISR Code here

  // To receive more interrupts from this PIE group, acknowledge this interrupt
  // PieCtrlRegs.PIEACK.all = PIEACK_GROUP12;

  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code
  asm ("      ESTOP0");
  for(;;);

}

//---------------------------------------------------------------------------
// Catch All Default ISRs:
```

```
//

interrupt void PIE_RESERVED(void)  // Reserved space.  For test.
{
  asm ("      ESTOP0");
  for(;;);
}

interrupt void rsvd_ISR(void)      // For test
{
  asm ("      ESTOP0");
  for(;;);
}

//===========================================================================
// End of file.
//===========================================================================

SciCommsGui_F28335.c
//===========================================================================
//===========================================================================
//
// FILE: SciCommsGui.c
//
// TITLE: GP Comms kernel as an interface to external GUI
//
// Version: 22 April 2009 - Release 1.2 - Internal Release (BRL)
//===========================================================================
//===========================================================================
#include "PeripheralHeaderIncludes.h"

#define PktSize 6
#define CmdNumber 16
#define MAX_CMD_NUM 8


// Function prototypes for Command RECEIVE State machine
// -----------------------------------------------------
void GetCmdByte(void);
void EchoCmdByte(void);
void GetSizeByte(void);
void EchoSizeByte(void);
void GetDataByte(void);
void EchoDataByte(void);
void PackWord(void);
```

```
void PackArray(void);
void CmdInterpreter(void);

// Function prototypes for Command Interpreter and dispatcher
// -------------------------------------------------------------
void LifePulseTsk(void); // 0
void TextSet(void); // 1
void ButtonSet(void); // 2
void SliderSet(void); // 3
void VariableGet(void); // 4
void ArrayGet(void); // 5
void DataGet(void); // 6
void SpareTsk07(void); // 7
void SpareTsk08(void); // 8

void SendData(void);

// Variable declarations
void (*RcvTaskPointer)(void);  // State pointer for Command Packet Receive
void (*CmdDispatcher[CmdNumber])(void); // Array of pointers to Function (i.e. tasks)

extern int *varSetTxtList[];
extern int *varSetBtnList[];
extern int *varSetSldrList[];
extern int *varGetList[];
extern int *arrayGetList[];
extern int *dataGetList[];

extern int16 CommsOKflg, SerialCommsTimer;

Uint16 LowByteFlag, SendTaskPtr;
Uint16 RxChar, RxWord;
Uint16 CmdPacket[PktSize];
Uint16 TaskDoneFlag, NumWords, wordsLeftToGet;

Uint16 dataOut;
int16 *memDataPtr;

int16  RcvTskPtrShdw; // for debug

int16  delayer;

int16 MemGetPtr;
Uint32 MemGetAddress;
int16 MemGetAmount;
```

```
Uint32 Temp;


void SCIA_Init()
{
// Note: Assumes Clocks to SCIA are turned on in InitSysCtrl()
// Note: Assumes GPIO pins for SCIA are configured to Primary function

int j = 0;

  SciaRegs.SCICCR.all =0x0007;   // 1 stop bit,  No loopback
                                 // No parity,8 char bits,
                                 // async mode, idle-line protocol
SciaRegs.SCICTL1.all =0x0003;  // enable TX, RX, internal SCICLK,
                                 // Disable RX ERR, SLEEP, TXWAKE
SciaRegs.SCICTL2.all =0x0003;
SciaRegs.SCICTL2.bit.TXINTENA =0;
SciaRegs.SCICTL2.bit.RXBKINTENA =0;
    SciaRegs.SCIHBAUD    =0x0000;

#if DSP2833x_DEVICE_H
//SciaRegs.SCILBAUD = 0x0079; // 79h = 38.4Kbaud @ LSPCLK = 150/4 MHz
SciaRegs.SCILBAUD = 0x0050; // 50h = 57.6Kbaud @ LSPCLK = 150 /4 MHz
//SciaRegs.SCILBAUD = 0x0028; // 28h = 115.2Kbaud @ LSPCLK = 150/4 MHz
#elif DSP2802x_DEVICE_H || DSP2803x_DEVICE_H
//SciaRegs.SCILBAUD = 0x0031; // 31h = 38.4Kbaud @ LSPCLK = 60/4 MHz
SciaRegs.SCILBAUD = 0x0020; // 20h = 57.6Kbaud @ LSPCLK = 60/4 MHz
//SciaRegs.SCILBAUD = 0x0010; // 10h = 115.2Kbaud @ LSPCLK = 60/4 MHz
#else  // F280x or F2804x
//SciaRegs.SCILBAUD  =0x00A2; // A2h = 19.2Kbaud @ LSPCLK = 100/4 MHz
//SciaRegs.SCILBAUD  =0x0050; // 50h = 38.4Kbaud @ LSPCLK = 100/4 MHz
SciaRegs.SCILBAUD = 0x0035; // 35h = 57.6Kbaud @ LSPCLK = 100/4 MHz
//SciaRegs.SCILBAUD = 0x001B; // 1Bh = 115.2Kbaud @ LSPCLK = 100/4 MHz
#endif

SciaRegs.SCICTL1.all =0x0023; // Relinquish SCI from Reset

    //SciaRegs.SCIFFTX.all=0xE040; // ENable FIFO enhancement
    SciaRegs.SCIFFTX.all=0x8040; // DISable FIFO enhancement
    SciaRegs.SCIFFRX.all=0x204f;
    SciaRegs.SCIFFCT.all=0x0;
    SciaRegs.SCIPRI.bit.SOFT=0x0;
    SciaRegs.SCIPRI.bit.FREE=0x1;
```

```
RcvTaskPointer = &GetCmdByte; // Initialize the CmdPacket Rcv Handler state machine ptr
RcvTskPtrShdw = 1; // DEBUG
SendTaskPtr = 0; // Init to 1st state
LowByteFlag = 1; // Start with LSB during Byte-to-Word packing

dataOut = 0;
*memDataPtr = 0;

RcvTskPtrShdw = 0; // for debug

delayer = 0;

MemGetPtr = 0;
MemGetAddress = 0x00000000;
MemGetAmount = 0;

// clear Command Packet
for (j=0; j<PktSize; j++)
{
CmdPacket[j] = 0x0;
}

j=0;

// init all dispatch Tasks
  CmdDispatcher[0] = LifePulseTsk;
  CmdDispatcher[1] = TextSet;
  CmdDispatcher[2] = ButtonSet;
  CmdDispatcher[3] = SliderSet;
  CmdDispatcher[4] = VariableGet;
  CmdDispatcher[5] = ArrayGet;
  CmdDispatcher[6] = DataGet;
  CmdDispatcher[7] = SpareTsk07;
  CmdDispatcher[8] = SpareTsk08;

}

//=========================================================
// Host Command RECEIVE and DISPATCH State Machine
//=========================================================

//======== SM Entry Point ================
void SerialHostComms()
{
(*RcvTaskPointer)(); // Call routine pointed to by state pointer
```

```
}
//=======================================

void GetCmdByte(void) // Task 1
{
if (SciaRegs.SCIRXST.bit.RXRDY == 1) // check if a char has been received
{
RxChar = SciaRegs.SCIRXBUF.all;
RcvTaskPointer = &EchoCmdByte; // point to next state
SerialCommsTimer = 0;
//RcvTskPtrShdw = 2; // DEBUG
EchoCmdByte();
}

else if (SciaRegs.SCIRXST.bit.BRKDT == 1 || SerialCommsTimer > 2500) //~2 s timeout
{
// If break detected or serialport times out, reset SCI
//--- Needed by some serialports when code is run with an emulator
  SciaRegs.SCICCR.all =0x0007;    // 1 stop bit,  No loopback
                                     // No parity,8 char bits,
                                      // async mode, idle-line protocol
  SciaRegs.SCICTL1.all =0x0003;   // enable TX, RX, internal SCICLK,
                                      // Disable RX ERR, SLEEP, TXWAKE
  SciaRegs.SCICTL2.all =0x0000;

  SciaRegs.SCICTL1.all =0x0023; // Relinquish SCI from Reset

asm(" RPT#8 || NOP");
//---

SendTaskPtr = 0; // Init to 1st state
SerialCommsTimer = 0;

CommsOKflg = 0;
RcvTaskPointer = &GetCmdByte; // go back and wait for new CMD
}
}

void EchoCmdByte(void) // Task 2
{
    if(SciaRegs.SCICTL2.bit.TXRDY == 1) // is TXBUF empty ?, i.e. TXRDY = 1
    {
     SciaRegs.SCITXBUF=RxChar; // if yes, echo back the received char
     CmdPacket[0] = RxChar;
RcvTaskPointer = &GetSizeByte;
```

```
//RcvTskPtrShdw = 3; // DEBUG
//RcvTaskPointer = &GetCmdByte; // Un-comment for simple echo test
SerialCommsTimer = 0; // Reset Time-out timer
}

}

void GetSizeByte(void) // Task 3
{
if (SciaRegs.SCIRXST.bit.RXRDY == 1) // check if a char has been received
{
RxChar = SciaRegs.SCIRXBUF.all;

RcvTaskPointer = &EchoSizeByte; // point to next state
//RcvTskPtrShdw = 4; // DEBUG
EchoSizeByte();
}

else if (SerialCommsTimer > 1000) // 1000*1mS = 1.0 sec timeout
{
CommsOKflg = 0;
RcvTaskPointer = &GetCmdByte; // Abort, go back wait for new CMD
SerialCommsTimer = 0;
}
}

void EchoSizeByte(void) // Task 4
{
    if(SciaRegs.SCICTL2.bit.TXRDY == 1) // is TXBUF empty ?, i.e. TXRDY = 1
    {
     SciaRegs.SCITXBUF=RxChar; // if yes, echo back the received char
     CmdPacket[1] = RxChar;
RcvTaskPointer = &GetDataByte;
//RcvTskPtrShdw = 5; // DEBUG
//RcvTaskPointer = &GetCmdByte; // Un-comment for Test
SerialCommsTimer = 0; // Reset Time-out timer
    }
}

void GetDataByte(void) // Task 5
{
if (SciaRegs.SCIRXST.bit.RXRDY == 1) // check if a char has been received
{
RxChar = SciaRegs.SCIRXBUF.all;
RcvTaskPointer = &EchoDataByte; // point to next state
```

```
//RcvTskPtrShdw = 6; // DEBUG
EchoDataByte();
}

else if (SerialCommsTimer > 500) // 1000*1mS = 1 sec timeout
{
CommsOKflg = 0;
RcvTaskPointer = &GetCmdByte; // Abort, go back wait for new CMD
SerialCommsTimer = 0;
}

}

void EchoDataByte(void) // Task 6
{
    if(SciaRegs.SCICTL2.bit.TXRDY == 1) // is TXBUF empty ?, i.e. TXRDY = 1
    {
     SciaRegs.SCITXBUF=RxChar; // if yes, echo back the received char
RcvTaskPointer = &PackWord;
//RcvTskPtrShdw = 7; // DEBUG
    }
}

void PackWord(void) // expects LSB first then MSB // Task 7
{
if(LowByteFlag == 1)
{
RxWord = RxChar;
LowByteFlag = 0;
RcvTaskPointer = &GetDataByte;
//RcvTskPtrShdw = 5; // DEBUG
GetDataByte();
}
else
{
RxWord = RxWord | (RxChar<<8);
LowByteFlag = 1;
CmdPacket[2] = RxWord; // store data in packet
RcvTaskPointer = &CmdInterpreter;
//RcvTskPtrShdw = 8; // DEBUG
TaskDoneFlag = 0;  // indicate new task underway
}
}

void CmdInterpreter(void) // Task 8
```

```
{
if (TaskDoneFlag == 0)
{
  (*CmdDispatcher[ CmdPacket[0] ] )();   // dispatch Task
}

// Incase Task never finishes
if (SerialCommsTimer > 2500) // 2500*1mS = 2.5 sec timeout
{
CommsOKflg = 0;
RcvTaskPointer = &GetCmdByte; // Abort, go back wait for new CMD
SerialCommsTimer = 0;
}
if (TaskDoneFlag == 1)
{
RcvTaskPointer = &GetCmdByte;
  //RcvTskPtrShdw = 1;    // DEBUG
}
}

//===========================================================
// Slave Tasks commanded by Host
//===========================================================
void LifePulseTsk(void)  // CmdPacket[0] = 0
{
if (CmdPacket[2]==0x0000 && CmdPacket[1]==0x00)  //LED2-ON
{
#if DSP2802x_DEVICE
GpioDataRegs.GPASET.bit.GPIO12=1;
#else
GpioDataRegs.GPASET.bit.GPIO31=1;
#endif
}
if (CmdPacket[2]==0x0001 && CmdPacket[1]==0x00)   //LED2-OFF
{
#if DSP2802x_DEVICE
GpioDataRegs.GPACLEAR.bit.GPIO12=1;
#else
GpioDataRegs.GPACLEAR.bit.GPIO31=1;
#endif
}
if (CmdPacket[2]==0x0002 && CmdPacket[1]==0x00)  //LED2-Toggle
{
#if DSP2802x_DEVICE
GpioDataRegs.GPATOGGLE.bit.GPIO12=1;
```

```
#else
GpioDataRegs.GPATOGGLE.bit.GPIO31=1;
#endif
}

CommsOKflg = 1;
SerialCommsTimer = 0;
TaskDoneFlag = 1;
}
//-----------------------------------------------------------
void TextSet(void) // CmdPacket[0] = 1
{
*varSetTxtList[CmdPacket[1]] = CmdPacket[2];

TaskDoneFlag = 1; // indicate Task execution is complete
}
//-----------------------------------------------------------
void ButtonSet(void)  // CmdPacket[0] = 2
{
*varSetBtnList[CmdPacket[1]] = CmdPacket[2];

TaskDoneFlag = 1; // indicate Task execution is complete
}
//-----------------------------------------------------------
void SliderSet(void)  // CmdPacket[0] = 3
{
*varSetSldrList[CmdPacket[1]] = CmdPacket[2];

TaskDoneFlag = 1; // indicate Task execution is complete
}
//-----------------------------------------------------------
void VariableGet(void) // CmdPacket[0] = 4
{
SendData();
}
//-----------------------------------------------------------
//Send a Uint16 array one element at a time
void ArrayGet(void) // CmdPacket[0] = 5
{
SendData();
}
//-----------------------------------------------------------
void DataGet(void) // CmdPacket[0] = 6
{
switch(MemGetPtr)
```

```
{
case 0:
MemGetAddress = CmdPacket[2];
MemGetPtr = 1;

wordsLeftToGet = 1;
SendTaskPtr = 1;
TaskDoneFlag = 1;
break;

case 1:
Temp = CmdPacket[2];
MemGetAddress = MemGetAddress + (Temp<<16);
memDataPtr = (int16*)MemGetAddress;
dataOut = *memDataPtr;
SendData();

if(TaskDoneFlag == 1)
{
MemGetPtr = 0;
}
break;
}

//TaskDoneFlag = 1;  // indicate Task execution is complete
}
//-------------------------------------------------------------
void SpareTsk07(void) // CmdPacket[0] = 7
{
TaskDoneFlag = 1;  // indicate Task execution is complete
}
//-------------------------------------------------------------
void SpareTsk08(void) // CmdPacket[0] = 8
{
TaskDoneFlag = 1;  // indicate Task execution is complete
}
//-------------------------------------------------------------

void SendData(void)
{
switch(SendTaskPtr)
{
case 0:  //initialization
if(CmdPacket[0] == 0x04)
{
```

```
memDataPtr = (int16 *) varGetList[CmdPacket[1]];
}
else
{
memDataPtr = (int16 *) arrayGetList[CmdPacket[1]];
}

dataOut = *memDataPtr;
wordsLeftToGet = CmdPacket[2];
//Note that case 0 rolls into case 1 (no break)

case 1:  //send LSB
if(wordsLeftToGet > 0)
{
if (SciaRegs.SCICTL2.bit.TXRDY == 1)
{
SciaRegs.SCITXBUF = dataOut & 0x000000FF;
SendTaskPtr = 2;
}
else
{
TaskDoneFlag = TaskDoneFlag;
break;
}
}
else
{
SendTaskPtr = 0;
TaskDoneFlag = 1;
break;
}

case 2: //send MSB
if (SciaRegs.SCICTL2.bit.TXRDY == 1)
{
SciaRegs.SCITXBUF = (dataOut>>8 & 0x000000FF);

memDataPtr = memDataPtr + 1;
dataOut = *memDataPtr;
wordsLeftToGet = wordsLeftToGet - 1;
SendTaskPtr = 1;
}
break;
}
}
```

```
Interrupts.h
#ifndef INTERRUPTS_H_
#define INTERRUPTS_H_


void InitInterrupts();
void EnableInterrupts();
#endif /*INTERRUPTS_H_*/

Interrupts.c
#include "PeripheralHeaderIncludes_F28335.h"
#include "Interrupts.h"
#include "ADCHandler.h"
#include "SerialHandler.h"

void InitInterrupts(){
EALLOW;
// Disable CPU interrupts and clear all CPU interrupt flags:
    IER = 0x0000;
    IFR = 0x0000;

PieVectTable.ADCINT = &adc_isr;
PieVectTable.SCIRXINTA = &scirxa_isr;
PieVectTable.SCITXINTA = &scitxa_isr;
EDIS;
}

void EnableInterrupts(){
EALLOW;
PieCtrlRegs.PIEIER1.bit.INTx6 = 1; //Enable ADCINT in PIE;
PieCtrlRegs.PIEIER9.bit.INTx1 = 1; // Enable SCIRXINTA in PIE
PieCtrlRegs.PIEIER9.bit.INTx2 = 1; // Enable SCITXINTA in PIE
IER |= M_INT1; //Enable CPU Interrupt block 1 - ADC,Timer0
IER |= M_INT9; // Enable CPU Interrupt block 9 - SCI
PieCtrlRegs.PIECTRL.bit.ENPIE = 1; //Enable Peripheral Interrupts.
EDIS;
EINT;
}

TimerHandler.h
#ifndef TIMERHANDLER_H_
#define TIMERHANDLER_H_

#include "Globals.h"
```

```
#include "PeripheralHeaderIncludes_F28335.h"
//#include "DSP28x_Project.h"     // Device Headerfile and Examples Include File


void InitTimerModule(void);
void Timer0Callback(void);

interrupt void timer0_isr();

#endif /*TIMERHANDLER_H_*/

TimerHandler.c
#include "TimerHandler.h"
#include "I2CHandler.h"
#include "Globals.h"

Uint16 timerCounter;

void InitTimerModule(){
CpuTimer0Regs.TCR.bit.TSS = 1; //Stop timer.
CpuTimer0Regs.TCR.bit.TIF = 1; //Clear any triggered interrupts.
CpuTimer0Regs.TPR.all = 0x0000;
CpuTimer0Regs.TPRH.all = 0x0000;
// CpuTimer0Regs.PRD.half.MSW = 0x0907; //Setup timer for 1 second at 150MHz.
// CpuTimer0Regs.PRD.half.LSW = 0xF00F;
// CpuTimer0Regs.PRD.half.MSW = 0x0241; //Setup timer for 0.25 second at 150MHz.
// CpuTimer0Regs.PRD.half.LSW = 0xFC03;
CpuTimer0Regs.PRD.half.MSW = 0x002D; //Setup timer for 20ms second at 150MHz.
CpuTimer0Regs.PRD.half.LSW = 0xFE02;
CpuTimer0Regs.TCR.all = 0xC020; //Start the timer back and reload the period into the ti
//Enable the interrupt.
//Timers count down from the period value.

timerCounter = 0;
}

interrupt void timer0_isr(){

// if(timerCounter == 0)
// {
// EPwm2Regs.CMPA.half.CMPA = (EPwm2Regs.CMPA.half.CMPA>>1);
// timerCounter++;
// }
// else
// {
```

```
// EPwm2Regs.CMPA.half.CMPA = (EPwm2Regs.CMPA.half.CMPA<<1);
// timerCounter = 0;
// }
MainFlags.bit.Timer0Flag = 1;
CpuTimer0Regs.TCR.bit.TIF = 1; //Clear timer interrupt.
PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;   // Acknowledge interrupt to PIE
}

void Timer0Callback(){
// static Uint16 counter = 0;
// Uint16 tmp2 = 0;
LEDStatus.all = 0;
LEDStatus2.all = 0;
ButtonStatus.all = PollButtons();
if(ButtonStatus.nib.PB < 0xF){
EPwm1Regs.TZFRC.bit.OST = 1; //Kill PWM1
EPwm2Regs.TZFRC.bit.OST = 1; //Kill PWM2
EPwm3Regs.TZFRC.bit.OST = 1; //Kill PWM3
EPwm4Regs.TZFRC.bit.OST = 1; //Kill PWM4
EPwm5Regs.TZFRC.bit.OST = 1; //Kill PWM5
EPwm6Regs.TZFRC.bit.OST = 1; //Kill PWM6
}
if(ButtonStatus.bit.PB1 == 0)
LEDStatus2.bit.LED0 = LCD_ON;
if(ButtonStatus.bit.PB2 == 0)
LEDStatus2.bit.LED1 = LCD_ON;
if(ButtonStatus.bit.PB3 == 0)
LEDStatus2.bit.LED2 = LCD_ON;
if(ButtonStatus.bit.PB4 == 0)
LEDStatus2.bit.LED3 = LCD_ON;
if(ButtonStatus.bit.SW1_1 == 0)
LEDStatus2.bit.LED4 = LCD_ON;
if(ButtonStatus.bit.SW1_2 == 0)
LEDStatus2.bit.LED5 = LCD_ON;
if(ButtonStatus.bit.SW1_3 == 0)
LEDStatus2.bit.LED6 = LCD_ON;
if(ButtonStatus.bit.SW1_4 == 0)
LEDStatus2.bit.LED7 = LCD_ON;
SetPCA3Status(LEDStatus2.all);
PhaseStatus = PollPhases();
if((PhaseStatus & 0x0001) == 0){  //Phase 1 Board plugged in.
if(EPwm1Regs.TZFLG.bit.OST == 1){ //channel off
LEDStatus.bit.LED0 = LCD_BLINK1;
}
else{
```

```
LEDStatus.bit.LED0 = LCD_ON;
}
}

if((PhaseStatus & 0x0002) == 0){ // Phase 2 Board plugged in.
if(EPwm2Regs.TZFLG.bit.OST == 1){ //channel off
LEDStatus.bit.LED1 = LCD_BLINK1;
}
else{
LEDStatus.bit.LED1 = LCD_ON;
}
}

if((PhaseStatus & 0x0004) == 0){ // Phase 3 Board plugged in.
if(EPwm3Regs.TZFLG.bit.OST == 1){ //channel off
LEDStatus.bit.LED2 = LCD_BLINK1;
}
else{
LEDStatus.bit.LED2 = LCD_ON;
}
}

if((PhaseStatus & 0x0008) == 0){ // Phase 4 Board plugged in.
if(EPwm4Regs.TZFLG.bit.OST == 1){ //channel off
LEDStatus.bit.LED3 = LCD_BLINK1;
}
else{
LEDStatus.bit.LED3 = LCD_ON;
}
}

if((PhaseStatus & 0x0010) == 0){ // Phase 5 Board plugged in.
if(EPwm5Regs.TZFLG.bit.OST == 1){ //channel off
LEDStatus.bit.LED4 = LCD_BLINK1;
}
else{
LEDStatus.bit.LED4 = LCD_ON;
}
}
// if(counter == 0){
// tmp2 = LCD_BLINK2
// LEDStatus += tmp2 <<10;
// counter++;
// }
// else{
```

```
// tmp2 = LCD_OFF
// LEDStatus += tmp2 <<10;
// counter = 0;
// }
// tmp += 0xAFA0;
SetPCA1Status(LEDStatus.all);


}
```

PWMHandler.h

```
#ifndef PWMHANDLER_H_
#define PWMHANDLER_H_


//#include "DSP28x_Project.h"     // Device Headerfile and Examples Include File
#include "PeripheralHeaderIncludes_F28335.h"

//*********************** PWM Parameters ************************************
//Period definitions
#define PRD400KHZ 0x374;


//Deadband definitions
#define EPWM_MAX_DB   0x03FF
#define EPWM_MIN_DB   0x0001

#define DB_UP   1
#define DB_DOWN 0

//*********************** End PWM Parameters ********************************

//EPWM Variables
//extern Uint32  EPwm1TimerIntCount;
//extern Uint32  EPwm2TimerIntCount;
//extern Uint32  EPwm3TimerIntCount;
//extern Uint16  EPwm1_DB_Direction;
//extern Uint16  EPwm2_DB_Direction;
//extern Uint16  EPwm3_DB_Direction;

extern Uint16 PwmPeriods[14];
extern Uint16 *PwmPointer;

void InitPwmModule();
```

```
//Pwm init functions
void EPwm1Init(void);
void EPwm2Init(void);
void EPwm3Init(void);
void EPwm4Init(void);
void EPwm5Init(void);
void EPwm6Init(void);

void EnableAllPWM(void);
void DisableAllPWM(void);

interrupt void epwm1_isr(void);
interrupt void epwm2_isr(void);
interrupt void epwm3_isr(void);


//PWM Constants

#ifdef __cplusplus
extern "C" {
#endif

// TBCTL (Time-Base Control)
//==========================
// CTRMODE bits
#define TB_COUNT_UP 0x0
#define TB_COUNT_DOWN 0x1
#define TB_COUNT_UPDOWN 0x2
#define TB_FREEZE 0x3
// PHSEN bit
#define TB_DISABLE 0x0
#define TB_ENABLE 0x1
// PRDLD bit
#define TB_SHADOW 0x0
#define TB_IMMEDIATE 0x1
// SYNCOSEL bits
#define TB_SYNC_IN 0x0
#define TB_CTR_ZERO 0x1
#define TB_CTR_CMPB 0x2
#define TB_SYNC_DISABLE 0x3
// HSPCLKDIV and CLKDIV bits
#define TB_DIV1 0x0
#define TB_DIV2 0x1
#define TB_DIV4 0x2
// PHSDIR bit
```

```
#define TB_DOWN 0x0
#define TB_UP 0x1

// CMPCTL (Compare Control)
//=========================
// LOADAMODE and LOADBMODE bits
#define CC_CTR_ZERO 0x0
#define CC_CTR_PRD 0x1
#define CC_CTR_ZERO_PRD 0x2
#define CC_LD_DISABLE 0x3
// SHDWAMODE and SHDWBMODE bits
#define CC_SHADOW 0x0
#define CC_IMMEDIATE 0x1

// AQCTLA and AQCTLB (Action Qualifier Control)
//=============================================
// ZRO, PRD, CAU, CAD, CBU, CBD bits
#define AQ_NO_ACTION 0x0
#define AQ_CLEAR 0x1
#define AQ_SET 0x2
#define AQ_TOGGLE 0x3

// DBCTL (Dead-Band Control)
//==========================
// OUT MODE bits
#define DB_DISABLE 0x0
#define DBA_ENABLE 0x1
#define DBB_ENABLE 0x2
#define DB_FULL_ENABLE 0x3
// POLSEL bits
#define DB_ACTV_HI 0x0
#define DB_ACTV_LOC 0x1
#define DB_ACTV_HIC 0x2
#define DB_ACTV_LO 0x3
// IN MODE
#define DBA_ALL         0x0
#define DBB_RED_DBA_FED 0x1
#define DBA_RED_DBB_FED 0x2
#define DBB_ALL         0x3

// CHPCTL (chopper control)
//=========================
// CHPEN bit
#define CHP_DISABLE 0x0
#define CHP_ENABLE 0x1
```

```
// CHPFREQ bits
#define CHP_DIV1 0x0
#define CHP_DIV2 0x1
#define CHP_DIV3 0x2
#define CHP_DIV4 0x3
#define CHP_DIV5 0x4
#define CHP_DIV6 0x5
#define CHP_DIV7 0x6
#define CHP_DIV8 0x7
// CHPDUTY bits
#define CHP1_8TH 0x0
#define CHP2_8TH 0x1
#define CHP3_8TH 0x2
#define CHP4_8TH 0x3
#define CHP5_8TH 0x4
#define CHP6_8TH 0x5
#define CHP7_8TH 0x6

// TZSEL (Trip Zone Select)
//=========================
// CBCn and OSHTn bits
#define TZ_DISABLE 0x0
#define TZ_ENABLE 0x1

// TZCTL (Trip Zone Control)
//=========================
// TZA and TZB bits
#define TZ_HIZ 0x0
#define TZ_FORCE_HI 0x1
#define TZ_FORCE_LO 0x2
#define TZ_NO_CHANGE 0x3

// ETSEL (Event Trigger Select)
//============================
#define ET_CTR_ZERO 0x1
#define ET_CTR_PRD 0x2
#define ET_CTRU_CMPA 0x4
#define ET_CTRD_CMPA 0x5
#define ET_CTRU_CMPB 0x6
#define ET_CTRD_CMPB 0x7

// ETPS (Event Trigger Pre-scale)
//==============================
// INTPRD, SOCAPRD, SOCBPRD bits
#define ET_DISABLE 0x0
```

```
#define ET_1ST 0x1
#define ET_2ND 0x2
#define ET_3RD 0x3


//---------------------------------
// HRPWM (High Resolution PWM)
//=================================
// HRCNFG
#define HR_Disable 0x0
#define HR_REP 0x1
#define HR_FEP 0x2
#define HR_BEP 0x3

#define HR_CMP 0x0
#define HR_PHS 0x1

#define HR_CTR_ZERO 0x0
#define HR_CTR_PRD 0x1


#ifdef __cplusplus
}
#endif /* extern "C" */


#endif /*PWMHANDLER_H_*/
```

PWMHandler.c

```
#include "PWMHandler.h"
#include "F28335_EPwm_Init.h"
#include "Globals.h"

//EPWM Variables
//Uint32  EPwm1TimerIntCount;
//Uint32  EPwm2TimerIntCount;
//Uint32  EPwm3TimerIntCount;
//Uint16  EPwm1_DB_Direction;
//Uint16  EPwm2_DB_Direction;
//Uint16  EPwm3_DB_Direction;

Uint16 PwmPeriods[14];
Uint16 *PwmPointer;
```

```
//
//interrupt void epwm1_isr(void)
//{
////   if(EPwm1_DB_Direction == DB_UP)
////   {
////       if(EPwm1Regs.DBFED < EPWM1_MAX_DB)
////       {
////          EPwm1Regs.DBFED++;
////          EPwm1Regs.DBRED++;
////       }
////       else
////       {
////          EPwm1_DB_Direction = DB_DOWN;
////          EPwm1Regs.DBFED--;
////          EPwm1Regs.DBRED--;
////       }
////   }
////   else
////   {
////       if(EPwm1Regs.DBFED == EPWM1_MIN_DB)
////       {
////          EPwm1_DB_Direction = DB_UP;
////          EPwm1Regs.DBFED++;
////          EPwm1Regs.DBRED++;
////       }
////       else
////       {
////          EPwm1Regs.DBFED--;
////          EPwm1Regs.DBRED--;
////       }
////   }
//   EPwm1TimerIntCount++;
//
//   // Clear INT flag for this timer
//   EPwm1Regs.ETCLR.bit.INT = 1;
//
//   // Acknowledge this interrupt to receive more interrupts from group 3
//   PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
//
//}
//
//interrupt void epwm2_isr(void)
//{
//
////   if(EPwm2_DB_Direction == DB_UP)
```

```
////   {
////       if(EPwm2Regs.DBFED < EPWM2_MAX_DB)
////       {
////          EPwm2Regs.DBFED++;
////          EPwm2Regs.DBRED++;
////       }
////       else
////       {
////          EPwm2_DB_Direction = DB_DOWN;
////          EPwm2Regs.DBFED--;
////          EPwm2Regs.DBRED--;
////       }
////   }
////   else
////   {
////       if(EPwm2Regs.DBFED == EPWM2_MIN_DB)
////       {
////          EPwm2_DB_Direction = DB_UP;
////          EPwm2Regs.DBFED++;
////          EPwm2Regs.DBRED++;
////       }
////       else
////       {
////          EPwm2Regs.DBFED--;
////          EPwm2Regs.DBRED--;
////       }
////   }
//
//   EPwm2TimerIntCount++;
//
//   // Clear INT flag for this timer
//   EPwm2Regs.ETCLR.bit.INT = 1;
//
//   // Acknowledge this interrupt to receive more interrupts from group 3
//   PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
//
//}
//
//interrupt void epwm3_isr(void)
//{
//   if(EPwm3_DB_Direction == DB_UP)
//   {
//       if(EPwm3Regs.DBFED < EPWM3_MAX_DB)
//       {
//          EPwm3Regs.DBFED++;
```

```
//          EPwm3Regs.DBRED++;
//       }
//       else
//       {
//          EPwm3_DB_Direction = DB_DOWN;
//          EPwm3Regs.DBFED--;
//          EPwm3Regs.DBRED--;
//       }
//   }
//   else
//   {
//       if(EPwm3Regs.DBFED == EPWM3_MIN_DB)
//       {
//          EPwm3_DB_Direction = DB_UP;
//          EPwm3Regs.DBFED++;
//          EPwm3Regs.DBRED++;
//       }
//       else
//       {
//          EPwm3Regs.DBFED--;
//          EPwm3Regs.DBRED--;
//       }
//   }
//
//
//   EPwm3TimerIntCount++;
//
//   // Clear INT flag for this timer
//   EPwm3Regs.ETCLR.bit.INT = 1;
//
//   // Acknowledge this interrupt to receive more interrupts from group 3
//   PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
//
//}

void InitPwmModule(){

InitEPwm1Gpio();
    InitEPwm2Gpio();
    InitEPwm3Gpio();
    InitEPwm4Gpio();
    InitEPwm5Gpio();
//    InitEPwm6Gpio();

    PwmPointer = PwmPeriods;
```

```
    PwmPeriods[0] = 0x358;
    PwmPeriods[1] = 0x2ED;
    PwmPeriods[2] = 0x29A;
    PwmPeriods[3] = 0x257;
    PwmPeriods[4] = 0x220;
    PwmPeriods[5] = 0x1F3;
    PwmPeriods[6] = 0x1CC;
    PwmPeriods[7] = 0x1AB;
    PwmPeriods[8] = 0x18F;
    PwmPeriods[9] = 0x176;
    PwmPeriods[10] = 0x160;
    PwmPeriods[11] = 0x14C;
    PwmPeriods[12] = 0x13B;
PwmPeriods[13] = 0x12B;

EPwm1Init();
   EPwm2Init();
    EPwm3Init();
    EPwm4Init();
    EPwm5Init();
    EPwm6Init();

    EALLOW;
    EPwm1Regs.TZFRC.bit.OST = 1; //Kill PWM1
    EPwm2Regs.TZFRC.bit.OST = 1; //Kill PWM2
    EPwm3Regs.TZFRC.bit.OST = 1; //Kill PWM3
    EPwm4Regs.TZFRC.bit.OST = 1; //Kill PWM3
    EPwm5Regs.TZFRC.bit.OST = 1; //Kill PWM3
//      EPwm6Regs.TZFRC.bit.OST = 1; //Kill PWM3
    EDIS;
    PhaseStatus.all = PhaseStatus.all & ALLPHASESOFF;

}

void EPwm1Init()
{

   EPwm1Regs.TBPRD = 880;//374;                        // Set timer period
   EPwm1Regs.TBPHS.half.TBPHS = 0x0000;           // Phase is 0
   EPwm1Regs.TBCTR = 0x0000;                      // Clear counter

   // Setup TBCLK
   EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;    // Count up
   EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;        // Disable phase loading
//   EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV4;       // Clock ratio to SYSCLKOUT
```

```
//   EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV4;
   EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Clock ratio to SYSCLKOUT
   EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;
   EPwm1Regs.TBCTL.bit.SYNCOSEL = TB_CTR_ZERO;

   EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;    // Load registers every ZERO
   EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
   EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
   EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

   EALLOW;
   EPwm1Regs.HRCNFG.all = 0x0;
   EPwm1Regs.HRCNFG.bit.EDGMODE = HR_FEP; //Falling Edge Position
   EPwm1Regs.HRCNFG.bit.CTLMODE = HR_CMP; //CMPAHR controls value.
   EPwm1Regs.HRCNFG.bit.HRLOAD = HR_CTR_ZERO; //Shadow load on CTR=0
   EDIS;
   // Setup compare
   EPwm1Regs.CMPA.half.CMPA = 45;
   EPwm1Regs.CMPA.half.CMPAHR = 0x0000;
//   EPwm1Regs.CMPB = 3000;

   // Set actions
   EPwm1Regs.AQCTLA.bit.ZRO = AQ_SET;
   EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR;             // Set PWM1A on Zero
//   EPwm1Regs.AQCTLA.bit.CAD = AQ_CLEAR;

   EPwm1Regs.AQCTLB.bit.ZRO = AQ_SET;
   EPwm1Regs.AQCTLB.bit.CAU = AQ_CLEAR;          // Set PWM1A on Zero
//   EPwm1Regs.AQCTLB.bit.CAD = AQ_SET;

   // Active Low PWMs - Setup Deadband
   EPwm1Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
   //EPwm1Regs.DBCTL.bit.POLSEL = DB_ACTV_LO;
   EPwm1Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
   EPwm1Regs.DBCTL.bit.IN_MODE = DBA_ALL;
   EPwm1Regs.DBRED = EPWM_MIN_DB;
   EPwm1Regs.DBFED = EPWM_MIN_DB;

// Setup Trip-Zone for ePWM1:
EALLOW;
EPwm1Regs.TZSEL.bit.OSHT1 = TZ_ENABLE; //TZ1 is a one-shot killer
EPwm1Regs.TZCTL.bit.TZA = TZ_FORCE_LO;
EPwm1Regs.TZCTL.bit.TZB = TZ_FORCE_LO;
EPwm1Regs.TZEINT.bit.OST = TZ_ENABLE;
EDIS;
```

```
   EPwm1Regs.ETSEL.bit.INTEN = 0;                // disable INT
}

void EPwm2Init()
{

   EPwm2Regs.TBPRD = 880;//374;                        // Set timer period
   EPwm2Regs.TBPHS.half.TBPHS = 176;//75;//0x00BB;           // Phase is 0
   EPwm2Regs.TBCTR = 0x0000;                      // Clear counter

   // Setup TBCLK
   EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;    // Count up
   EPwm2Regs.TBCTL.bit.PHSEN = TB_ENABLE;        // Disable phase loading
//   EPwm2Regs.TBCTL.bit.HSPCLKDIV = TB_DIV4;       // Clock ratio to SYSCLKOUT
//   EPwm2Regs.TBCTL.bit.CLKDIV = TB_DIV4;
   EPwm2Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Clock ratio to SYSCLKOUT
   EPwm2Regs.TBCTL.bit.CLKDIV = TB_DIV1;
   EPwm2Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;

   EPwm2Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;    // Load registers every ZERO
   EPwm2Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
   EPwm2Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
   EPwm2Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

   EALLOW;
   EPwm2Regs.HRCNFG.all = 0x0;
   EPwm2Regs.HRCNFG.bit.EDGMODE = HR_FEP; //Falling Edge Position
   EPwm2Regs.HRCNFG.bit.CTLMODE = HR_CMP; //CMPAHR controls value.
   EPwm2Regs.HRCNFG.bit.HRLOAD = HR_CTR_ZERO; //Shadow load on CTR=0
   EDIS;
   // Setup compare
   EPwm2Regs.CMPA.half.CMPA = 45;
   EPwm2Regs.CMPA.half.CMPAHR = 0x0000;
//   EPwm2Regs.CMPB = 3000;

   // Set actions
   EPwm2Regs.AQCTLA.bit.ZRO = AQ_SET;
   EPwm2Regs.AQCTLA.bit.CAU = AQ_CLEAR;             // Set PWM1A on Zero
//   EPwm2Regs.AQCTLA.bit.CAD = AQ_CLEAR;

   EPwm2Regs.AQCTLB.bit.ZRO = AQ_SET;
   EPwm2Regs.AQCTLB.bit.CAU = AQ_CLEAR;          // Set PWM1A on Zero
//   EPwm2Regs.AQCTLB.bit.CAD = AQ_SET;
```

```
   // Active Low PWMs - Setup Deadband
   EPwm2Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
   //EPwm2Regs.DBCTL.bit.POLSEL = DB_ACTV_LO;
   EPwm2Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
   EPwm2Regs.DBCTL.bit.IN_MODE = DBA_ALL;
   EPwm2Regs.DBRED = EPWM_MIN_DB;
   EPwm2Regs.DBFED = EPWM_MIN_DB;

// Setup Trip-Zone for ePWM1:
EALLOW;
EPwm2Regs.TZSEL.bit.OSHT1 = TZ_ENABLE; //TZ1 is a one-shot killer
EPwm2Regs.TZCTL.bit.TZA = TZ_FORCE_LO;
EPwm2Regs.TZCTL.bit.TZB = TZ_FORCE_LO;
EPwm2Regs.TZEINT.bit.OST = TZ_ENABLE;
EDIS;

   EPwm2Regs.ETSEL.bit.INTEN = 0;                // disable INT
}

void EPwm3Init()
{

   EPwm3Regs.TBPRD = 880;//374;                       // Set timer period
   EPwm3Regs.TBPHS.half.TBPHS = 352;//150;           // Phase is 0
   EPwm3Regs.TBCTR = 0x0000;                      // Clear counter

   // Setup TBCLK
   EPwm3Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;    // Count up
   EPwm3Regs.TBCTL.bit.PHSEN = TB_ENABLE;        // Disable phase loading
//   EPwm3Regs.TBCTL.bit.HSPCLKDIV = TB_DIV4;       // Clock ratio to SYSCLKOUT
//   EPwm3Regs.TBCTL.bit.CLKDIV = TB_DIV4;
   EPwm3Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Clock ratio to SYSCLKOUT
   EPwm3Regs.TBCTL.bit.CLKDIV = TB_DIV1;
   EPwm3Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;

   EPwm3Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;    // Load registers every ZERO
   EPwm3Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
   EPwm3Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
   EPwm3Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

   EALLOW;
   EPwm3Regs.HRCNFG.all = 0x0;
   EPwm3Regs.HRCNFG.bit.EDGMODE = HR_FEP; //Falling Edge Position
   EPwm3Regs.HRCNFG.bit.CTLMODE = HR_CMP; //CMPAHR controls value.
   EPwm3Regs.HRCNFG.bit.HRLOAD = HR_CTR_ZERO; //Shadow load on CTR=0
```

```
   EDIS;
   // Setup compare
   EPwm3Regs.CMPA.half.CMPA = 45;
   EPwm3Regs.CMPA.half.CMPAHR = 0x0000;
//   EPwm3Regs.CMPB = 3000;

   // Set actions
   EPwm3Regs.AQCTLA.bit.ZRO = AQ_SET;
   EPwm3Regs.AQCTLA.bit.CAU = AQ_CLEAR;             // Set PWM1A on Zero
//   EPwm3Regs.AQCTLA.bit.CAD = AQ_CLEAR;

   EPwm3Regs.AQCTLB.bit.ZRO = AQ_SET;
   EPwm3Regs.AQCTLB.bit.CAU = AQ_CLEAR;          // Set PWM1A on Zero
//   EPwm3Regs.AQCTLB.bit.CAD = AQ_SET;

   // Active Low PWMs - Setup Deadband
   EPwm3Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
   //EPwm3Regs.DBCTL.bit.POLSEL = DB_ACTV_LO;
   EPwm3Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
   EPwm3Regs.DBCTL.bit.IN_MODE = DBA_ALL;
   EPwm3Regs.DBRED = EPWM_MIN_DB;
   EPwm3Regs.DBFED = EPWM_MIN_DB;

// Setup Trip-Zone for ePWM1:
EALLOW;
EPwm3Regs.TZSEL.bit.OSHT1 = TZ_ENABLE; //TZ1 is a one-shot killer
EPwm3Regs.TZCTL.bit.TZA = TZ_FORCE_LO;
EPwm3Regs.TZCTL.bit.TZB = TZ_FORCE_LO;
EPwm3Regs.TZEINT.bit.OST = TZ_ENABLE;
EDIS;

   EPwm3Regs.ETSEL.bit.INTEN = 0;                // disable INT
}

void EPwm4Init()
{

   EPwm4Regs.TBPRD = 880;//374;                        // Set timer period
   EPwm4Regs.TBPHS.half.TBPHS = 528;//224;           // Phase is 0
   EPwm4Regs.TBCTR = 0x0000;                        // Clear counter

   // Setup TBCLK
   EPwm4Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;    // Count up
   EPwm4Regs.TBCTL.bit.PHSEN = TB_ENABLE;        // Disable phase loading
//   EPwm4Regs.TBCTL.bit.HSPCLKDIV = TB_DIV4;       // Clock ratio to SYSCLKOUT
```

```
//   EPwm4Regs.TBCTL.bit.CLKDIV = TB_DIV4;
   EPwm4Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Clock ratio to SYSCLKOUT
   EPwm4Regs.TBCTL.bit.CLKDIV = TB_DIV1;
   EPwm4Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;

   EPwm4Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;    // Load registers every ZERO
   EPwm4Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
   EPwm4Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
   EPwm4Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

   EALLOW;
   EPwm4Regs.HRCNFG.all = 0x0;
   EPwm4Regs.HRCNFG.bit.EDGMODE = HR_FEP; //Falling Edge Position
   EPwm4Regs.HRCNFG.bit.CTLMODE = HR_CMP; //CMPAHR controls value.
   EPwm4Regs.HRCNFG.bit.HRLOAD = HR_CTR_ZERO; //Shadow load on CTR=0
   EDIS;
   // Setup compare
   EPwm4Regs.CMPA.half.CMPA = 45;
   EPwm4Regs.CMPA.half.CMPAHR = 0x0000;
//   EPwm4Regs.CMPB = 3000;

   // Set actions
   EPwm4Regs.AQCTLA.bit.ZRO = AQ_SET;
   EPwm4Regs.AQCTLA.bit.CAU = AQ_CLEAR;             // Set PWM1A on Zero
//   EPwm4Regs.AQCTLA.bit.CAD = AQ_CLEAR;

   EPwm4Regs.AQCTLB.bit.ZRO = AQ_SET;
   EPwm4Regs.AQCTLB.bit.CAU = AQ_CLEAR;          // Set PWM1A on Zero
//   EPwm4Regs.AQCTLB.bit.CAD = AQ_SET;

   // Active Low PWMs - Setup Deadband
   EPwm4Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
   //EPwm4Regs.DBCTL.bit.POLSEL = DB_ACTV_LO;
   EPwm4Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
   EPwm4Regs.DBCTL.bit.IN_MODE = DBA_ALL;
   EPwm4Regs.DBRED = EPWM_MIN_DB;
   EPwm4Regs.DBFED = EPWM_MIN_DB;

// Setup Trip-Zone for ePWM1:
EALLOW;
EPwm4Regs.TZSEL.bit.OSHT1 = TZ_ENABLE; //TZ1 is a one-shot killer
EPwm4Regs.TZCTL.bit.TZA = TZ_FORCE_LO;
EPwm4Regs.TZCTL.bit.TZB = TZ_FORCE_LO;
EPwm4Regs.TZEINT.bit.OST = TZ_ENABLE;
EDIS;
```

```
   EPwm4Regs.ETSEL.bit.INTEN = 0;                // disable INT
}

void EPwm5Init()
{

   EPwm5Regs.TBPRD = 880;                          // Set timer period
   EPwm5Regs.TBPHS.half.TBPHS = 704;           // Phase is 0
   EPwm5Regs.TBCTR = 0x0000;                       // Clear counter

   // Setup TBCLK
   EPwm5Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;    // Count up
   EPwm5Regs.TBCTL.bit.PHSEN = TB_ENABLE;        // Disable phase loading
//   EPwm5Regs.TBCTL.bit.HSPCLKDIV = TB_DIV4;       // Clock ratio to SYSCLKOUT
//   EPwm5Regs.TBCTL.bit.CLKDIV = TB_DIV4;
   EPwm5Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Clock ratio to SYSCLKOUT
   EPwm5Regs.TBCTL.bit.CLKDIV = TB_DIV1;
   EPwm5Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;

   EPwm5Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;    // Load registers every ZERO
   EPwm5Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
   EPwm5Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
   EPwm5Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

   EALLOW;
   EPwm5Regs.HRCNFG.all = 0x0;
   EPwm5Regs.HRCNFG.bit.EDGMODE = HR_FEP; //Falling Edge Position
   EPwm5Regs.HRCNFG.bit.CTLMODE = HR_CMP; //CMPAHR controls value.
   EPwm5Regs.HRCNFG.bit.HRLOAD = HR_CTR_ZERO; //Shadow load on CTR=0
   EDIS;
   // Setup compare
   EPwm5Regs.CMPA.half.CMPA = 45;
   EPwm5Regs.CMPA.half.CMPAHR = 0x0000;
//   EPwm5Regs.CMPB = 3000;

   // Set actions
   EPwm5Regs.AQCTLA.bit.ZRO = AQ_SET;
   EPwm5Regs.AQCTLA.bit.CAU = AQ_CLEAR;             // Set PWM1A on Zero
//   EPwm5Regs.AQCTLA.bit.CAD = AQ_CLEAR;

   EPwm5Regs.AQCTLB.bit.ZRO = AQ_SET;
   EPwm5Regs.AQCTLB.bit.CAU = AQ_CLEAR;          // Set PWM1A on Zero
//   EPwm5Regs.AQCTLB.bit.CAD = AQ_SET;
```

```
   // Active Low PWMs - Setup Deadband
   EPwm5Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
   //EPwm5Regs.DBCTL.bit.POLSEL = DB_ACTV_LO;
   EPwm5Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
   EPwm5Regs.DBCTL.bit.IN_MODE = DBA_ALL;
   EPwm5Regs.DBRED = EPWM_MIN_DB;
   EPwm5Regs.DBFED = EPWM_MIN_DB;

// Setup Trip-Zone for ePWM1:
EALLOW;
EPwm5Regs.TZSEL.bit.OSHT1 = TZ_ENABLE; //TZ1 is a one-shot killer
EPwm5Regs.TZCTL.bit.TZA = TZ_FORCE_LO;
EPwm5Regs.TZCTL.bit.TZB = TZ_FORCE_LO;
EPwm5Regs.TZEINT.bit.OST = TZ_ENABLE;
EDIS;

   EPwm5Regs.ETSEL.bit.INTEN = 0;                // disable INT
}
void EPwm6Init()
{

   EPwm6Regs.TBPRD = 374;                           // Set timer period
   EPwm6Regs.TBPHS.half.TBPHS = 0x0000;             // Phase is 0
   EPwm6Regs.TBCTR = 0x0000;                        // Clear counter


   // Setup TBCLK
   EPwm6Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Count up
   EPwm6Regs.TBCTL.bit.PHSEN = TB_DISABLE;        // Disable phase loading
   EPwm6Regs.TBCTL.bit.HSPCLKDIV = TB_DIV4;       // Clock ratio to SYSCLKOUT
   EPwm6Regs.TBCTL.bit.CLKDIV = TB_DIV4;          // Slow so we can observe on the scope

   // Setup compare
   EPwm6Regs.CMPA.half.CMPA = 45;
   EPwm6Regs.CMPA.half.CMPAHR = 0x0000;

   // Set actions
   EPwm6Regs.AQCTLA.bit.CAU = AQ_SET;              // Set PWM3A on Zero
   EPwm6Regs.AQCTLA.bit.CAD = AQ_CLEAR;

   EPwm6Regs.AQCTLB.bit.CAU = AQ_CLEAR;            // Set PWM3A on Zero
   EPwm6Regs.AQCTLB.bit.CAD = AQ_SET;

   // Active high complementary PWMs - Setup the deadband
   EPwm6Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
```

```
   EPwm6Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
   EPwm6Regs.DBCTL.bit.IN_MODE = DBA_ALL;
   EPwm6Regs.DBRED = EPWM_MIN_DB;
   EPwm6Regs.DBFED = EPWM_MIN_DB;

   // Interrupt where we will change the deadband
//   EPwm6Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO;       // Select INT on Zero event
//   EPwm6Regs.ETSEL.bit.INTEN = 1;                  // Enable INT
//   EPwm6Regs.ETPS.bit.INTPRD = ET_3RD;             // Generate INT on 3rd event
   EPwm6Regs.ETSEL.bit.INTEN = 0;                  // Disable INT


    //PWM Trigger
    EPwm6Regs.ETSEL.bit.SOCAEN = 1;
    EPwm6Regs.ETSEL.bit.SOCASEL = 4;
    EPwm6Regs.ETPS.bit.SOCAPRD = 1;
    EPwm6Regs.CMPA.half.CMPA = 0x0080;
    EPwm6Regs.TBPRD = PRD400KHZ;
    EPwm6Regs.TBCTL.bit.CTRMODE = 0;
}

void EnableAllPWM(void){
EPwm1Regs.TZCLR.all = 5;
EPwm2Regs.TZCLR.all = 5;
EPwm3Regs.TZCLR.all = 5;
EPwm4Regs.TZCLR.all = 5;
EPwm5Regs.TZCLR.all = 5;
EPwm6Regs.TZCLR.all = 5;
PhaseStatus.all = PhaseStatus.all | ALLPHASESON;
}
void DisableAllPWM(void){
EPwm1Regs.TZFRC.bit.OST = 1; //Kill PWM1
EPwm2Regs.TZFRC.bit.OST = 1; //Kill PWM2
EPwm3Regs.TZFRC.bit.OST = 1; //Kill PWM3
EPwm4Regs.TZFRC.bit.OST = 1; //Kill PWM4
EPwm5Regs.TZFRC.bit.OST = 1; //Kill PWM5
EPwm6Regs.TZFRC.bit.OST = 1; //Kill PWM6
PhaseStatus.all = PhaseStatus.all & ALLPHASESOFF;
}

Tasks.h
#ifndef TASKS_H_
#define TASKS_H_

#include "Globals.h"
```

```
void TaskMachineInit(void);

// State Machine function prototypes
//-----------------------------------
// Alpha states
void A0(void); //state A0
void B0(void); //state B0
void C0(void); //state C0

// A branch states
void A1(void); //state A1
void A2(void); //state A2

// B branch states
void B1(void); //state B1
void B2(void); //state B2

// C branch states
void C1(void); //state C1
void C2(void); //state C2

// Variable declarations
extern void (*Alpha_State_Ptr)(void); // Base States pointer
extern void (*A_Task_Ptr)(void); // State pointer A branch
extern void (*B_Task_Ptr)(void); // State pointer B branch
extern void (*C_Task_Ptr)(void); // State pointer C branch


extern int16 SerialCommsTimer;
extern int16 CommsOKflg;
extern char* msg;

extern int16 Gui_LedPrd_ms; // LED Prd in ms
extern int16 LedBlinkTimer;

extern Uint16 voutRef;
extern Uint16 epsilon;
extern Uint16 epsilonSmall;

#endif /*TASKS_H_*/
```

Tasks.c

```
#include "PeripheralHeaderIncludes_F28335.h"
#include "Tasks.h"
```

```
#include "Globals.h"
#include "ADCHandler.h"
#include "SerialHandler.h"
#include "PwmHandler.h"
//===========================================================================
// STATE-MACHINE SEQUENCING AND SYNCRONIZATION
//===========================================================================

#pragma CODE_SECTION(A0, "ramfuncs");
#pragma CODE_SECTION(A1, "ramfuncs");
#pragma CODE_SECTION(B0, "ramfuncs");
#pragma CODE_SECTION(B1, "ramfuncs");
#pragma CODE_SECTION(C0, "ramfuncs");
#pragma CODE_SECTION(C1, "ramfuncs");


void (*Alpha_State_Ptr)(void); // Base States pointer
void (*A_Task_Ptr)(void); // State pointer A branch
void (*B_Task_Ptr)(void); // State pointer B branch
void (*C_Task_Ptr)(void); // State pointer C branch

//Uint16 PhaseStatus;
union LEDS LEDStatus;
union LEDS LEDStatus2;

union MAINFLAGS MainFlags;
union BUTTONS ButtonStatus;
union PHASEFLAGS PhaseStatus;

int16 SerialCommsTimer;
int16 CommsOKflg;
char* msg;

int16 Gui_LedPrd_ms; // LED Prd in ms
int16 LedBlinkTimer;

Uint16 voutRef;
Uint16 epsilon;
Uint16 epsilonSmall;
Uint16 controlCounter;
Uint8 start;

//PID terms
Uint16 Kp;
Uint16 Ki;
```

```
Uint16 Kd;
Uint16 error;
Uint16 integral;
Uint16 derivative;
Uint16 previous_error;
Uint16 output;
Uint16 value;
//Unit16 enter_I;


void TaskMachineInit(void){
Alpha_State_Ptr = &A0;
A_Task_Ptr = &A1;
B_Task_Ptr = &B1;
C_Task_Ptr = &C1;
voutRef = 1358; //0x054E
epsilon = 30;
epsilonSmall = 2;
PhaseStatus.bit.Controller = 0;
PhaseStatus.all = PHASE1ADCON | PHASE2ADCON;
MainFlags.bit.AdcTxFlag = 0;
start = 0;
controlCounter = 0;
Kp = 1;
Ki = 10;
Kd = 1;
// dt = .001;
integral = 0;
derivative = 0;
error = 0;
previous_error = 0;
output = 0;
value = 100;
}

//--------------------------------- FRAMEWORK -----------------------------------
void A0(void)
{
// loop rate synchronizer for A-tasks
if(CpuTimer0Regs.TCR.bit.TIF == 1)
{
CpuTimer0Regs.TCR.bit.TIF = 1; // clear flag

//-----------------------------------------------------------
(*A_Task_Ptr)(); // jump to an A Task (A1,A2,A3,...)
```

```
//-----------------------------------------------------------

SerialCommsTimer++;
}



Alpha_State_Ptr = &B0; // Comment out to allow only A tasks
}

void B0(void)
{
// loop rate synchronizer for B-tasks
if(CpuTimer1Regs.TCR.bit.TIF == 1)
{
CpuTimer1Regs.TCR.bit.TIF = 1; // clear flag

//-----------------------------------------------------------
(*B_Task_Ptr)(); // jump to a B Task (B1,B2,B3,...)
//-----------------------------------------------------------
}

Alpha_State_Ptr = &C0; // Allow C state tasks
}

void C0(void)
{
// loop rate synchronizer for C-tasks
if(CpuTimer2Regs.TCR.bit.TIF == 1)
{
CpuTimer2Regs.TCR.bit.TIF = 1; // clear flag

//-----------------------------------------------------------
(*C_Task_Ptr)(); // jump to a C Task (C1,C2,C3,...)
//-----------------------------------------------------------
}

Alpha_State_Ptr = &A0; // Back to State A0
}


//--------------------------------- USER -----------------------------------------

//===========================================================================
// A - TASKS
```

```
//===========================================================================
// Each active task runs every (CpuTimer0 period) * (# of active A Tasks)
//---------------------------------------------------------
void A1(void) // SCI-GUI
//---------------------------------------------------------
{
// SerialHostComms(); // Serialport controls LED2 (GPIO-31)
// Will not blink until GUI is connected

if(LedBlinkTimer == 0)
{
GpioDataRegs.GPBTOGGLE.bit.GPIO34 = 1; // toggle GPIO34 which controls LD3 on most contr
#if DSP2834x_DEVICE_H
GpioDataRegs.GPBTOGGLE.bit.GPIO54 = 1; // toggle GPIO54 which controls LD3 on the C2834x
#endif

GpioDataRegs.GPATOGGLE.bit.GPIO31 = 1; // toggle GPIO34 which controls LD3 on most contr
LedBlinkTimer = Gui_LedPrd_ms >> 1; // divide by 2 (one LED blink period is 2 LED toggle
}
else
{
LedBlinkTimer--;
}


//-------------------
//the next time CpuTimer0 'counter' reaches Period value go to A1
A_Task_Ptr = &A1;
//-------------------
}

//---------------------------------
void A2(void) //  SPARE
//---------------------------------
{

//-------------------
//task never runs; Task A1 always never set A_Task_Ptr to come to A2
A_Task_Ptr = &A1;
//-------------------
}


//===========================================================================
// B - TASKS
```

```
//===========================================================================
// Each active task runs every (CpuTimer1 period) * (# of active B Tasks)
//---------------------------------
void B1(void) //  SPARE
//---------------------------------
{
if(PhaseStatus.bit.Controller){
start = 1;
//PID
//error calc
error = (voutRef - SampleTable[0]);
integral = integral + error*(0.031);
derivative = (error - previous_error)/0.0031;
// output = Kp*error;                  //P-Controller
// output = Kp*error + Ki*integral;  //PI-Controller
output = (Kp*error + Ki*integral + Kd*derivative); //PID-Controller
output = output>>6;
previous_error = error;

//write output to EPwm 1-5 registers CMPA large (upper) CMPAHR is high resolution (lower
if(SampleTable[0] > (voutRef+epsilon)){
//possible scaling for output
EPwm1Regs.CMPA.half.CMPAHR=0;
EPwm2Regs.CMPA.half.CMPAHR=0;
EPwm3Regs.CMPA.half.CMPAHR=0;
EPwm4Regs.CMPA.half.CMPAHR=0;
EPwm5Regs.CMPA.half.CMPAHR=0;
EPwm1Regs.CMPA.half.CMPA-=output;
EPwm2Regs.CMPA.half.CMPA-=output;
EPwm3Regs.CMPA.half.CMPA-=output;
EPwm4Regs.CMPA.half.CMPA-=output;
EPwm5Regs.CMPA.half.CMPA-=output;
}
else if(SampleTable[0] < (voutRef+epsilon)){
EPwm1Regs.CMPA.half.CMPAHR=0;
EPwm2Regs.CMPA.half.CMPAHR=0;
EPwm3Regs.CMPA.half.CMPAHR=0;
EPwm4Regs.CMPA.half.CMPAHR=0;
EPwm5Regs.CMPA.half.CMPAHR=0;
EPwm1Regs.CMPA.half.CMPA+=output;
EPwm2Regs.CMPA.half.CMPA+=output;
EPwm3Regs.CMPA.half.CMPA+=output;
EPwm4Regs.CMPA.half.CMPA+=output;
EPwm5Regs.CMPA.half.CMPA+=output;
}
```

```
else{ //Now in range of approximately. Fine tune incramental control
if(SampleTable[0] > (voutRef+epsilonSmall)){
if(EPwm1Regs.CMPA.half.CMPAHR != 0){
EPwm1Regs.CMPA.half.CMPAHR -= 0x0100;
EPwm2Regs.CMPA.half.CMPAHR -= 0x0100;
EPwm3Regs.CMPA.half.CMPAHR -= 0x0100;
EPwm4Regs.CMPA.half.CMPAHR -= 0x0100;
EPwm5Regs.CMPA.half.CMPAHR -= 0x0100;
}
else{
EPwm1Regs.CMPA.half.CMPA--;
EPwm2Regs.CMPA.half.CMPA--;
EPwm3Regs.CMPA.half.CMPA--;
EPwm4Regs.CMPA.half.CMPA--;
EPwm5Regs.CMPA.half.CMPA--;
EPwm1Regs.CMPA.half.CMPAHR = 0x2000;
EPwm2Regs.CMPA.half.CMPAHR = 0x2000;
EPwm3Regs.CMPA.half.CMPAHR = 0x2000;
EPwm4Regs.CMPA.half.CMPAHR = 0x2000;
EPwm5Regs.CMPA.half.CMPAHR = 0x2000;
}
}else if (SampleTable[0] < (voutRef-epsilonSmall)){
if(EPwm1Regs.CMPA.half.CMPAHR < 0x4000){
EPwm1Regs.CMPA.half.CMPAHR += 0x0100;
EPwm2Regs.CMPA.half.CMPAHR += 0x0100;
EPwm3Regs.CMPA.half.CMPAHR += 0x0100;
EPwm4Regs.CMPA.half.CMPAHR += 0x0100;
EPwm5Regs.CMPA.half.CMPAHR += 0x0100;
}
else{
EPwm1Regs.CMPA.half.CMPA++;
EPwm2Regs.CMPA.half.CMPA++;
EPwm3Regs.CMPA.half.CMPA++;
EPwm4Regs.CMPA.half.CMPA++;
EPwm5Regs.CMPA.half.CMPA++;
EPwm1Regs.CMPA.half.CMPAHR = 0x2000;
EPwm2Regs.CMPA.half.CMPAHR = 0x2000;
EPwm3Regs.CMPA.half.CMPAHR = 0x2000;
EPwm4Regs.CMPA.half.CMPAHR = 0x2000;
EPwm5Regs.CMPA.half.CMPAHR = 0x2000;
}
}
}

}
```

```
//}

//-----------------
//the next time CpuTimer1 'counter' reaches Period value go to B2
B_Task_Ptr = &B1;
//-----------------
}

//---------------------------------
void B2(void) //  SPARE
//---------------------------------
{

//-----------------
//the next time CpuTimer1 'counter' reaches Period value go to B1
B_Task_Ptr = &B1;
//-----------------
}


//===========================================================================
// C - TASKS
//===========================================================================
// Each active task runs every (CpuTimer2 period) * (# of active C Tasks)
//---------------------------------
void C1(void)  // SPARE
//---------------------------------
{
// if(start == 0){
// controlCounter++;
// if(controlCounter > 10){
// PhaseStatus.bit.Controller = 1;
// EnableAllPWM();
// }
// }
if(MainFlags.bit.AdcTxFlag == 1){
scia_xmit_async('\n');
scia_xmit_async('\r');
scia_xmit_num_async(SampleTable[0]);
scia_xmit_async('-');
scia_xmit_num_async(SampleTable[1]);
scia_xmit_async('|');
scia_xmit_num_async(SampleTable[2]);
scia_xmit_async('-');
scia_xmit_num_async(SampleTable[3]);
```

```
scia_xmit_async('|');
scia_xmit_num_async(SampleTable[4]);
scia_xmit_async('-');
scia_xmit_num_async(SampleTable[5]);
scia_xmit_async('|');
scia_xmit_num_async(SampleTable[6]);
scia_xmit_async('-');
scia_xmit_num_async(SampleTable[7]);
scia_xmit_async('|');
scia_xmit_num_async(SampleTable[8]);
scia_xmit_async('-');
scia_xmit_num_async(SampleTable[9]);
}
//-----------------
//the next time CpuTimer2 'counter' reaches Period value go to C2
C_Task_Ptr = &C1;
//-----------------

}

//-------------------------------
void C2(void) //  SPARE
//-------------------------------
{

//-----------------
//the next time CpuTimer2 'counter' reaches Period value go to C1
C_Task_Ptr = &C1;
//-----------------
}

SerialHandler.h
#ifndef SERIALHANDLER_H_
#define SERIALHANDLER_H_
#include "PeripheralHeaderIncludes_F28335.h"     // Device Headerfile and Examples Inclu

#define TXBUFFSIZE 256
//Union and structure definitions
struct SCIISRBITS{        // bits    description
Uint16 TXFlag:1;     // 0     TX Interrupt Occured
Uint16 RXFlag:1;      // 1     RX Interrupt Occured
Uint16  unassigned:14;
};

union SCI_ISR_FLAGS{
```

```
Uint16 all;
struct SCIISRBITS bit;
};

struct SCIFLAGBITS{        // bits    description
Uint16 UpdateDuty:1;     // 0     Update Duty Cycle Flag
Uint16  UpdateHRDuty:1;  // 1 Update HR Duty cycle flag
Uint16 UpdateDBRising:1;       // 2     Update Deadband rising flag
Uint16 UpdateDBFalling:1; // 3 Update deadband falling flag
Uint16  UpdatePeriod:1;  // 4 Update Period
Uint16  KillPWM:1; // 5 Kill PWM
Uint16  UpdatePhase2:1; // 6 Update Phase 2
Uint16  UpdatePhase3:1; // 7 Update Phase 3
Uint16 UpdatePhase4:1; // 8 Update Phase 4
Uint16 UpdatePhase5:1; // 9 Update Phase 5
Uint16  SendEverything:1; // 10 Send Complete Status
Uint16  UpdateVoutRef:1; // 11  Modify Vout
Uint16  UpdateEpsilon:1; // 12  Modify epsilon
Uint16  UpdateEpsilonSmall:1;   // 13 Modify epsilonSmall
Uint16  unassigned:2;
};

union SCIFLAGS{
Uint16 all;
struct SCIFLAGBITS bit;
};


//Global Scope variables
extern union SCI_ISR_FLAGS SciIsrFlags;
extern union SCIFLAGS SciFlags;

extern Uint16 ReceivedChar;
//extern Uint16 XmitFlag;

//Async TXBuffer FIFO;
extern Uint16 TXBuffer[TXBUFFSIZE];
extern Uint16* TXBufferFIFOFront;
extern Uint16* TXBufferFIFOEnd;
extern Uint16* TXBufferEnd;

interrupt void scirxa_isr(void);
interrupt void scitxa_isr(void);

void scia_echoback_init(void);
```

```
void scia_fifo_init(void);
void scia_xmit(int a);
void scia_msg(char *msg);
void scia_xmit_num(Uint16 val);

void scia_fifo_init_async(void);
void scia_xmit_async(int a);
void scia_msg_async(char *msg);
void scia_xmit_num_async(Uint16 val);

void InitSciaGpio();

#endif /*SERIALHANDLER_H_*/
```

SerialHandler.c

```
#include "SerialHandler.h"
#include "PWMHandler.h"
#include "Globals.h"
#include "PeripheralHeaderIncludes_F28335.h"

union SCI_ISR_FLAGS SciIsrFlags;
union SCIFLAGS SciFlags;

Uint16 ReceivedChar;
//#pragma DATA_SECTION(TXBuffer, "scitxbuffer")
Uint16 TXBuffer[TXBUFFSIZE];
Uint16* TXBufferFIFOFront;
Uint16* TXBufferFIFOEnd;
Uint16* TXBufferEnd;

#pragma CODE_SECTION(scirxa_isr, "ramfuncs");
#pragma CODE_SECTION(scitxa_isr, "ramfuncs");

interrupt void scirxa_isr(void){
// char* msg;
static Uint16 value;
EALLOW;
ReceivedChar = SciaRegs.SCIRXBUF.all;
switch(ReceivedChar){
case 'x':
EPwm1Regs.TZFRC.bit.OST = 1; //Kill PWM1
EPwm2Regs.TZFRC.bit.OST = 1; //Kill PWM2
EPwm3Regs.TZFRC.bit.OST = 1; //Kill PWM3
EPwm4Regs.TZFRC.bit.OST = 1; //Kill PWM4
EPwm5Regs.TZFRC.bit.OST = 1; //Kill PWM5
```

```
EPwm6Regs.TZFRC.bit.OST = 1; //Kill PWM6
PhaseStatus.all = PhaseStatus.all & ALLPHASESOFF;
SciFlags.bit.KillPWM = 1;
break;
case '0':
EPwm1Regs.TZCLR.all = 5;
EPwm2Regs.TZCLR.all = 5;
EPwm3Regs.TZCLR.all = 5;
EPwm4Regs.TZCLR.all = 5;
EPwm5Regs.TZCLR.all = 5;
EPwm6Regs.TZCLR.all = 5;
PhaseStatus.all = PhaseStatus.all | ALLPHASESON;
break;
case '1':
EPwm1Regs.TZCLR.all = 5;
PhaseStatus.bit.Phase1 = 1;
PhaseStatus.bit.Phase1ADC = 1;
break;
case '2':
EPwm2Regs.TZCLR.all = 5;
PhaseStatus.bit.Phase2 = 1;
PhaseStatus.bit.Phase2ADC = 1;
break;
case '3':
EPwm3Regs.TZCLR.all = 5;
PhaseStatus.bit.Phase3 = 1;
PhaseStatus.bit.Phase3ADC = 1;
break;
case '4':
EPwm4Regs.TZCLR.all = 5;
PhaseStatus.bit.Phase4 = 1;
PhaseStatus.bit.Phase4ADC = 1;
break;
case '5': //Phase 5
EPwm5Regs.TZCLR.all = 5;
PhaseStatus.bit.Phase5 = 1;
PhaseStatus.bit.Phase5ADC = 1;
break;
case 'i':
value = EPwm1Regs.CMPA.half.CMPA + 1;
SciFlags.bit.UpdateDuty = 1;
break;
case 'k':
value = EPwm1Regs.CMPA.half.CMPA - 1;
SciFlags.bit.UpdateDuty = 1;
```

```
break;
case '.':
value = EPwm1Regs.CMPA.half.CMPAHR + 0x100;
SciFlags.bit.UpdateHRDuty = 1;
break;
case ',':
value = EPwm1Regs.CMPA.half.CMPAHR - 0x100;
SciFlags.bit.UpdateHRDuty = 1;
break;
case 'y':
value = voutRef + 1;
SciFlags.bit.UpdateVoutRef = 1;
break;
case 'h':
value = voutRef - 1;
SciFlags.bit.UpdateVoutRef = 1;
break;
case 't':
value = epsilon + 1;
SciFlags.bit.UpdateEpsilon = 1;
break;
case 'g':
value = epsilon - 1;
SciFlags.bit.UpdateEpsilon = 1;
break;
case 'r':
value = epsilonSmall + 1;
SciFlags.bit.UpdateEpsilonSmall = 1;
break;
case 'f':
value = epsilonSmall - 1;
SciFlags.bit.UpdateEpsilonSmall = 1;
break;
case 'n':
if(--PwmPointer < PwmPeriods)
PwmPointer = PwmPeriods + 13;
value = *PwmPointer;
SciFlags.bit.UpdatePeriod = 1;
break;
case 'm':
if(++PwmPointer > PwmPeriods+13)
PwmPointer = PwmPeriods;
value = *PwmPointer;
SciFlags.bit.UpdatePeriod = 1;
break;
```

```
case 's': //status
SciFlags.bit.SendEverything = 1;
break;
case 'p':
value = 0x10;
EPwm1Regs.CMPA.half.CMPA = value;
EPwm2Regs.CMPA.half.CMPA = value;
EPwm3Regs.CMPA.half.CMPA = value;
EPwm4Regs.CMPA.half.CMPA = value;
EPwm5Regs.CMPA.half.CMPA = value;
EPwm6Regs.CMPA.half.CMPA = value;
value = 0;
EPwm1Regs.CMPA.half.CMPAHR = value;
EPwm2Regs.CMPA.half.CMPAHR = value;
EPwm3Regs.CMPA.half.CMPAHR = value;
EPwm4Regs.CMPA.half.CMPAHR = value;
EPwm5Regs.CMPA.half.CMPAHR = value;
EPwm6Regs.CMPA.half.CMPAHR = value;
break;
case 'v':
PhaseStatus.bit.Controller = 1;
break;
case 'V':
PhaseStatus.bit.Controller = 0;
break;
case 'q':
MainFlags.bit.AdcTxFlag = 0;
break;
case 'Q':
MainFlags.bit.AdcTxFlag = 1;
}
scia_xmit_async(ReceivedChar);
if(SciFlags.bit.KillPWM == 1){
SciFlags.bit.KillPWM = 0;
}
if(SciFlags.bit.UpdateDuty == 1){
EPwm1Regs.CMPA.half.CMPA = value;
EPwm2Regs.CMPA.half.CMPA = value;
EPwm3Regs.CMPA.half.CMPA = value;
EPwm4Regs.CMPA.half.CMPA = value;
EPwm5Regs.CMPA.half.CMPA = value;
EPwm6Regs.CMPA.half.CMPA = value;
     scia_xmit_num_async(value);
     SciFlags.bit.UpdateDuty = 0;
    }
```

```
    if(SciFlags.bit.UpdateHRDuty == 1){
EPwm1Regs.CMPA.half.CMPAHR = value;
EPwm2Regs.CMPA.half.CMPAHR = value;
EPwm3Regs.CMPA.half.CMPAHR = value;
EPwm4Regs.CMPA.half.CMPAHR = value;
EPwm5Regs.CMPA.half.CMPAHR = value;
EPwm6Regs.CMPA.half.CMPAHR = value;
     scia_xmit_num_async(value);
     SciFlags.bit.UpdateHRDuty = 0;
    }
if(SciFlags.bit.UpdateDBRising == 1){
EPwm1Regs.DBRED = value;
EPwm2Regs.DBRED = value;
EPwm3Regs.DBRED = value;
EPwm4Regs.DBRED = value;
EPwm5Regs.DBRED = value;
EPwm6Regs.DBRED = value;
     scia_xmit_num_async(value);
     SciFlags.bit.UpdateDBRising = 0;
    }
if(SciFlags.bit.UpdateDBFalling == 1){
EPwm1Regs.DBFED = value;
EPwm2Regs.DBFED = value;
EPwm3Regs.DBFED = value;
EPwm4Regs.DBFED = value;
EPwm5Regs.DBFED = value;
     scia_xmit_num_async(value);
     SciFlags.bit.UpdateDBFalling = 0;
}
if(SciFlags.bit.UpdatePeriod == 1){
EPwm1Regs.TBPRD = value;
EPwm2Regs.TBPRD = value;
EPwm3Regs.TBPRD = value;
EPwm4Regs.TBPRD = value;
EPwm5Regs.TBPRD = value;
EPwm6Regs.TBPRD = value;
scia_xmit_num_async(value);
SciFlags.bit.UpdatePeriod = 0;
}
if(SciFlags.bit.SendEverything == 1){
scia_xmit_async('\n');
scia_xmit_async('\r');
scia_xmit_num_async(EPwm1Regs.TBPRD);
scia_xmit_async('-');
scia_xmit_num_async(EPwm1Regs.CMPA.half.CMPA);
```

```
scia_xmit_async('-');
scia_xmit_num_async(EPwm1Regs.DBRED);
scia_xmit_async('-');
scia_xmit_num_async(EPwm1Regs.DBFED);
scia_xmit_async('-');
scia_xmit_num_async(EPwm1Regs.TZFRC.all);
scia_xmit_async('|');
scia_xmit_num_async(EPwm2Regs.TZFRC.all);
scia_xmit_async('-');
scia_xmit_num_async(EPwm2Regs.TBPHS.half.TBPHS);
scia_xmit_async('|');
scia_xmit_num_async(EPwm3Regs.TZFRC.all);
scia_xmit_async('-');
scia_xmit_num_async(EPwm3Regs.TBPHS.half.TBPHS);
scia_xmit_async('|');
scia_xmit_num_async(EPwm4Regs.TZFRC.all);
scia_xmit_async('-');
scia_xmit_num_async(EPwm4Regs.TBPHS.half.TBPHS);
scia_xmit_async('|');
scia_xmit_num_async(EPwm5Regs.TZFRC.all);
scia_xmit_async('-');
scia_xmit_num_async(EPwm5Regs.TBPHS.half.TBPHS);
scia_xmit_async('|');
scia_xmit_num_async(AdcRegs.ADCRESULT0);
scia_xmit_async('-');
scia_xmit_num_async(AdcRegs.ADCRESULT1);
scia_xmit_async('-');
scia_xmit_num_async(AdcRegs.ADCRESULT2);
scia_xmit_async('-');
scia_xmit_num_async(AdcRegs.ADCRESULT3);
SciFlags.bit.SendEverything = 0;
}
if(SciFlags.bit.UpdateVoutRef == 1){
voutRef = value;
scia_xmit_num_async(value);
SciFlags.bit.UpdateVoutRef = 0;
}
if(SciFlags.bit.UpdateEpsilon == 1){
epsilon = value;
scia_xmit_num_async(value);
SciFlags.bit.UpdateEpsilon = 0;
}
if(SciFlags.bit.UpdateEpsilonSmall == 1){
epsilonSmall = value;
scia_xmit_num_async(value);
```

```
SciFlags.bit.UpdateEpsilonSmall = 0;
}
    SciIsrFlags.bit.RXFlag = 1;
    SciaRegs.SCIFFRX.bit.RXFFINTCLR = 1; // Reset interrupt.
PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;   // Acknowledge interrupt to PIE
EDIS;
}



// Test 1,SCIA  DLB, 8-bit word, baud rate 0x000F, default, 1 STOP bit, no parity
void scia_echoback_init()
{
    // Note: Clocks were turned on to the SCIA peripheral
    // in the InitSysCtrl() function

  SciaRegs.SCICCR.all =0x0007;   // 1 stop bit,  No loopback
                                 // No parity,8 char bits,
                                 // async mode, idle-line protocol
SciaRegs.SCICTL1.all =0x0003;  // enable TX, RX, internal SCICLK,
                                 // Disable RX ERR, SLEEP, TXWAKE
SciaRegs.SCICTL2.all =0x0003;
SciaRegs.SCICTL2.bit.TXINTENA =1;
// SciaRegs.SCICTL2.bit.RXBKINTENA =1;
SciaRegs.SCIFFRX.bit.RXFFIENA = 1;

// #if (CPU_FRQ_150MHZ)
SciaRegs.SCIHBAUD    =0x0001;  // 9600 baud @LSPCLK = 37.5MHz.
SciaRegs.SCILBAUD    =0x00E7;
// #endif
// #if (CPU_FRQ_100MHZ)
//      SciaRegs.SCIHBAUD    =0x0001;  // 9600 baud @LSPCLK = 20MHz.
//      SciaRegs.SCILBAUD    =0x0044;
// #endif
SciaRegs.SCICTL1.all =0x0023;  // Relinquish SCI from Reset
}

// Transmit a character from the SCI
void scia_xmit(int a)
{
    while (SciaRegs.SCIFFTX.bit.TXFFST != 0) {}
    SciaRegs.SCITXBUF=a;

}
```

```
void scia_msg(char * msg)
{
    int i;
    i = 0;
    while(msg[i] != '\0')
    {
        scia_xmit(msg[i]);
        i++;
    }
}

// Initalize the SCI FIFO
void scia_fifo_init()
{
    SciaRegs.SCIFFTX.all=0xE040;
//    SciaRegs.SCIFFRX.all=0x204f;
    SciaRegs.SCIFFRX.all=0x2061; //1 byte fifo generates interrupt
//    SciaRegs.SCIFFRX.all=0x2071;
    SciaRegs.SCIFFCT.all=0x0;

}

void scia_xmit_num(Uint16 val){
int nibble = val & 0x000F;
int ascii[4];
int counter = 0;
for(counter = 0; counter < 4; counter++){
if(nibble > 9)
ascii[3-counter] = nibble+55;
else
ascii[3-counter] = nibble+48;
val = val >> 4;
nibble = val &0x000F;
}
scia_xmit(ascii[0]);
scia_xmit(ascii[1]);
scia_xmit(ascii[2]);
scia_xmit(ascii[3]);
}


interrupt void scitxa_isr(void){
Uint16 i;
//Interrupt occured, meaning TX FIFO empty
//Check if TXBuffer is empty. If so, disable interrupts and finish out.
```

```
if(TXBufferFIFOFront == TXBufferFIFOEnd){
SciaRegs.SCIFFTX.bit.TXFFIENA = 0; //disable interrupts
}
else{ //Data in TXBuffer
for(i = 0; i < 15; i++)
{
SciaRegs.SCITXBUF=*TXBufferFIFOFront;
TXBufferFIFOFront++;
if(TXBufferFIFOFront > TXBufferEnd)
TXBufferFIFOFront = TXBuffer;
if(TXBufferFIFOFront == TXBufferFIFOEnd)
break;
}
}
SciIsrFlags.bit.TXFlag = 1;
SciaRegs.SCIFFTX.bit.TXFFINTCLR = 1; // Reset interrupt.
PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;   // Acknowledge interrupt to PIE
}


// Initalize the SCI FIFO
void scia_fifo_init_async()
{
    SciaRegs.SCIFFTX.all=0xE040; //Empty TX generates interrupt, but don't enable the in
    SciaRegs.SCIFFRX.all=0x2061; //1 byte fifo generates interrupt
    SciaRegs.SCIFFCT.all=0x0;
TXBufferFIFOFront = TXBuffer;
TXBufferFIFOEnd = TXBuffer;
TXBufferEnd = TXBuffer + (TXBUFFSIZE-1)*sizeof(Uint16); // the -1 is there because TXBuf
}

// Transmit a character from the SCI
void scia_xmit_async(int a)
{
    //while (SciaRegs.SCIFFTX.bit.TXFFST != 0) {}
    *TXBufferFIFOEnd++ = a; //Store a in TXBufferFIFOEnd and then increment TXBufferFIFO
//    TXBufferFIFOEnd++;
    if(TXBufferFIFOEnd > TXBufferEnd)
     TXBufferFIFOEnd = TXBuffer;
//    SciaRegs.SCITXBUF=a;
SciaRegs.SCIFFTX.bit.TXFFIENA = 1;
}

void scia_msg_async(char * msg)
{
```

```
    int i;
    i = 0;
    while(msg[i] != '\0')
    {
        scia_xmit_async(msg[i]);
        i++;
    }
}



void scia_xmit_num_async(Uint16 val){
int nibble = val & 0x000F;
int ascii[4];
int counter = 0;
for(counter = 0; counter < 4; counter++){
if(nibble > 9)
ascii[3-counter] = nibble+55;
else
ascii[3-counter] = nibble+48;
val = val >> 4;
nibble = val &0x000F;
}
scia_xmit_async(ascii[0]);
scia_xmit_async(ascii[1]);
scia_xmit_async(ascii[2]);
scia_xmit_async(ascii[3]);
}


void InitSciaGpio()
{
   EALLOW;

GpioCtrlRegs.GPAPUD.bit.GPIO28 = 0;    // Enable pull-up for GPIO28 (SCIRXDA)
GpioCtrlRegs.GPAPUD.bit.GPIO29 = 0;    // Enable pull-up for GPIO29 (SCITXDA)

/* Set qualification for selected pins to asynch only */
// Inputs are synchronized to SYSCLKOUT by default.
// This will select asynch (no qualification) for the selected pins.

GpioCtrlRegs.GPAQSEL2.bit.GPIO28 = 3;  // Asynch input GPIO28 (SCIRXDA)

/* Configure SCI-A pins using GPIO regs*/
// This specifies which of the possible GPIO pins will be SCI functional pins.
```

```
GpioCtrlRegs.GPAMUX2.bit.GPIO28 = 1;   // Configure GPIO28 for SCIRXDA operation
GpioCtrlRegs.GPAMUX2.bit.GPIO29 = 1;   // Configure GPIO29 for SCITXDA operation

    EDIS;
}
```

I2CHandler.h

```
#ifndef I2CHANDLER_H_
#define I2CHANDLER_H_

#include "PeripheralHeaderIncludes_F28335.h"
//#include "DSP28x_Project.h"     // Device Headerfile and Examples Include File

#define I2C_PCA0_ADDR        0x60 //0xC0
#define I2C_PCA1_ADDR        0x61 //0xC1
#define I2C_PCA2_ADDR        0x62 //0xC2
#define I2C_PCA3_ADDR        0x63 //0xC3

#define I2C_PCA_CTL_INPUT 0x00
#define I2C_PCA_CTL_PSC0 0x01
#define I2C_PCA_CTL_PWM0 0x02
#define I2C_PCA_CTL_PSC1 0x03
#define I2C_PCA_CTL_PWM1 0x04
#define I2C_PCA_CTL_LS0 0x05
#define I2C_PCA_CTL_LS1 0x06

#define I2C_PCA_CTL_AUTO 0x10;

#define LCD_OFF  0x00;
#define LCD_ON 0x01;
#define LCD_BLINK1 0x02;
#define LCD_BLINK2  0x03;

#define I2C_NUMBYTES          4
#define I2C_EEPROM_HIGH_ADDR  0x00
#define I2C_EEPROM_LOW_ADDR   0x30

#define I2C_TX 0x00
#define I2C_RX 0x01

#define I2CBUFFSIZE 16
//extern Uint16 I2CTXFlag;
//extern Uint16 I2CRXFlag;
```

```
struct I2C_MSG{
Uint16 SlaveAddress;
Uint16 CommandByte;
Uint16 MsgStatus;
Uint16 NumOfBytes;
// void  (*CallbackFunc)(struct I2C_MSG* msg);
Uint16 MsgBuffer[8]; //Should not need full buffer size
// Uint16 MsgBuffer[I2C_MAX_BUFFER_SIZE];
};

extern struct I2C_MSG I2cMsgOut1;

extern struct I2C_MSG I2cMsgIn1;

void I2CA_Init();
//void   InitI2CModule(void);
void  InitPCAs();
Uint16 PollButtons(); //Determine which phases are inserted into the carrier.
Uint16 PollPhases(); //Determine which phases are inserted into the carrier.
void  SetPCA1Status(Uint16 value); //Set PCA1 LED Status. 2 bits per LED:
void  SetPCA3Status(Uint16 value); //Set PCA1 LED Status. 2 bits per LED:

Uint16 I2CA_WriteData(struct I2C_MSG *msg);
Uint16 I2CA_ReadData(struct I2C_MSG *msg);
Uint16 I2CA_ReadDataBlocking(struct I2C_MSG *msg);
//Uint16 I2CA_ReadDataBlocking(struct I2C_MSG *msg);
interrupt void i2c_int1a_isr(void);


//Uint16 I2CA_WriteDataAsync(struct I2C_MSG *msg);

#endif /*I2CHANDLER_H_*/

I2CHandler.c
#include "I2CHandler.h"
#include "SerialHandler.h"
#include "DSP28x_Project.h"     // Device Headerfile and Examples Include File
//
//
struct I2C_MSG I2cMsgOut1={I2C_PCA1_ADDR,
                      I2C_PCA_CTL_LS0,
                          1,
  0x42};

struct I2C_MSG I2cMsgIn1={ I2C_PCA2_ADDR,
```

```
  I2C_PCA_CTL_INPUT,
                          1};
//
//Uint16 i2cMode = 0;
//
//struct I2C_MSG I2CBuffer[I2CBUFFSIZE];
//struct I2C_MSG *I2CFIFOFront;
//struct I2C_MSG *I2CFIFOBack;
//struct I2C_MSG *I2CBufferEnd;
//
//
////Uint16 I2CTXFlag;
////Uint16 I2CRXFlag;
//
//
////void   InitI2CModule(void){
////   // Initialize I2C
////   I2caRegs.I2CSAR = I2C_PCA0_ADDR; // Slave address - EEPROM control code
////
////   #if (CPU_FRQ_150MHZ)             // Default - For 150MHz SYSCLKOUT
////        I2caRegs.I2CPSC.all = 14;   // Prescaler - need 7-12 Mhz on module clk (150/
////   #endif
////   #if (CPU_FRQ_100MHZ)             // For 100 MHz SYSCLKOUT
////     I2caRegs.I2CPSC.all = 9;     // Prescaler - need 7-12 Mhz on module clk (100/10
////   #endif
////
//////   I2caRegs.I2CCLKL = 10; // NOTE: must be non zero
//////   I2caRegs.I2CCLKH = 5; // NOTE: must be non zero
////   I2caRegs.I2CCLKL = 100; // NOTE: must be non zero
////   I2caRegs.I2CCLKH = 100; // NOTE: must be non zero
////   I2caRegs.I2CIER.all = 0x24; // Enable SCD & ARDY interrupts
////
////   I2caRegs.I2CFFTX.all = 0x6000; // Enable FIFO mode and TXFIFO
////   I2caRegs.I2CFFRX.all = 0x2040; // Enable RXFIFO, clear RXFFINT,
////
////   I2caRegs.I2CMDR.all = 0x0020; // Take I2C out of reset
////    // Stop I2C when suspended
////   i2cMode = I2C_TX;
////   return;
////}
//
void InitPCAs(){
Uint16 result;
I2cMsgOut1.SlaveAddress = I2C_PCA1_ADDR;
I2cMsgOut1.CommandByte = I2C_PCA_CTL_PSC0 | I2C_PCA_CTL_AUTO;
```

```
I2cMsgOut1.MsgBuffer[0] = 0x03; //PCS0 ~ 40Hz
I2cMsgOut1.MsgBuffer[1] = 0x30; //PWM0
I2cMsgOut1.MsgBuffer[2] = 0x97; //PCS1  ~half a second
I2cMsgOut1.MsgBuffer[3] = 0x80; //PWM1
I2cMsgOut1.MsgBuffer[4] = 0x00; //LS0
I2cMsgOut1.MsgBuffer[5] = 0x00; //LS1
I2cMsgOut1.NumOfBytes = 6;
do{
result = I2CA_WriteData(&I2cMsgOut1);
}while(result != I2C_SUCCESS);
I2cMsgOut1.SlaveAddress = I2C_PCA3_ADDR;
do{
result = I2CA_WriteData(&I2cMsgOut1);
}while(result != I2C_SUCCESS);
}


Uint16 PollPhases(){
Uint16 result;
I2cMsgIn1.SlaveAddress = I2C_PCA0_ADDR;
I2cMsgIn1.CommandByte = I2C_PCA_CTL_INPUT;
I2cMsgIn1.NumOfBytes = 1;
do{
result = I2CA_ReadDataBlocking(&I2cMsgIn1);
}while(result != I2C_SUCCESS);
return I2cMsgIn1.MsgBuffer[1];
}

Uint16 PollButtons(){
Uint16 result;
I2cMsgIn1.SlaveAddress = I2C_PCA2_ADDR;
I2cMsgIn1.CommandByte = I2C_PCA_CTL_INPUT;
I2cMsgIn1.NumOfBytes = 1;
do{
result = I2CA_ReadDataBlocking(&I2cMsgIn1);
}while(result != I2C_SUCCESS);
return I2cMsgIn1.MsgBuffer[1];
}

void  SetPCA1Status(Uint16 value){
Uint16 retval = 0;
I2cMsgOut1.SlaveAddress = I2C_PCA1_ADDR;
I2cMsgOut1.CommandByte = I2C_PCA_CTL_LS0 | I2C_PCA_CTL_AUTO;
I2cMsgOut1.MsgBuffer[0] = value & 0x00FF;
I2cMsgOut1.MsgBuffer[1] = (value>>8) & 0x00FF;
```

```
I2cMsgOut1.NumOfBytes = 2;
do{
retval = I2CA_WriteData(&I2cMsgOut1);
}
while (retval != I2C_SUCCESS);
if(retval != I2C_SUCCESS){
char* msg;
msg = "Error in Writing I2C SetPCA1\0";
scia_msg_async(msg);
}
}
void  SetPCA3Status(Uint16 value){
Uint16 retval = 0;
I2cMsgOut1.SlaveAddress = I2C_PCA3_ADDR;
I2cMsgOut1.CommandByte = I2C_PCA_CTL_LS0 | I2C_PCA_CTL_AUTO;
I2cMsgOut1.MsgBuffer[0] = value & 0x00FF;
I2cMsgOut1.MsgBuffer[1] = (value>>8) & 0x00FF;
I2cMsgOut1.NumOfBytes = 2;
do{
retval = I2CA_WriteData(&I2cMsgOut1);
}
while (retval != I2C_SUCCESS);
if(retval != I2C_SUCCESS){
char* msg;
msg = "Error in Writing I2C SetPCA1\0";
scia_msg_async(msg);
}
}

void I2CA_Init(void)
{
   // Initialize I2C
   I2caRegs.I2CSAR = I2C_PCA0_ADDR; // Slave address - EEPROM control code

//   #if (CPU_FRQ_150MHZ)             // Default - For 150MHz SYSCLKOUT
        I2caRegs.I2CPSC.all = 14;   // Prescaler - need 7-12 Mhz on module clk (150/15 =
//   #endif
//   #if (CPU_FRQ_100MHZ)             // For 100 MHz SYSCLKOUT
//     I2caRegs.I2CPSC.all = 9;     // Prescaler - need 7-12 Mhz on module clk (100/10 =
//   #endif

//   I2caRegs.I2CCLKL = 10; // NOTE: must be non zero
//   I2caRegs.I2CCLKH = 5; // NOTE: must be non zero
   I2caRegs.I2CCLKL = 50; // NOTE: must be non zero
   I2caRegs.I2CCLKH = 50; // NOTE: must be non zero
```

```
   I2caRegs.I2CIER.all = 0x24; // Enable SCD & ARDY interrupts

   I2caRegs.I2CFFTX.all = 0x6000; // Enable FIFO mode and TXFIFO
   I2caRegs.I2CFFRX.all = 0x2040; // Enable RXFIFO, clear RXFFINT,

   I2caRegs.I2CMDR.all = 0x0020; // Take I2C out of reset
    // Stop I2C when suspended

//// I2CBuffer[I2CBUFFSIZE];
// I2CFIFOFront = I2CBuffer;
// I2CFIFOBack = I2CBuffer;
//// I2CBufferEnd = I2CBuffer + sizeof(struct I2C_MSG)*(I2CBUFFSIZE-1);
// I2CBufferEnd = &(I2CBuffer[I2CBUFFSIZE-1]);
   return;
}

Uint16 I2CA_WriteData(struct I2C_MSG *msg)
{
   Uint16 i;

   // Wait until the STP bit is cleared from any previous master communication.
   // Clearing of this bit by the module is delayed until after the SCD bit is
   // set. If this bit is not checked prior to initiating a new message, the
   // I2C could get confused.
   if (I2caRegs.I2CMDR.bit.STP == 1)
   {
      return I2C_STP_NOT_READY_ERROR;
   }

   // Setup slave address
   I2caRegs.I2CSAR = msg->SlaveAddress;
   //I2caRegs.I2CSAR = I2C_PCA1_ADDR;

   // Check if bus busy
   if (I2caRegs.I2CSTR.bit.BB == 1)
   {
      return I2C_BUS_BUSY_ERROR;
   }

   // Setup number of bytes to send
   // MsgBuffer + Address
   I2caRegs.I2CCNT = msg->NumOfBytes+1;
   //I2caRegs.I2CCNT = 1; //send control byte and data byte.

   // Setup data to send
```

```
   I2caRegs.I2CDXR = msg->CommandByte;
//   I2caRegs.I2CDXR = msg->MemoryLowAddr;
//// for (i=0; i<msg->NumOfBytes-2; i++)
   for (i=0; i<msg->NumOfBytes; i++)
   {
      I2caRegs.I2CDXR = *(msg->MsgBuffer+i);
   }
// I2caRegs.I2CDXR = I2C_PCA_CTL_LS0;
// I2caRegs.I2CDXR = 0x42;

   // Send start as master transmitter
   I2caRegs.I2CMDR.all = 0x6E20;

   return I2C_SUCCESS;
}

Uint16 I2CA_ReadData(struct I2C_MSG *msg)
{
   // Wait until the STP bit is cleared from any previous master communication.
   // Clearing of this bit by the module is delayed until after the SCD bit is
   // set. If this bit is not checked prior to initiating a new message, the
   // I2C could get confused.
   if (I2caRegs.I2CMDR.bit.STP == 1)
   {
      return I2C_STP_NOT_READY_ERROR;
   }

   I2caRegs.I2CSAR = msg->SlaveAddress;

//   I2caRegs.I2CCNT = 2;
//   I2caRegs.I2CDXR = msg->MemoryHighAddr;
//   I2caRegs.I2CDXR = msg->MemoryLowAddr;

//   I2caRegs.I2CDXR = msg->MemoryHighAddr;
   I2caRegs.I2CMDR.all = 0x2620; // Send data to setup EEPROM address

   //else if(msg->MsgStatus == I2C_MSGSTAT_RESTART)
   {
      I2caRegs.I2CCNT = msg->NumOfBytes; // Setup how many bytes to expect
//      I2caRegs.I2CMDR.all = 0x2C20; // Send restart as master receiver
      I2caRegs.I2CMDR.all = 0xAC20; // Send restart as master receiver, enable NACK.
   }

   return I2C_SUCCESS;
}
```

```
Uint16 I2CA_ReadDataBlocking(struct I2C_MSG *msg)
{
   // Wait until the STP bit is cleared from any previous master communication.
   // Clearing of this bit by the module is delayed until after the SCD bit is
   // set. If this bit is not checked prior to initiating a new message, the
   // I2C could get confused.
   if (I2caRegs.I2CMDR.bit.STP == 1)
   {
      return I2C_STP_NOT_READY_ERROR;
   }

   I2caRegs.I2CSAR = msg->SlaveAddress;

//   I2caRegs.I2CCNT = 2;
//   I2caRegs.I2CDXR = msg->MemoryHighAddr;
//   I2caRegs.I2CDXR = msg->MemoryLowAddr;

//   I2caRegs.I2CDXR = msg->MemoryHighAddr;
   I2caRegs.I2CMDR.all = 0x2620; // Send data to setup EEPROM address

   //else if(msg->MsgStatus == I2C_MSGSTAT_RESTART)
   {
      I2caRegs.I2CCNT = msg->NumOfBytes; // Setup how many bytes to expect
//      I2caRegs.I2CMDR.all = 0x2C20; // Send restart as master receiver
      I2caRegs.I2CMDR.all = 0xAC20; // Send restart as master receiver, enable NACK.
   }
   while(I2caRegs.I2CSTR.bit.NACKSNT == 0){
    ;
   }
   I2cMsgIn1.MsgBuffer[1] = I2caRegs.I2CDRR;
   I2caRegs.I2CSTR.bit.NACKSNT = 1;


   return I2C_SUCCESS;
}


interrupt void i2c_int1a_isr(void)     // I2C-A
{
   Uint16 IntSource;
//   Uint16 res;

   // Read interrupt source
```

```
   IntSource = I2caRegs.I2CISRC.all;

   // Interrupt source = stop condition detected
switch(IntSource){
case I2C_SCD_ISRC: // stop condition detected
// if(I2CFIFOFront != I2CFIFOBack) // Data in I2CBuffer
//     { //try to send out data in our queue.
//     res = I2CA_WriteData(I2CFIFOFront);
//     if(res == I2C_SUCCESS){
//     I2CFIFOFront++;
//     if(I2CFIFOFront > I2CBufferEnd)
//     I2CFIFOFront = I2CBuffer;
//     }
//     I2caRegs.I2CFFTX.bit.TXFFINTCLR = 1;
//     break;
//     }
    //nothing in queue, must be receiving data:
case I2C_RX_ISRC:
I2cMsgIn1.MsgBuffer[0] = I2caRegs.I2CCNT;
I2cMsgIn1.MsgBuffer[1] = I2caRegs.I2CDRR;
break;
case I2C_ARDY_ISRC: // register access ready
break;
case I2C_NO_ISRC:  //No Interrupt source
break;
case I2C_ARB_ISRC:
break; // Arbitration lost
case I2C_NACK_ISRC: // No-ack condition detected
break;
case I2C_TX_ISRC: // Transmit data ready
break;
case I2C_AAS_ISRC: // Addressed as slave
break;
default:
break;
}
// Enable future I2C (PIE Group 8) interrupts
PieCtrlRegs.PIEACK.all = PIEACK_GROUP8;
}
//
//Uint16 I2CA_WriteDataAsync(struct I2C_MSG *msg){
// *I2CFIFOBack++ = *msg; //Possibly very expensive.
// if(I2CFIFOBack > I2CBufferEnd)
// I2CFIFOBack = I2CBuffer;
// if(I2caRegs.I2CFFTX.bit.TXFFIENA == 0){
```

```
// I2CA_WriteData(I2CFIFOFront);
// I2caRegs.I2CFFTX.bit.TXFFIENA = 1;
// }
// return I2C_SUCCESS;
//}
//
```

ADCHandler.h

```
#ifndef ADCHANDLER_H_
#define ADCHANDLER_H_

//#include "DSP28x_Project.h"     // Device Headerfile and Examples Include File
#include "PeripheralHeaderIncludes_F28335.h"

#define CPU_RATE    6.667L   // for a 150MHz CPU clock speed (SYSCLKOUT)
//************************ ADC start parameters ********************************
//#if (CPU_FRQ_150MHZ)     // Default - 150 MHz SYSCLKOUT
  #define ADC_MODCLK 0x3 // HSPCLK = SYSCLKOUT/2*ADC_MODCLK2 = 150/(2*3)   = 25.0 MHz
//#endif
#if (CPU_FRQ_100MHZ)
  #define ADC_MODCLK 0x2 // HSPCLK = SYSCLKOUT/2*ADC_MODCLK2 = 100/(2*2)   = 25.0 MHz
#endif
#define ADC_CKPS   0x1   // ADC module clock = HSPCLK/2*ADC_CKPS   = 25.0MHz/(1*2) = 12.
#define ADC_SHCLK  0xf   // S/H width in ADC module periods                        = 16
#define AVG        1000  // Average sample limit
#define ZOFFSET    0x00  // Average Zero offset
#define BUF_SIZE   1024  // Sample buffer size
//************************ End ADC start parameters ****************************

void InitADCModule();

interrupt void adc_isr(void);


#define DELAY_US(A)  DSP28x_usDelay(((((long double) A * 1000.0L) / (long double)CPU_RAT

//ADC Variables
extern Uint16 SampleTable[BUF_SIZE];

#endif /*ADCHANDLER_H_*/
```

ADCHandler.c

```
#include "ADCHandler.h"
#include "PeripheralHeaderIncludes_F28335.h"     // Device Headerfile and Examples Inclu
#include "PWMHandler.h"
```

```
//#include "Globals.h"

#define ADC_usDELAY  5000L

#pragma CODE_SECTION(adc_isr, "ramfuncs");
//#pragma CODE_SECTION(InitAdc, "ramfuncs");
//#pragma CODE_SECTION(InitADCModule, "ramfuncs");

//ADC Variables
Uint16 SampleTable[BUF_SIZE];
volatile long bob = 5000L;
volatile char counter = 0;

void InitAdc(void);


interrupt void  adc_isr(void)
{
static unsigned char i;
static Uint16 idx = 10;
SampleTable[idx+36] = AdcRegs.ADCRESULT9>>4; //I5
SampleTable[idx+32] = AdcRegs.ADCRESULT8>>4; //Vout5
SampleTable[idx+28] = AdcRegs.ADCRESULT7>>4; //I4
SampleTable[idx+24] = AdcRegs.ADCRESULT6>>4; //Vout4
SampleTable[idx+20] = AdcRegs.ADCRESULT5>>4; //I3
SampleTable[idx+16] = AdcRegs.ADCRESULT4>>4; //Vout3
SampleTable[idx+12] = AdcRegs.ADCRESULT3>>4; //I2
SampleTable[idx+8]  = AdcRegs.ADCRESULT2>>4; //Vout2
SampleTable[idx+4]  = AdcRegs.ADCRESULT1>>4; //I1
   SampleTable[idx++]  = AdcRegs.ADCRESULT0>>4;  //Vout1

   if(idx > 13){
   idx = 10;
   for(i = 0; i < 10; i++){
SampleTable[i] = (SampleTable[10+4*i] + SampleTable[11+4*i] + SampleTable[12+4*i] + Samp
}
   //MainFlags.bit.AdcComputeFlag = 1;
   }
//  Voltage2[0] = AdcRegs.ADCRESULT1; //>>4;

  // If 40 conversions have been logged, start over
//  if(ConversionCount == 0x1FD) //two seconds
//  {
//     ConversionCount = 0;
//     XmitFlag = 1;
```

```
//  }
//  else ConversionCount++;

  // Reinitialize for next ADC sequence
   EALLOW;
   AdcRegs.ADCTRL2.bit.RST_SEQ1 = 1;         // Reset SEQ1
   AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1;       // Clear INT SEQ1 bit
   PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;   // Acknowledge interrupt to PIE
EDIS;
   return;
}




void InitADCModule(){
    InitAdc();
    EALLOW;
    // Specific ADC setup for this example:
    AdcRegs.ADCTRL1.bit.ACQ_PS = ADC_SHCLK;
    AdcRegs.ADCTRL3.bit.ADCCLKPS = ADC_CKPS;
    AdcRegs.ADCTRL3.bit.SMODE_SEL = 1;  // Simultaneous sampling mode
    AdcRegs.ADCTRL1.bit.SEQ_CASC = 1;         // 1  Cascaded mode
    AdcRegs.ADCTRL1.bit.CONT_RUN = 0;        // Setup noncontinuous run

//Single phase setup
    //AdcRegs.ADCMAXCONV.all = 0x0000;        // Setup all conv's on SEQ1, only 1 conver
    //AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0x00;  // Sample Phase 1 Vout/IOut store in ADCR

    //2 Phase Setup
    AdcRegs.ADCMAXCONV.all = 0x0004;        // Setup all conv's on SEQ1, only 1 conversi
    AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0x00;  // Sample Phase 1 Vout/IOut store in ADCRES
    AdcRegs.ADCCHSELSEQ1.bit.CONV01 = 0x02; // Sample Phase 2 Vout/IOut store in ADCRESU
    AdcRegs.ADCCHSELSEQ1.bit.CONV02 = 0x04; // Sample Phase 3 Vout/IOut store in ADCRESU
    AdcRegs.ADCCHSELSEQ1.bit.CONV03 = 0x06; // Sample Phase 4 Vout/IOut store in ADCRESU
    AdcRegs.ADCCHSELSEQ2.bit.CONV04 = 0x08; // Sample Phase 5 Vout/IOut store in ADCRESU

    AdcRegs.ADCTRL2.bit.EPWM_SOCA_SEQ1 = 1; // Enable SOCA from ePWM to start SEQ1
    AdcRegs.ADCTRL2.bit.INT_ENA_SEQ1 = 1;   // Enable SEQ1 interrupt (every EOS)

SysCtrlRegs.HISPCP.all = ADC_MODCLK;
EDIS;
}
```

```
//#include "DSP2833x_Device.h"     // DSP2833x Headerfile Include File
//#include "DSP2833x_Examples.h"   // DSP2833x Examples Include File


//---------------------------------------------------------------------------
// InitAdc:
//---------------------------------------------------------------------------
// This function initializes ADC to a known state.
//
void InitAdc(void)
{
    extern void DSP28x_usDelay(Uint32 Count);


    // *IMPORTANT*
// The ADC_cal function, which  copies the ADC calibration values from TI reserved
// OTP into the ADCREFSEL and ADCOFFTRIM registers, occurs automatically in the
// Boot ROM. If the boot ROM code is bypassed during the debug process, the
// following function MUST be called for the ADC to function according
// to specification. The clocks to the ADC MUST be enabled before calling this
// function.
// See the device data manual and/or the ADC Reference
// Manual for more information.
counter =1;
    EALLOW;
SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 1;
ADC_cal();
EDIS;




    // To powerup the ADC the ADCENCLK bit should be set first to enable
    // clocks, followed by powering up the bandgap, reference circuitry, and ADC core.
    // Before the first conversion is performed a 5ms delay must be observed
// after power up to give all analog circuits time to power up and settle

    // Please note that for the delay function below to operate correctly the
// CPU_RATE define statement in the DSP2833x_Examples.h file must
// contain the correct CPU clock period in nanoseconds.
EALLOW;
    AdcRegs.ADCTRL3.all = 0x00E0;  // Power up bandgap/reference/ADC circuits
    EDIS;
    //DELAY_US(ADC_usDELAY);         // Delay before converting ADC channels
    //DSP28x_usDelay(ADC_usDELAY);
```

```
    while(bob-- > 0){
     counter++;
    }
}
```