OBSERVER BASED FAULT DETECTION IN DC-DC POWER CONVERTERS

BY

KIERAN T. LEVIN

THESIS

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Electrical and Computer Engineering in the Graduate College of the University of Illinois at Urbana-Champaign, 2012

Urbana, Illinois

Adviser:

Assistant Professor Alejandro D. Domínguez-García
ABSTRACT

Power electronics today are limited in their operational lifetimes, which can have negative consequences for critical systems which depend on power electronic to stay functional. To help mitigate the effects of system failures due to power electronics, a fault detection filter has been implemented to detect both hard and soft faults in the power supply, while determining how much load the supply can power while staying in specification in its reduced operating state allowing reduced system operation or maintenance. In this thesis, we study the effectiveness of such filters by testing them in a hardware testbed. This testbed is comprised of a dc-dc buck converter. The detection filters for monitoring the health of the components in this dc-dc converter, as well as the converter controls, are implemented in a low-cost DSP. Using a lossy converter model which runs in real time on the DSP, the model is continuously compared to the actual converter states generating an error signal which can be used to characterize both the nature of the fault and fault magnitude. A dc-dc converter is controlled by a TMS320F28335 DSP which runs the fault detection filter. The fault detection filter uses an explicit solver with a variable time step to compute the filter residuals, allowing for accurate fault detection on low cost hardware.
To my wife Elizabeth, who removed the distractions in my life, allowing me
to finish writing this thesis.

To my parents, for their love and support.
I thank Alejandro Domínguez-García for advising me, and for being patient and helpful.
# TABLE OF CONTENTS

**CHAPTER 1** INTRODUCTION ........................................ 1
   1.1 FDI in power electronics .................................... 1
   1.2 Current work using FDI in power electronics .......... 2
   1.3 Relevance ..................................................... 3
   1.4 Our approach ............................................... 4
   1.5 Use of existing hardware .................................. 4
   1.6 Limitations .................................................. 5

**CHAPTER 2** THEORY OF PIECEWISE LINEAR OBSERVERS ......... 6
   2.1 Examples .................................................... 6
   2.2 FDI filter design .......................................... 7
   2.3 Integrator solver ........................................... 9
   2.4 Explicit solver ............................................ 10

**CHAPTER 3** HARDWARE PLATFORM ................................. 11
   3.1 Hardware overview ......................................... 11
   3.2 Limitations ................................................ 15
   3.3 Algorithms ............................................... 16

**CHAPTER 4** RESULTS ............................................... 18
   4.1 Detectability of faults ..................................... 18
   4.2 Raw measurements ......................................... 27
   4.3 No fault static behavior .................................. 27
   4.4 Faulty output capacitor .................................. 29
   4.5 Faulty inductor ............................................ 33

**CHAPTER 5** CONCLUSIONS ........................................ 35
   5.1 Successes .................................................. 35
   5.2 Failures .................................................... 35

**APPENDIX A** SCHEMATICS .......................................... 36

**APPENDIX B** THEORETICAL NOISE OF MEASUREMENTS ....... 38
CHAPTER 1

INTRODUCTION

Electrical systems today require higher levels of reliability and self diagnosis. With the advent of long life electronics for biomedical and commercial applications, the need for fault diagnosis to help achieve higher levels of reliability will become more important. Real time fault detection in dc-dc converters is an important step to building higher reliability systems. If faults can be detected early they can allow the converter to change its operating parameters, or if it is operating with a smart system it may be able to communicate its health to the load or supply, allowing other parts of the system to adjust accordingly - allowing for longer operation - and prevent catastrophic failures from occurring unexpectedly.

Fault-tolerance may be defined as the ability of a system to adapt and compensate in a planned, systematic way to component faults and keep delivering, completely or partially, the functionality for which it was designed. Fault-tolerant electrical power supplies are paramount in many applications, ranging from safety- and mission-critical systems for aircraft, spacecraft and automobiles, to communication systems, computers supporting financial markets, and industrial control equipment. In all these applications, power electronics systems are the essential building blocks of the electric power supply.

1.1 FDI in power electronics

Fault detection in power electronics, and for electronic circuits in general, has existed for many years; however, only recently has the processing power of embedded computers increased to the point where complex and fast digital filters are able to be implemented cheaply enough to allow small computational units to run complex real time filters to detect the presence of faults
in real time. Many existing topologies have been proposed to detect faults, such as the addition of thermal sensors to detect excessive power loss or failed components, limits built into controller chips that can sense over-current or over-voltage conditions or the addition of extra sensors to the power converter.

In a power electronics system, key elements to achieving fault tolerance are: component redundancy, a fault diagnosis system, and a reconfiguration system that, upon information provided by the diagnosis system, removes faulty components and usually substitutes them with redundant ones. A fault diagnosis system executes three tasks [1]: i) detection makes a binary decision whether or not a fault has occurred, ii) isolation determines the location of the faulty component, and iii) a severity assessment determines the extent of the fault.

In general, methods for fault diagnosis can be broadly classified into three categories [1]: i) model-based, which includes observer-based detection filters and fault knowledge-based methods that can point to specific faults by measuring certain variables; ii) artificial intelligence, which uses neural networks and fuzzy logic to develop expert systems that, once trained, can point to specific faults; and iii) empirical and signal processing methods, which use spectral analysis to identify specific signatures of a certain fault.

1.2 Current work using FDI in power electronics

In the context of power electronics, there has been work on fault knowledge-based methods [2, 3, 4]. There has also been substantial work on the application of artificial intelligence [5, 6, 7], and signal processing methods [8].

In this thesis, we focus on developing observer-based fault diagnosis methods for power electronics systems. In particular, we extend the method for fault diagnosis of linear time-invariant (LTI) systems developed by Beard in [9] and Jones in [10], to switched-linear systems commonly encountered in power electronics. In the Beard-Jones method, a detection filter with the same structure of a Luenberger observer (see, e.g., [11]) is constructed. In the absence of faults, the filter works exactly the same way as the Luenberger observer, and the state estimates obtained from the filter converge to the actual values asymptotically. If a fault occurs, the state predicted by
the detection filter differs from the true state of the system. For particular faults, by appropriately choosing the filter gain, the filter residuals have certain geometrical characteristics that make the fault identifiable.

1.3 Relevance

Partly faulted components in power circuits can cause a multitude of problems, causing premature failure or a catastrophic failure in other components. For a simple boost or buck converter, component failures such as soft capacitor faults or bad solder joints near high temperature components can slowly degrade over time causing gradual degradation of performance. Loss of converter functionality can cause catastrophic failures in components that are powered by the converter.

1.3.1 Electrolytic failure

One example of a common component failure in power converters is the electrolytic capacitor. With a mean time between failures between 10,000 and 100,000 hours [12], failures due to capacitor ageing are quite high after a few years of operation. A common soft fault in electrolytic capacitors is a slow ageing process causing the ESR of the capacitor to increase slowly over the lifetime of the unit [12]. Once the ESR increases beyond the designer’s limits the capacitor is said to have failed. This increased internal resistance causes the capacitor to heat itself further, causing a runaway effect near the end of the capacitor’s life. In the case of a power supply - if this was detected early - the capacitor could be repaired, or the power supply could operate at reduced capacity near the end of its life, allowing for graceful degradation of its performance. Also in this case reducing the load would reduce the ripple on the output capacitance - thus extending the life of the damaged supply. In situations where repair is difficult this performance/operational trade-off can sometimes be accepted or preferred instead of a binary outcome.
1.3.2 Inductor failure

Inductor failures are usually due to heat or mechanical stresses on the windings, causing shorts or open circuits due to manufacturing flaws or electrical overstresses. A common cause of inductor failures is corrosion [13] which can cause failures in the winding insulation or where the inductor is soldered to the board.

1.3.3 Solder joint failure

A second example of a common failure mode for electronics used in power supplies is the failure of a solder joint on the PCB. Many modern power supplies are using more exotic components which can withstand higher temperatures. This puts higher stresses on the copper PCB and related solder joints. Over the lifetime of the supply there may be mechanical stresses on the board, especially for parts which are in high vibration environments. Similar to the capacitor failure mode mentioned above a small failure can grow into a larger failure over time due to increasing the resistance around a solder connection causing increased local heating, further stressing the connection and leading to premature failure. If partial failures happen, longer lifetimes can be achieved by reducing stresses on these components [14].

1.4 Our approach

Our approach to detecting faults uses a software observer running on a DSP in real time. This model uses measurements from the actual circuit to compare the model operational parameters to the actual system to generate an error residual. The error residual is then used as a fingerprint to determine the magnitude and type of error, which is then used to alert the user or other subsystem to the failure.

1.5 Use of existing hardware

Using existing intelligent controllers in the converter can provide significant cost and manufacturing savings. Thus the observer is implemented in soft-
ware on a C2000 DSP. This DSP is often used in high performance power electronics including motor drive controllers and multiphase inverters. Incorporating the observer into an existing design can allow for an inexpensive method for adding fault detection to existing power supply designs with little modification.

1.6 Limitations

The piecewise linear observer needs to sample the system dynamics at a minimum of once each switch action. Furthermore the sampling window needs to be small in comparison to the operating frequency of the converter so that the initial conditions for the observer and final conditions of the converter can be measured using a small enough time window so that the system does not change appreciably during the measurement interval. Furthermore in the current implementation there is a possibility that more instrumentation may be required to determine the different states of the converter than what is normally used by a conventional power supply, which can increase cost. The hardware platform uses a minimum of four measurements for a buck, boost, or flyback dc-dc converter, input voltage, output voltage, output current, and inductor current. Many existing power supply designs use some combination of these measurements, so it is not always necessary for additional instrumentation to be added for the operation of the fault detection filter.

This implementation uses existing microprocessors such as digital signal processors from Texas Instruments. This allows a low cost addition which is adaptable to many existing solutions where the switching frequency is on the order of tens of kilohertz. With our current hardware and a variable time step solver we have found it easy to run our observer in real time with switching frequencies up to 20 kHz using a DSP. While this is a low operational frequency for many power supplies in digital or personal electronics, it is common for larger and more expensive dc-dc converters which have low nominal operating frequencies such as those found in high power electronics used in the automotive, industrial, and power distribution electronics; this allows a solution to be implemented without costly or exotic hardware. Furthermore, additional optimization of the filter may be possible, allowing the observer to run on even lower cost hardware.
Observers are well understood for control and estimation of linear systems, and many previous works describe the use of averaged models for control of switched power supplies. However these models cannot be used to detect certain failures in switched systems. In this chapter we introduce the need for piecewise linear observers.

In power electronics, it is common to use an averaged model of the system to design the control. The use of averaged models together with the Beard-Jones approach might look like a feasible solution for fault diagnosis in power electronics systems. Careful analysis of certain types of faults, e.g., degradation in the output filter capacitor of a buck converter, shows that this is not the case; i.e., a detection filter based on the averaged model of the converter cannot capture this type of fault. To overcome the limitations of averaged models for fault detection filter design, we use the converter switching model and develop fault sensitive detection filters based on piecewise linear observers [15].

2.1 Examples

An excellent example of a piecewise system is a buck power supply shown in Fig. 2.1. The converter has two switches $S_1$ and $S_2$ of which during normal operation, one or the other switch is on, allowing current to flow through the inductor $L$. During normal operation switch $S_1$ is kept on for some percentage of the time which is related to the desired output voltage from the converter as a percentage of the input voltage. When a switch action is implemented the circuit is effectively reconfigured into a different circuit. For the model of this circuit the matrix coefficients are changed to reflect the new physical model. As the converter is switching rapidly the observer must change out
Figure 2.1: Circuit equivalent of a buck converter

the matrix coefficients rapidly to reflect the actual circuit. Thus when the converter is in state $\sigma = 1$ the observer will be in the corresponding state. When the switches change, effectively reconfiguring the circuit into a different topology, the observer needs to follow this change.

2.2 FDI filter design

Our converter can be easily represented in state space in the following form:

$$\frac{dx(t)}{dt} = A_\sigma x(t) + B_\sigma u(t)$$

where $A_\sigma$ and $B_\sigma$ are piecewise linear. When the configuration of the converter changes, then the state matrices also change. We can define the number of configurations as $P$ so that $\sigma \in P$. In the case of a simple buck or boost converter operating in continuous mode the converter has two configurations. If we label these configurations $[1, 2]$, then it can be said $\sigma \in [1, 2]$. To further understand the configurations of a switch mode power converter take the example of a simple buck converter shown in Fig. 2.1.

When this converter is running in continuous mode it has two operating conditions. For this example state $\sigma = 1$ is when $S_1$ is on and $S_2$ is off. State $\sigma = 2$ is when $S_1$ is off and $S_2$ is on. The converter alternates between these two states rapidly with a switching period of 10 kHz as defined by the controller and the duty cycle. Whenever the switches change their configuration the observer also switches its coefficients to represent the new state of
the circuit. If we choose the states in the circuit to be $V_c$ and $I_L$, the state equations can be written when $\sigma = 1$ as

$$\dot{i}_L = \frac{1}{L}(-i_LR_{dson} - i_LR_L) + \frac{1}{L}(v_{in} - v_{Load}) \quad (2.1)$$

$$\dot{v}_C = \frac{1}{C}i_L - \frac{1}{C}i_{Load} \quad (2.2)$$

In the other operating state when $\sigma = 2$ the state equations become

$$\dot{i}_L = \frac{1}{L}(-i_LR_d - i_LR_L) + \frac{1}{L}(V_{Don} - v_{Load}) \quad (2.3)$$

$$\dot{v}_C = \frac{1}{C}i_L - \frac{1}{C}i_{Load} \quad (2.4)$$

Writing these equations in matrix form we get

$$S_1 \text{ on } \sigma = 1$$

$$\begin{bmatrix} \dot{I}_L \\ \dot{V}_c \end{bmatrix} = \begin{bmatrix} \frac{R_{dson} + R_L + R_{res}}{L} & \frac{1}{L} \\ \frac{1}{C} & 0 \end{bmatrix} \begin{bmatrix} I_L \\ V_c \end{bmatrix} + \begin{bmatrix} \frac{1}{L} & \frac{R_{res}}{L} \\ 0 & -\frac{1}{C} \end{bmatrix} \begin{bmatrix} V_{input} \\ I_{Load} \end{bmatrix} \quad (2.5)$$

$$S_2 \text{ on } \sigma = 2$$

$$\begin{bmatrix} \dot{I}_L \\ \dot{V}_c \end{bmatrix} = \begin{bmatrix} \frac{R_d + R_L + R_{res}}{L} & \frac{1}{L} \\ \frac{1}{C} & 0 \end{bmatrix} \begin{bmatrix} I_L \\ V_c \end{bmatrix} + \begin{bmatrix} \frac{1}{L} & \frac{R_{res}}{L} \\ 0 & -\frac{1}{C} \end{bmatrix} \begin{bmatrix} V_d \\ I_{Load} \end{bmatrix} \quad (2.6)$$

Looking at the equivalent circuit model for this system we can see the configurations of the necessary components of each state are shown in Figures 2.2, and 2.3.
2.3 Integrator solver

How do we solve the equation in the time domain? One method is to integrate both sides of the equation

\[ \frac{d}{dt}x(t) = Ax(t) + Bu(t) \]  \hspace{1cm} (2.7)

We can find the state of the system at some future \( t=T \) providing the input \( u \) is known for all \( t_0 < t < T \); this input can be gathered in real time by measuring the inputs to the circuit. Using this approach we can sample the system and compute the output of the observer many times each time it switches on or off. This does work but is rather crude as it requires a very small time step to get accurate results for a fixed time step solver. Unfortunately dc-dc converters normally run at high frequencies. Most dc-dc converters run above 20 kHz, so a small time step relative to the switching of the converter would require at least 10 steps per switching cycle, leading to a sampling frequency of at least 200 kHz. Even with this high sampling rate the results from this method are not as accurate as we would hope. Taking the system dynamics and moving to the frequency domain it can be shown that if the state equations are expanded in a Taylor series they will have to be truncated at the Nyquist rate. With a sample rate of 200 kHz, the Nyquist rate starts at 100 kHz. Thus we can only expand out this series a few terms until it gets truncated. Looking at the waveforms of the inductor and capacitor current and voltage we see they are triangular and thus to properly capture the dynamics would need many terms of the power series to
be present to approach a good approximation of the original. Thus using a method of discrete time sampling to approximate the system running at the speeds required for high performance dc-dc converters would be challenging using low cost hardware.

2.4 Explicit solver

A better way to estimate the state of the system is to use an explicit solver. We can see that the solution to the state equation of the system at some later time

\[
\frac{dx(t)}{dt} = Ax(t) + Bu(t) \tag{2.8}
\]

is the equation

\[
x(t) = e^{At}x(0) + A^{-1}(e^{At} - I)Bu(t) \tag{2.9}
\]

This yields an exact solution of the system at time \( t \) provided we know the exact state of the system at \( t(0) \), the input \( u(t) \) and the system dynamics. Since the converter hardware is not changing between switch actions, the matrices \( A \) and \( B \) are static while in states \( \sigma = 1 \) and \( \sigma = 2 \). If the starting state, \( x(0) \), is measured, then the only time-varying parameter is \( u(t) \). However for many power conversion systems the output slew rate is slow in comparison to the switching frequency. Thus for converters that drive static or slowly varying loads the input \( u(t) \) can be approximated as constant over the timescale of each switching cycle. The explicit solver is used to solve for the next state of the converter; then the coefficient matrices \( A \) and \( B \) are switched and the state at \( x(t) \) is used for the next starting state

\[
x(t + \tau) = e^{A_\sigma x(t)} + A_\sigma^{-1}(e^{A_\sigma t} - I)B_\sigma u(t) \tag{2.10}
\]

This is the method used with the addition of an observer to calculate the estimated system parameters at each switch action. This method has more advantages than the integrator method mentioned above as it requires only one calculation per switch operation, and it preserves all the system dynamics without having to sample at higher sample rates than once per switch action.
CHAPTER 3

HARDWARE PLATFORM

3.1 Hardware overview

The fault detection filter is implemented in hardware using a custom switching converter circuit that can be easily reconfigured to allow testing of several types of power supply topologies, notably buck, boost, and flyback converter types. The converter is instrumented and measurements are fed into a DSP to allow the observer filter to run in real time alongside the actual hardware. The DSP has several different interfaces including a real time LCD display which shows the health of the converter as shown in Fig. 3.1. The block diagram of the entire converter is shown in Fig. 3.2 and includes a separate power supply, electronic load, converter and DSP that is controlled by a standard desktop PC. The full schematic of the converter is in Appendix A.

Figure 3.1: Test Platform Hardware health display
3.1.1 Power electronics stage

The power electronics stage, shown in Fig. 3.3, connects to a bench power supply on the input, an electronic load on the output, and the DSP for sensor and control signals. The DSP measures input current, voltage, output current and voltage as well as inductor current. It can control three separate MOSFET switches which operate between the supply and inductor or transformer, output and inductor and transformer, and an open collector switch that can be used to switch in and out loads, or wired to generate faults in components. All switches can be bypassed or shorted by jumpers, thus taking them out of the circuit as needed to minimize the unwanted effects of voltage drops and power losses which are not present in the observer model for components which are not active in specific configurations. For example, the switch used for boost operation is shorted when the converter is configured for buck operation.

3.1.2 Sensors

The power supply is monitored by current and voltage sensors on the input and output. For this hardware testbed the input and output current, voltage,
and current through the inductor are measured. The current measurements are taken using an allegro ACS714 high precision Hall effect sensor [16]. This sensor has an isolated current path which allows it to be placed at any potential in the circuit. Each current sensor is connected through a filter and voltage divider to convert its 0-5 V output to a suitable 0 - 3.3 V range used by the DSP input ADC. This sensor has added measurable additional noise due to the fact that the internal gain amplifiers are chopper stabilized, thus generating some external ripple on the output of the sensor.

Voltage measurements are performed by attaching the corresponding node of the circuit through a 10 kΩ variable resistor to divide the voltage down to a range the DSP can handle. This voltage is also capped by reverse protection diodes to ground and the 3.3 V rail to prevent any unexpected damage to the DSP from voltage spikes above the or below the DSP’s acceptable voltage range.

Due to the design of the DSP analog-to-digital converter, the measurements must be buffered to provide a low impedance source for the inputs. This is done using two low-noise four-channel operational amplifiers connected as voltage followers [17]. This buffering allows the DSP to sample the inputs using small sampling windows to get an accurate state of the converter at switch transitions.
3.1.3 Digital signal processor

The core of the hardware platform is the digital signal processor. This provides digital control to allow the converter to track output voltage and current, and also hosts the observer and detection filter. The digital signal processor is a TMS320F28335 DSP from Texas Instruments [18]. It is designed for embedded control systems as well as motor and power supply control, and is the only DSP in its product class to feature a hardware floating point unit (FPU). Using the FPU to perform calculations greatly simplifies the software design requirements. Furthermore this DSP includes a built-in sigma delta analog-digital converter which is used to sample the state of the hardware [19]. This allows customizable sample windows, as well as simultaneous sampling of all measurement channels to occur. For the piecewise linear observer, the sample windows must be small compared to the evolution of the system state to provide almost instantaneous sampling of all measurements. This allows the system state to be precisely measured at switch intervals to provide an accurate snapshot of the system state. The DSP also supports high resolution PWM output which is used to control the converter switches as well as drive the timing for the sampling of the analog-digital converter [20]. The PWM timer block generates a software interrupt whenever the output state changes, thus allowing a software trigger for the analog-digital converter, allowing the DSP to sample all measurements in time with switch changes. This tight integration of measurements and control allows the DSP to accurately measure the system state at the exact timing required for the model to evolve using linear dynamics.

For measurement purposes the DSP has a high speed external memory interface which allows recording of measured data as well as the evolution of system states accurately and at full speed. To do this at each time step the measurements and system states are written into the external memory until the memory is full; then the capture is stopped and the capture is written out through a serial link to the computer at non-real-time speed. This is similar to having a trigger based digital storage oscilloscope built into the DSP, and is very useful for highly accurate debugging and the gathering of results. This high speed capturing of data can also be configured to trigger events while the system is capturing data through the computer interface, which allows the observation of system transitions from one operating condition
to another. For example if the system dynamics of a load transient are to be captured, the DSP can capture data for several hundred cycles before the transient, trigger the third FET causing the load resistance to change, and continue to capture data in real time for several thousand cycles after this event. This capability makes it an excellent way to capture all system parameters and data during a transient.

3.1.4 Supply and load

The power circuit is powered by a Kenwood PD56-10D benchtop adjustable output lab bench power supply. The load consists of a HP 6060B electronic load interfaced to the control computer through a GPIB port.

3.2 Limitations

This test platform is built entirely with low cost hardware. For purposes of experiments and due to processor speed, the switching frequency was chosen to be 10 kHz. This was chosen to ensure that the ADCs on the DSP had enough time to perform conversion over the widest possible duty cycle range and also to allow the DSP enough time to solve for the filter residual during each cycle. The level of noise as well as the maximum accuracy of the ADCs on the DSP generate a lower bound on the level of fault that is detectable in the observer. Furthermore this is highly dependent on the fundamental requirements of the power supply.

For example, the ADC input on the DSP is 12 bits, with a range of 0 to 3.3 V. This gives an input voltage range that is scaled from 20 V down to 3.3 V of $\frac{20}{4096} = 4.88 \text{mV/bit}$. If specification requires that the output voltage not have a ripple higher than 50 mV, then there are only four usable bits that will change between between a measurement of ripple. Detecting the change in error with such a small number of differing states can prove futile however, at high speeds the ADC has an even lower number of effective bits.
3.3 Algorithms

The central algorithm which the DSP solves is the explicit solution to the equation \( \frac{dx(t)}{dt} = Ax(t) + Bu(t) \). To solve this equation the observer must calculate \( e^{At} \) for each switching period. There are many ways to compute matrix exponentials; however when time is critical, a computationally efficient manner of computing the exponential is needed. The Padé squaring and scaling algorithm was chosen as an adequate candidate [21].

To design the software to solve fixed size matrix exponentials, C code was generated from the Matlab demo file expmdemo1 to test the effectiveness of computing the solution using this method. However when this was done it was found that for small embedded systems the C code generated by Matlab was inefficient for many parts of the algorithm. For example, part of the algorithm requires scaling the input and output values of the matrix by powers of two, which is extremely easy to perform in software using the arithmetic shift operation and takes 1 CPU cycle to perform. However in code generated by Matlab the scaling was performed using division, which can be a timely CPU operation. Repeated scaling and squaring is a key part of the algorithm so the entire matrix exponential algorithm was written from scratch.

The observer has an upper bound on the time it can take to calculate the next step in the simulation. However the real world measurements can have samples that come in very short succession when the duty cycle is near 0 or 100%. To help mitigate the need to do each iteration before the subsequent measurement, the measurements are pushed into a circular ring buffer and removed when the filter has finished its calculation and requires another measurement. This allows for the requirement that during each switching period the algorithm needs to perform two iterations; however, it does not matter how long each iteration takes as long as they finish with an average time of less than the time it takes for the converter to complete one switching cycle.

This method introduces some delay into the detection of faults. However the worst case behavior will be if the output of the filter generates its result halfway though the next switching cycle. Thus for this implementation it would be impossible to detect a failure until partway through the subsequent switching cycle.
3.3.1 Integration solver

The first approach that was used to solve the observer equations was an integrator solver, with one integrator for each state. The solver ran at a fixed frequency of 10 times the switching frequency, 10 kHz. The integration based solver was first designed in Simulink and it was attempted to be ported to the DSP. However the code generated from the Simulink code could not run fast enough on the hardware used, so the algorithm was manually ported to the DSP written in C using interrupt driven hardware. Using this method the integration solver ran in real time on the hardware.

3.3.2 Explicit solver

Following the approach used by the integration solver, the explicit solver was first tested using a system model written in Matlab before being ported to the DSP. In the explicit solver the interrupt software was rewritten to trigger measurements slaved from switch events which were controlled by a software PWM control loop on the DSP. The ADC measurements are slaved from the PWM controller to take one complete measurement of the system state. Each set of measurements is then pushed into a ring buffer that allows the algorithm to run with a fixed, known time step while the measurements are pushed into the ring with different interval timing. This allows the variable time step measurements to be computed within a fixed time. Finally the output of the explicit solver is then placed into a feature detection filter that converts the measured error of the system state into a specific failure case. The code listing for the explicit solver is listed in Appendix C.

Voltage Controller

The DSP uses a PID controller implemented in software to regulate the output voltage so it is set to a known value. This is implemented by measuring the output voltage of the converter - and setting the reference voltage using serial commands from a computer. Each iteration of the converter the error is fed into the PID controller, allowing regulation of the supply.
CHAPTER 4

RESULTS

4.1 Detectability of faults

For an observer to be effective at detecting faults, the integrity of the detection filter must be determined. For a simple observer based fault detection circuit, the output of the filter is the error between the actual measurements and the simulated system. In the case of a piecewise linear observer for a buck or boost converter, the states are the storage elements. The most useful outcome of this system would be to detect any fault in the circuit as well as the severity.

4.1.1 System model

The simulation assumes perfect switching. This is not the case in reality as all switches have some finite time to transition from their on to off state; however, since the actual hardware runs at a slow speed, switch transitions take only a small fraction of the cycle time and thus can be assumed to be instantaneous. Finally, this perfect system model ignores the effect of noise in the system; however, the effects of noise will be considered later on.

4.1.2 Effects of noise

An observer based fault detection filter consists of an observer, a physical system, sensors, and external noise. The effects of noise on the stability of the observer and its ability to detect faults are directly related. If there is too much noise or sensor variation in the measurements, the observer is effectively useless as the signals that it is attempting to measure will be buried in the noise. To model this behavior we can model the observer using a standard
system model of an observer with noise added. For the purpose of these experiments the level of noise input to the observer through the DSP was measured using two methods.

4.1.3 Idle noise

The first measurement that was taken was measuring the jitter of any DSP measurements when the converter was not running but all sensors were on. 4096 samples of data were measured and both the average and maximum values were recorded for each measurement channel. Iterating over several times, the amount of noise present in the sensors was determined. It can be noted that different sensors have different noise levels. In the case of the current sensor we were using, a chopper stabilized op amp used in the gain stage caused increased noise levels to be present.

4.1.4 Running error

The second set of noise measurements taken was to determine the accuracy of the measurements in relation to each other at a nominal operating point. For this measurement the converter was switched on with a fixed duty cycle as well as source and load. Then again the measurements were recorded over thousands of cycles. Finally the measurements were linearized around this fixed operating point and the difference between nominal values and measured values was calculated for each measurement. Finally the average and maximum of these values were calculated to determine the average and maximum deviation around nominal measurements. This allows the measurement of the ability of the sensors to measure each sensor in the system while running at a fixed steady state equilibrium to find the measurement jitter, or noise present present around a specific operating point of the converter.

4.1.5 Simulation

To test the ability of the system to detect faults, simulations were developed to determine the minimum detectable faults in the presence of noise in the sensors. To do this a model of the converter was built in Matlab using
Table 4.1: Buck Converter Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_{ESR}$</td>
<td>0.082 Ω</td>
</tr>
<tr>
<td>$R_{ds}$</td>
<td>0.022 Ω</td>
</tr>
<tr>
<td>$R_d$</td>
<td>0.10 Ω</td>
</tr>
<tr>
<td>$R_L$</td>
<td>0.04 Ω</td>
</tr>
<tr>
<td>$L$</td>
<td>500.9 µH</td>
</tr>
<tr>
<td>$C$</td>
<td>534.0 µH</td>
</tr>
<tr>
<td>$V_d$</td>
<td>514 mV</td>
</tr>
<tr>
<td>$V_{in}$</td>
<td>12 V</td>
</tr>
<tr>
<td>$V_{load}$</td>
<td>5 V</td>
</tr>
<tr>
<td>$I_{load}$</td>
<td>0.8 A</td>
</tr>
</tbody>
</table>

A piecewise linear system model listed in Appendix D.6. A piecewise linear observer was connected to the output of the model and both are simulated in tandem using the parameters in Table 4.1. Finally the power supply model allows faults to be injected into different components. For energy storage elements the faults that can be injected vary from the component’s nominal value up to a fully failed component that does not perform at all. In the case of the capacitor, the capacitance would vary from its nominal value down to zero. For parasitic elements such as $R_d$, $R_{ds}$, and $R_{ESR}$, the failure is modeled as a resistance that varies from its nominal value up to several ohms. This takes into account a physical circuit where it is possible to model bad solder joints or other faults as an increased resistance in this leg of the circuit due to excessive heating.

The results of the simulation are generated by sweeping each component failure level from no failure to full failure in 10 discrete steps. The graphs show the error vector of the observer. Furthermore a bounding circle representing the uncertainty due to noise is graphed around the center of the error graph. It is assumed that when the error generated by a failure falls within this circle, then the fault will be hidden within measurement uncertainty, and thus cannot be flagged as an error.

Capacitor faults

Two types of faults are common in capacitors; a common problem in electrolytic capacitors, for example, is the gradual increase of the series resistance
A second form of wear-out that can happen is a decrease in capacitance; for example, in ceramic capacitors the capacitance can decrease over time due to oxide vacancy migration. We can model a capacitance fault as $C(t) = C + \lambda_c(t)$, where $\lambda_c$ goes from zero to $-C$. If we substitute this into Eqn. 2.5 and Eqn. 2.6 it is possible to factor out the error dynamics from the nominal equations. This allows the error dynamics to become

$$\dot{\epsilon} = \begin{bmatrix} 0 & 0 \\ 0 & \frac{1}{C+\lambda_c(t)} - \frac{1}{C} - \frac{1}{C+\lambda_c(t)} \frac{d}{dt} \lambda_c(t) \end{bmatrix} \begin{bmatrix} I_L \\ V_C \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & \frac{1}{C+\lambda_c(t)} \end{bmatrix} \begin{bmatrix} V_d \\ I_{Load} \end{bmatrix}$$

(4.1)

for both matrices in the switched observer. In this case it is easy to see the error dynamics point directly along the direction $V_c$ and are symmetrical around zero. The effectiveness of detecting this is quite low for our converter. If we look at the output of the error filter it can easily be seen that the filter only detects capacitor fault errors that are greater than 90% of its nominal value as seen in Fig. 4.1

A second method in which the capacitor may fail is through increasing ESR. Depending on the capacitor type, this may be due to normal operational wear, sudden failure of the electrodes in the capacitor, or external damage. High levels of failure can cause high levels of ripple on the output, increased heating of components, and increased levels of stress on other components. An ESR fault can be described as $R_{ESR}(t) = R_{ESR} + \lambda_{ESR}(t)$, where $\lambda_{ESR}(t)$ varies with time increasing when a fault occurs. For ESR faults the error
vector becomes

$$\hat{e} = \begin{bmatrix} -\frac{\lambda_{ESR}(t)}{L} & 0 & I_L \\ 0 & 0 & V_C \\ 0 & -\frac{\lambda_{ESR}(t)}{L} & I_{Load} \end{bmatrix} + \begin{bmatrix} 0 \\ V_d \\ I_{Load} \end{bmatrix} \tag{4.2}$$

which is symmetric around an error of zero. However it is interesting to note that the magnitude of the error depends on the output load; thus at low loading of the supply, it becomes progressively harder to detect a fault. For a nominal operating condition of the test supply it is easy to detect ESR failures. Sweeping the ESR from a nominal value to several ohms we can see that almost any deviation outside of a nominal value is outside the noise threshold, as shown in Fig. 4.2.

![Figure 4.2: ESR fault](image)

**Inductor fault**

Inductor faults can permanently damage a power supply; if an inductor winding shorts it can cause overcurrent in the switches, causing them to fail. If a winding-to-winding short occurs, then excessive power is dissipated in the shorted winding causing heat to build up in the inductor causing further damage to the inductor or overheating of components nearby. In the case of inductance change in the inductor, the fault vector becomes Eqn. 4.3 when
$S_1$ is on and becomes Eqn. 4.4 when $S_2$ is on.

\[
\hat{e} = \begin{bmatrix}
\frac{R_{\text{dson}}+R_L+R_{\text{ESR}}}{L+\lambda_L(t)} & \frac{d\lambda_L(t)}{dt} - \frac{1}{L} & -\frac{1}{L+\lambda_L(t)} & 0 \\
0 & 0 & \frac{1}{L+\lambda_L(t)} - \frac{1}{L} & -\frac{R_{\text{ESR}}}{L} \\
\frac{1}{L+\lambda_L(t)} & 0 & \frac{R_{\text{ESR}}}{L+\lambda_L(t)} - \frac{RESR}{L} & 0 \\
0 & 0 & \frac{L+\lambda_L(t)}{L+\lambda_L(t)} - \frac{1}{L} & 0
\end{bmatrix}
\begin{bmatrix}
I_L \\
V_C \\
V_{in} \\
I_{Load}
\end{bmatrix}
\tag{4.3}
\]

Running simulations for the inductor fault shows that for the case of the test converter it is easy to detect inductor faults. Furthermore most small power inductors are built with fewer than 100 turns. And often due to coil geometry, faults will happen either directly across adjacent windings or between winding layers. Thus with such inductors even small failures such as a shorted winding would be detectable with this system. To demonstrate this, Fig. 4.3 shows a simulation of inductor faults ranging from an inductor losing some inductance all the way to being replaced with a circuit element with no inductance. The second type of failure commonly associated with inductors is an increase of the series resistance of the inductor. This can be caused by failures in the wire, manufacturing failures or manufacturing defects in the inductor. This can be modeled by $R_L(t) = R_L + \lambda R_L(t)$. In the case of a fault, $\lambda R_L(t)$ will increase. For purposes of this simulation $R_L$ is varied from its nominal value up to several ohms. In this case the error dynamics become

\[
\hat{e} = \begin{bmatrix}
\frac{R_{\text{d}}+R_L+R_{\text{ESR}}}{L+\lambda_L(t)} & \frac{d\lambda_L(t)}{dt} - \frac{1}{L} & 0 & 0 \\
0 & 0 & \frac{1}{L+\lambda_L(t)} - \frac{1}{L} & -\frac{R_{\text{ESR}}}{L} \\
\frac{1}{L+\lambda_L(t)} & 0 & \frac{R_{\text{ESR}}}{L+\lambda_L(t)} - \frac{RESR}{L} & 0 \\
0 & 0 & \frac{L+\lambda_L(t)}{L+\lambda_L(t)} - \frac{1}{L} & 0
\end{bmatrix}
\begin{bmatrix}
I_L \\
V_C \\
V_{in} \\
I_{Load}
\end{bmatrix}
\tag{4.4}
\]

This fault vector is very similar to the inductor fault vector; however, for both matrices this fault vector points along the same direction instead of being symmetrical around zero as shown in Fig. 4.4.
Diode resistance failure

If the diode or FET used to switch the inductor to ground fails it could short to ground, causing no output and damaging the input transistor. A second failure mode is failing open, this would cause a large negative voltage spike on the input FET. Also increases in the device resistance could signify early stage failures in the device such as lead separation. To model the failure of $S_2$ we can separate out the fault dynamics of the equivalent resistance $R_d$ and it can be shown that these fault dynamics only show up in one switched matrix, $\sigma = 2$, and are present along the direction $I_L$ as

$$
\dot{\hat{e}} = \begin{bmatrix}
\frac{\lambda_{R_{\text{son}}}(t)}{L} & 0 \\
0 & 0
\end{bmatrix}
\begin{bmatrix}
I_L \\
V_C
\end{bmatrix}
$$

(4.6)

The output of the error dynamics shows that detecting changes in the diode resistance between its nominal value and 1 $\Omega$ is accomplished as long as the resistance is above 0.2 $\Omega$ for this circuit and power level as shown in Fig. 4.5.
$S_1$ Failure

Most commonly $S_1$ is a FET transistor driven by gate drive logic signals. Common failures include this switch getting stuck on or off, or if the physical device fails, an increase or decrease of the equivalent resistance of the device. Commonly if this switch fails open then the circuit cannot supply any power to the output but can be replaced if needed; however if the switch fails closed, the circuits connected to the output will get damaged. The fault dynamics
for a switch stuck open are only present in the matrix $\sigma = 1$:

$$\hat{e} = \begin{bmatrix} -\frac{V_{\text{INPUT}}}{L} \\ 0 \end{bmatrix}$$  (4.7)

Similarly if $S_1$ is shorted then the corresponding matrix $\sigma = 2$ produces an error vector

$$\hat{e} = \begin{bmatrix} \frac{V_{\text{INPUT}}}{L} \\ 0 \end{bmatrix}$$  (4.8)

The fault dynamics for a failure causing a resistance increase in $R_{\text{son}}$ only show up in the dynamics of the observer model when operating in $\sigma = 1$ and are dependent on the operating point of the converter:

$$\hat{e} = \begin{bmatrix} \lambda R_{\text{son}}(t) \\ \frac{L}{0} \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} I_L \\ V_C \end{bmatrix}$$  (4.9)

Finally if the FET has an increase in its resistance, then it can be detected if its nominal resistance increases beyond a few hundred milliohms as shown in Fig. 4.6.

![Figure 4.6: $R_{\text{son}}$ fault dynamics](image)

4.1.6 Conclusions of fault detectability

It is shown that the output of an observer based fault detection filter can be used to both detect and characterize both the magnitude and type of fault for individual components in a buck converter. Using this method it
is possible to detect specific levels of faults depending on the magnitude of the fault as well as the operating parameters of the converter and the accuracy of measurements taken by the converter. Not all levels of faults can be detected, and for some components such as the capacitor, detecting any fault but the largest can become difficult for the observer; however, for most converter parameters, variations are at least as large as noise measurements in the example system and are thus detected. Further work that could be conducted in this research could include a study on the reliability of the fault detection filter, studying measurement systems which are low cost, and porting the filter to smaller and lower power computational resources to reduce cost.

4.2 Raw measurements

The raw measurements are read from the converter over a serial and then saved into a Matlab workspace. An example of the raw data that is captured is in Fig. 4.7. This shows all the states of the observer as well as the sample by sample measurements of the converter state as measured by the DSP. Finally the system configuration parameter shows the internal state of the observer state machine which alternates between the different states of the converter $\sigma = [1, 2]$. This allows the processing software to separate out measurements that are for each state. The rest of the measurements shown are error measurements, which is the difference between the measured values and the observer values. Their level of deviation from zero is the error residual, and thus a measure along each individual axis of the failure level of the converter.

4.3 No fault static behavior

Observer performance was measured over a wide output operating range to validate the full operation of the observer without faults. To perform this an automated load was used in conjunction with a Matlab script to switch the load resistance and converter output voltage. For each output voltage and load resistance a sample of 2000 data points of the converter state and
Figure 4.7: Raw measurements of converter operating nominally
observer state was saved. This allows the observation of the state and measurements over a significant period of time. Using the file in Appendix D.1 the load is controlled over GPIB and the measurements and output voltage of the converter are set and gathered using a serial link to the DDSPSP. Using the internal measurements of the converter, the no fault behavior was measured over a subset of 250 samples from the converter while it was switching at several different load levels and output voltages. It was determined that an approximate worst case of noise preset was less than 0.1 V for the capacitor voltage and 0.3 for the inductor voltage as shown in Fig. 4.8.

4.4 Faulty output capacitor

For the faulty output capacitor the ESR was tested as this is the most common observable failure mode and has the most impact on circuit perfor-
Figure 4.9: System dynamics with no faults and load of 10 Ω
Figure 4.10: System dynamics with faulted ESR and load of 2 Ω

mance. Since the simulations have shown above that reducing the output capacitance does not have as appreciable an impact on circuit performance, and that most common failure modes that include sudden or slow capacitor degradation manifest themselves in the form of increased ESR, the test setup only included the output capacitance to be varied. To test this setup the output capacitor was wired in series with common resistors of varying types, sweeping from 2 Ω to 20 Ω. Only a few of the results are shown here in the interest of space, but the full set of measurements is available. It is easy to see in the comparison from the no fault system behavior shown in Fig. 4.8 and 4.9, that when the ESR increases in Fig. 4.10 the error residual increases away from zero. This is exactly as we would hope based on Fig. 4.2. Also as the load to the converter is changed, as is the case in Fig. 4.11, it is important to note that the level of fault is independent of the output load.
Figure 4.11: System dynamics with faulted ESR and load of 10 Ω
4.5 Faulty inductor

The observer performance with respect to the inductor was also tested in a similar manner to the capacitor test. First it was connected in series with a resistor simulating a soft failure of the inductor. This could be caused by a bad solder joint, damage to the windings, etc. However inductor measurements were not as successful as those for the capacitor, namely because the current sensors used had a higher noise floor than the circuit used to measure voltages. This additional noise makes it difficult to distinguish between the different levels of failure. However the average failure errors have been marked by a circuit at their mean value. This allows one to see the progression of component failure by the inductor as shown in Fig. 4.12 and Fig. 4.13. However this graph, while still showing the general trend of the inductor failure in state 1, does not display any clear trend in state 2 due to noise.
Figure 4.13: System dynamics with faulted inductor and load of 10 Ω
CHAPTER 5

CONCLUSIONS

5.1 Successes

This experiment proved that a software based observer can detect hardware faults in real time from simple measurements taken from the hardware and compared with the converter model. The experiments showed that it is possible to detect specific types of component failures during operation. However not all component faults could be accurately detected. Furthermore the converter interface is able to display converter health information in real time, which is extremely helpful for both debugging problems and tuning. After designing two separate observers, one using a course integrator based solver and a second using an explicit solver, the clear winner in terms of accuracy and performance is the explicit solver.

5.2 Failures

While the experiments proved that the technique was useful for detecting certain faults, it was difficult to detect some faults which could manifest in the system and to distinguish certain faults which generate very small fault signature vectors due to noise. These challenges, in combination with expected noise in the combined system, made the detection of capacitor faults, and changes in the series resistances of switches, difficult. Thus faults in certain components are sometimes harder to detect than other faults in the system.
APPENDIX A

SCHEMATICS

Figure A.1 is the full schematic of the converter.
Figure A.1: Full schematic of converter
APPENDIX B

THEORETICAL NOISE OF MEASUREMENTS

The noise of all measurements in the experimental setup must be quantified to determine the minimum level of detectable fault, and also the level of accuracy of all measurements used to estimate the observer state.

To determine the level of noise that would be present in all measurements one can calculate the expected noise contributions from each analog component of the measurement system. For the platform presented, the measurements have several major sources of noise. First, current sensors have built in amplification, which uses a chopper stabilized op-amp to generate an output with less drift but more noise. This is then scaled using variable resistors and then buffered using a discreet op-amp. Finally the result is converted using an ADC before being sent into the software algorithm.

Working backwards from the algorithm, the minimum detectable voltage difference at the ADC is $V_{\text{max}} = \frac{3.3}{4096}$ or 0.8 mV. However this number will get even worse as the effective resolution decreases with smaller sampling windows and higher conversion speeds. According to the electrical characteristics of the ADC peripheral on the DSP, at 100 kHz the SNR is 68 dB and the converter will operate with an ENOB of 10.9. Since the converter may sample output duty cycles as low as 10% of 10 kHz it is within the normal operating range of the ADC to have as few as 100 µs between samples. This gives a minimum detectable voltage at the input of the converter of 1.7 mV above the noise floor.

For optimal system performance the noise level of all components feeding into the ADC should have a noise floor below the ENOB noise floor. This will allow the largest level of inaccuracy to be from the quantization and noise of the ADC and not from external sources.
APPENDIX C
SOURCE CODE

C.1 Ezdsp28335SwitcherMain.c

/*
 Ezdsp28335SwitcherMain.c − Main Program to provide observer based fault detection
 Copyright (c) 2011 Kieran T Levin. All right reserved.

 This program is free software; you can redistribute it and/or
 modify it under the terms of the GNU Lesser General Public
 License as published by the Free Software Foundation; either
 version 2.1 of the License, or (at your option) any later version.

 This library is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 Lesser General Public License for more details.

 You should have received a copy of the GNU Lesser General Public
 License along with this library; if not, write to the Free Software
 Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
 */

#include "DSP28x_Project.h"  // Device Headerfile
#include "switcherlibs.h"
#include "stdio.h"
#include "string.h"

#define STATESTOP 1
#define STATERRUN 2
#define STATEDUMP 4
#define STATERECORD 8

// interrupt void simulation(void);
interrupt void cpu_timer0_isr(void);
interrupt void adc_isr(void);
interrupt void pwm_isr(void);
volatile unsigned char dumpready;

typedef struct storage { 
    float internalstates[11];
}storage;

#pragma DATA_SECTION(myStorage,"ZONE7DATA");
struct storage myStorage [((0x10000/sizeof(storage)))];
int storageCounter;

// float temp1; 
float vcError; 
float ilError; 
char pwm;
```c
void init_zone7(void) {

// Make sure the XINTF clock is enabled
SysCtrlRegs.PCLKCR3.bit.XINTFENCLK = 1;

// Configure the GPIO for XINTF with a 16-bit data bus
// This function is in DSP2833x_Xintf.c
InitXintf16Gpio();

EALLOW;
// All Zones

// Timing for all zones based on XTIMCLK = SYSCLKOUT
XintfRegs.XINTCNF2.bit.XTIMCLK = 0;
// Buffer up to 3 writes
XintfRegs.XINTCNF2.bit.WRBUFF = 3;
// XCLKOUT is enabled
XintfRegs.XINTCNF2.bit.CLKOFF = 0;
// XCLKOUT = XTIMCLK
XintfRegs.XINTCNF2.bit.CLKMODE = 0;

// Zone 7

// When using ready, ACTIVE must be 1 or greater
// Lead must always be 1 or greater
XintfRegs.XTIMING7.bit.XWRLEAD = 1;
XintfRegs.XTIMING7.bit.XRDLEAD = 1;
XintfRegs.XTIMING7.bit.XWRACTIVE = 2;
XintfRegs.XTIMING7.bit.XRDACTIVE = 3;
XintfRegs.XTIMING7.bit.XWRTRAIL = 1;
XintfRegs.XTIMING7.bit.XRDTRAIL = 0;

// don’t double all Zone read/write lead/active/trail timing
XintfRegs.XTIMING7.bit.X2TIMING = 0;

// Zone will not sample XREADY signal
XintfRegs.XTIMING7.bit.USERRDY = 0;
XintfRegs.XTIMING7.bit.USERMDE = 0;

// 1.1 = x16 data bus
// 0.1 = x32 data bus
// other values are reserved
XintfRegs.XTIMING7.bit.XSIZE = 3;
EDIS;

// Force a pipeline flush to ensure that the write to
// the last register configured occurs before returning.
asm(" RPT #7 || NOP");

EALLOW;

// Perform additional configuration of the XINTF for speed up */
XintfRegs.XINTCNF2.bit.XTIMCLK = 0; // XTIMCLK=SYSCLKOUT/1
XintfRegs.XINTCNF2.bit.CLKOFF = 0; // XCLKOUT is enabled
XintfRegs.XINTCNF2.bit.CLKMODE = 0; // XCLKOUT = XTIMCLK
```

// Make sure write buffer is empty before configuring buffering depth
while (XintfRegs.XINTCNF2.bit.WLEVEL != 0) ; // poll the WLEVEL bit
XintfRegs.XINTCNF2.bit.WDBUFF = 0 ; // No write buffering

// Example: Assume Zone 7 is slow, so add additional BCYC cycles whenever
// switching from Zone 7 to another Zone. This will help avoid bus contention.
XintfRegs.XBANK.bit.BCYC = 3 ; // Add 7 cycles
XintfRegs.XBANK.bit.BANK = 3 ; // select zone 7

/* Zone 0 Configuration */
XintfRegs.XTIMING0.bit.X2TIMING = 0 ; // Timing scale factor = 1
XintfRegs.XTIMING0.bit.XSIZE = 3 ; // Always write as 16b
XintfRegs.XTIMING0.bit.BREADYMODE = 1 ; // XREADY is asynchronous
XintfRegs.XTIMING0.bit.BUSERREADY = 0 ; // Disable XREADY
XintfRegs.XTIMING0.bit.XRDLEAD = 1 ; // Read lead time
XintfRegs.XTIMING0.bit.XRDACTIVE = 2 ; // Read active time
XintfRegs.XTIMING0.bit.XRDTRAIL = 1 ; // Read trail time
XintfRegs.XTIMING0.bit.XWRLREAD = 1 ; // Write lead time
XintfRegs.XTIMING0.bit.XWRACTIVE = 2 ; // Write active time
XintfRegs.XTIMING0.bit.XWRTRAIL = 1 ; // Write trail time

/* Zone 6 Configuration */
XintfRegs.XTIMING6.bit.X2TIMING = 0 ; // Timing scale factor = 1
XintfRegs.XTIMING6.bit.XSIZE = 3 ; // Always write as 16b
XintfRegs.XTIMING6.bit.BREADYMODE = 1 ; // XREADY is asynchronous
XintfRegs.XTIMING6.bit.BUSERREADY = 0 ; // Disable XREADY
XintfRegs.XTIMING6.bit.XRDLEAD = 1 ; // Read lead time
XintfRegs.XTIMING6.bit.XRDACTIVE = 2 ; // Read active time
XintfRegs.XTIMING6.bit.XRDTRAIL = 1 ; // Read trail time
XintfRegs.XTIMING6.bit.XWRLREAD = 1 ; // Write lead time
XintfRegs.XTIMING6.bit.XWRACTIVE = 2 ; // Write active time
XintfRegs.XTIMING6.bit.XWRTRAIL = 1 ; // Write trail time

/* Zone 7 Configuration */
XintfRegs.XTIMING7.bit.X2TIMING = 0 ; // Timing scale factor = 1
XintfRegs.XTIMING7.bit.XSIZE = 3 ; // Always write as 16b
XintfRegs.XTIMING7.bit.BREADYMODE = 1 ; // XREADY is asynchronous
XintfRegs.XTIMING7.bit.BUSERREADY = 0 ; // Disable XREADY
XintfRegs.XTIMING7.bit.XRDLEAD = 1 ; // Read lead time
XintfRegs.XTIMING7.bit.XRDACTIVE = 2 ; // Read active time
XintfRegs.XTIMING7.bit.XRDTRAIL = 1 ; // Read trail time
XintfRegs.XTIMING7.bit.XWRLREAD = 1 ; // Write lead time
XintfRegs.XTIMING7.bit.XWRACTIVE = 2 ; // Write active time
XintfRegs.XTIMING7.bit.XWRTRAIL = 1 ; // Write trail time

EDIS;
InitXintf16Gpio();

/* Flush pipeline to ensure that the write is complete. Wait to be sure. */
asm("RPT #6 || NOP");

Istruct Vcontroller;
Istruct Vc;
Istruct I1;
Istruct Itest;
real_T input[4], output[4];
int i = 0;
float prevtime, curtime, deltatime;
volatile float setvoltage, fail;
char shouldfail = 0;
real_T Vinput, Voutput, Vcapacitor, Iinductor, Iinput, Ioutput, Iobserver, Vobserver,
Rload, cd, Iinductorold;
real_T A1[4], A2[4], A1inv[4], A2inv[4], B1[2], B2[2], Result[2], Acurrent[4], Atemp[4],
Estimates[2];
adcringsample + sampleprocessing;
char sbuffer[40];
char sbuffer2[40];
char * warning;
int state, scibyte;
unsigned char errorcurr;
unsigned char errorprev;
/* Running starts here this starts by initializing everything and then going into a
algorithm processing loop */

main()
{
    errorcurr = 0x22;
    errorprev = 0x22;
    processing = FALSE;
    dumpready = FALSE;
    storagecounter = 0;
    vcError = 0;
    ilError = 0;
    state = 0;
    setvoltage = 5;
    adcconverting = FALSE;
    fail = 1;
    // Generate the A1 and A2 Matrices
    A1[0] = 1/INDUCTOR*(−RDSON−RINDUCTOR);
    A1[1] = 0;
    A1[3] = 0;
    A2[0] = 1/INDUCTOR*(−RDIODE−RINDUCTOR);
    A2[1] = 0;
    A2[3] = 0;
    linductorold = 0;

    //set the output buffer to all zeros
    memset(mystorage, 0, sizeof(mystorage));
    // precompute the inverses of these matrices
    Minv2x2(A1, A1inv);
    Minv2x2(A2, A2inv);

    // Initialize hardware:
    // Step 1. Initialize System Control:
    // PLL, WatchDog, enable Peripheral Clocks
    // This example function is found in the DSP2833x_SysCtrl.c file.
    InitSysCtrl();
    // Initialize the external memory for storing waveforms
    init_zone7();
    /* initial ePWM GPIO assignment... */
    config_ePWM_GPIO();
    // only init the pins for the SCI-A port.
    InitSciaGpio();

    // Clear all interrupts and initialize PIE vector table:
    // Disable CPU interrupts
    DINT;

    // Initialize PIE control registers to their default state.
    // The default state is all PIE interrupts disabled and flags
    // are cleared.
    //
    // InitPieCtrl();

    // Disable CPU interrupts and clear all CPU interrupt flags:
    IER = 0x0000;
    IFR = 0x0000;

    // Initialize the PIE vector table with pointers to the shell Interrupt
    // Service Routines (ISR).
    // This will populate the entire table, even if the interrupt
    // is not used in this example. This is useful for debug purposes.
    // The shell ISR routines are found in DSP2833x_DefaultISR.c.
    // This function is found in DSP2833x_PieVect.c.
    InitPieVectTable();

    scia_fifo_init(); // Initialize the SCI FIFO
sci_echoback_init(); // Initialize SCI for echoback
ADC_Init(); // Init ADC control registers
EPWMInit(); // Init PWM control registers

// Setup interrupt functions for hardware
EALLOW; // This is needed to write to EALLOW protected registers
// PieVectTable.TINT0 = &cpu_timer0_isr;
PieVectTable.ADCINT = &adc_isr;
PieVectTable.EPWM1INT = &pwm1_isr;
PieVectTable.SCHRINTA = &sci_isr;

// PieCtrlRegs.PIECTRL.bit.ENPIE = 1; // Enable the PIE block
// PieCtrlRegs.PIEIER9.bit.INTx1=1; // PIE Group 9, INT1
// PieCtrlRegs.PIEIER9.bit.INTx2=1; // PIE Group 9, INT2
// PieCtrlRegs.PIEIER9.bit.INTx3=1; // PIE Group 9, INT3
// PieCtrlRegs.PIEIER9.bit.INTx4=1; // PIE Group 9, INT4
EDIS; // This is needed to disable write to EALLOW protected registers

// Enable CPU INT3 which is connected to EPWM1-6 INT:
EALLOW; // This is needed to write to EALLOW protected registers
// Enable CPU INT3 which is connected to EPWM1-6 INT:
// PieCtrlRegs.PIECTRL.bit.INT3 = 1; // Enable INT3
// PieCtrlRegs.PIEIER3.bit.INT3 = 1; // Enable INT3
// PIE Group 3: INT 1-6
// PieCtrlRegs.PIEIER3.bit.INTx = 3; // Enable INT3
InitCpuTimers();
/* Configure CPU-Timer 0 to interrupt every 1.0E-005 sec. */
/* Parameters: Timer Pointer, CPU Freq in MHz, Period in usec. */
// ConfigCpuTimer(&CpuTimer0, 150.0, SAMPLERATE);
// StartCpuTimer0();

// Configure GPIO32 as a GPIO output pin
EDIS; // This is needed to disable write to EALLOW protected registers
// Configure GPIO32 as a GPIO output pin
EALLOW;
GpioCtrlRegs.GPIOMUX1.bit.GPIO32 = 0;
GpioCtrlRegs.GPIODIR.bit.GPIO32 = 1;

// setup lcd panel pins
GpioCtrlRegs.GPIOMUX2.bit.GPIO63 = 0;
GpioCtrlRegs.GPIOMUX2.bit.GPIO62 = 0;
GpioCtrlRegs.GPIOMUX2.bit.GPIO61 = 0;
GpioCtrlRegs.GPIOMUX2.bit.GPIO60 = 0;
GpioCtrlRegs.GPIOMUX2.bit.GPIO59 = 0;
GpioCtrlRegs.GPIOMUX2.bit.GPIO58 = 0;
GpioCtrlRegs.GPIOMUX2.bit.GPIO57 = 0;
GpioCtrlRegs.GPIOMUX2.bit.GPIO56 = 0;
GpioCtrlRegs.GPIOMUX2.bit.GPIO55 = 0;
GpioCtrlRegs.GPIOMUX2.bit.GPIO54 = 0;
GpioCtrlRegs.GPIOMUX2.bit.GPIO53 = 0;

// setup lcd panel pins directions
GpioCtrlRegs.GPIODIR.bit.GPIO63 = 1;
GpioCtrlRegs.GPIODIR.bit.GPIO62 = 1;
GpioCtrlRegs.GPIODIR.bit.GPIO61 = 1;
GpioCtrlRegs.GPIODIR.bit.GPIO60 = 1;
GpioCtrlRegs.GPIODIR.bit.GPIO59 = 1;
GpioCtrlRegs.GPIODIR.bit.GPIO58 = 1;
GpioCtrlRegs.GPIODIR.bit.GPIO57 = 1;
GpioCtrlRegs.GPIODIR.bit.GPIO56 = 1;
GpioCtrlRegs.GPIODIR.bit.GPIO55 = 1;
GpioCtrlRegs.GPIODIR.bit.GPIO54 = 1;
GpioCtrlRegs.GPIODIR.bit.GPIO53 = 1;
GpioCtrlRegs.GPIODIR.bit.GPIO52 = 1;
EDIS;
DELAY_US(100); // Init controller on LCD
LCDinit(20, 2);
DELAY_US(100);
// LCDinit(20, 2);
LCDPrint("Converter Nominal",">90% Functional");
while(1) {
    LCDwrite('A'); DELAY_US(100);
    // Enable CPU int1 which is connected to CPU-Timer 0, CPU int13
    // which is connected to CPU-Timer 1, and CPU int 14, which is connected
    // to CPU-Timer 2:
    IER |= M_INT1;
    // Enable TINT0 in the PIE; Group 1 interrupt 7
    PieCtrlRegs.PIESEL1.bit.INTx7 = 1;
    // Enable global Interrupts and higher priority real-time debug events:
    EINT; // Enable Global interrupt INTM
    EIR; // Enable Global realtime interrupt DBG
    // Init the data structure handling passing data from sensors to algorithm
    initRingbuf();
    // Init PID controller for output voltage control
    Vcontroller.Igain = 15;
    Vcontroller.Integral = 0;
    Vcontroller.ssaturation = TRUE;
    Vcontroller.satHI = 0.97;
    Vcontroller.satLO = 0.1;
    // In old algorithm - init integrator for Vc
    Vcontroller.Igain = 3;
    Vc.Integral = 0;
    Ic.InputPrev = 0;
    Vc.ssaturation = TRUE;
    Vc.satHI = 500;
    Vc.satLO = -500;
    Ic.Igain = 1;
    Ic.Integral = 0;
    Ic.InputPrev = 0;
    Ic.ssaturation = TRUE;
    Ic.satHI = 500;
    Ic.satLO = -500;
    Itest.Igain = 1;
    Itest.Integral = 0;
    Itest.ssaturation = FALSE;
    Itest.satHI = 500;
    Itest.satLO = -65530;
    Iloobserver = 0;
    Vcobserver = 0;
    // send init done out serial port
    print("\r\nInit done!\r\n");
    EPWMset(1,1,0.5);
    while(1) {
        if (adcRingsavailable() & (state == STATERUN) || (state == STATERECORD)) {
            sampleprocessing = popRingElem();
            // Do all our element processing here:
            currtime = sampleprocessing->time;
            deltatime = currtime - prevtime;
            if (deltatime < 0) deltatime += 1e-4; // if we start a new cycle - we want to add
            // on the time
            // for the last cycle so we have a good time update
            input = VOLT2AMP * (DCOFFSET - (ADC2VOLT * sampleprocessing->samples[3]));
            output = VOLT2AMP * (DCOFFSET - (ADC2VOLT * sampleprocessing->samples[0]));
            inductor = VOLT2AMP * (DCOFFSET - (ADC2VOLT * sampleprocessing->samples[2]));
            Vinput = ADC2VOLTOUT * sampleprocessing->samples[4];
            Voutput = ADC2VOLTOUT * sampleprocessing->samples[1];
            Vcapacitor = Voutput - RESR*(inductor->output);
            Rload = Voutput/output;
            cd = Rload/(Rload + RESR);
            // calculate the ripple
            inductor = inductor + (inductor - inductorold)/deltatime * 0.0000001;
            inductorold = inductor;
            if (abs(inductorold) > 10) inductorold = 0;
            // ////////// Test values
    }
/--------------------------------------------------------------------*/
** Inductor = 2; **
** Vcap = 5; **
** Voutput = 5; **
** Iobserver = 2; **
** Vobserver = 5; **
** Ioutput = 1; **
** Iinput = 0; **
** Vinput = 10; **
** delta = 1e-7; **
** sampleprocessing->rawtime = 5; **
** Rload = Voutput / Ioutput; **
** cd = Rload / (Rload + RESR); **

// ///////////
if (sampleprocessing>rawtime < 100)
{
//we just finished a time when S1 is off so use sigma=2 matrices
A1[0] = ((-RDSON - RINDUCTOR - cd*RESR)*INDUCTOR);
B1[0] = Vinput*INDUCTOR;
B1[1] = 0;
}
else {
A1[0] = ((-VDiode - RINDUCTOR - cd*RESR)*INDUCTOR);
B1[0] = (-VDiode*INDUCTOR);
B1[1] = 0;
}
Mmpy2x2(A1, deltatime, Atemp); //multiply A*time
// find e'*At
Mexp2x2(Atemp, A2);
B2[0] = Iobserver;
B2[1] = Vobserver; //generate X(0)
Mmpy2x1(A2, B2, Result); //put expm(A*Time)*[Iobs;Vcobs] into result
Minv2x2(A1, A1inv); //calculate inverse
B2[0] = (Inductor - Iobserver);
Atemp[0] = exp(20*deltatime);
Atemp[1] = 0;
Atemp[2] = 0;
Atemp[3] = Atemp[0];
Mmpy2x1(Atemp, B2, A1);
Macc2x1(A1, Result); // + expm(20*eye(2)*Time)*[I1;Vc] = [Iobs;Vcobs]
//now also save A3 until we add it back in
Atemp[0] = -1;
Atemp[3] = -1; // = eye(2)
Macc2x2(A2, Atemp); //expm(A*Time) - eye(2)
Mmpy2x1(Atemp, B1, A2); //expm(A*Time) - eye(2)*B
Mmpy2x1(A1inv, A2, Atemp);
Macc2x1(Atemp, Result); //add into result
//calculate the error
Iobserver = Result[0];
Vobserver = Result[1];
if (Iobserver > 100) Iobserver = 100;
if (Vobserver > 100) Vobserver = 100;
//our error is sitting in B2
if (storagecounter < (0x010000/(sizeof(storage)))){
    if (shouldfail){
        if (storagecounter == (0x010000/(sizeof(storage)/2))){
            fail = 1;
        }
    }
}
mystorage[storagecounter].internalstates[0] = (float)B2[0];
mystorage[storagecounter].internalstates[1] = (float)B2[1];

45
mystorage[storagecounter].internalstates[2] = (float)((sampleprocessing->rawtime < 100)?2:1);
mystorage[storagecounter].internalstates[3] = Inductor;
mystorage[storagecounter].internalstates[8] = Voutput;
mystorage[storagecounter].internalstates[9] = Inoutput;
mystorage[storagecounter].internalstates[10] = deltatime; // prints in higher res to computer
storagecounter++;
}
errorcurr = Error2Zone(B2[0], B2[1]);
if (sampleprocessing->rawtime < 100) {// note this is backwards from above
    warning = "Nominal ";
    if (errorcurr != 0x22 && errorprev == 0x21) {
        warning = "Cap Fault ";
    } else if (errorcurr == 0x41 && errorprev == 0x14) {
        warning = "Rd Fault ";
    } else if (errorcurr == 0x24 && errorprev == 0x14) {
        warning = "Rson F ";
    } else if (errorcurr == 0x42) {
        warning = "RInductor ";
    }
} else {// s2 is on
    LCDupdate(); // pet the led display pushing one char out at a time // tick a bit
    if (sampleprocessing->rawtime < 100) {
        i++;
        if (i == 500) {
            // calculate some error estimates into component failures
            sprintf(sbuffer, "EL%f ", B2[0]);
            sprintf(sbuffer+11, "%s ", warning);
            sprintf(sbuffer2, "EC%f ", B2[1]);
            sprintf(sbuffer2+11, "%X %X ", errorcurr, errorprev);
            LCDPrint(sbuffer, sbuffer2);
        }
        errorprev = errorcurr;
        prevtime = currtime;
    }
}

// check for button presses
if (scia_has_byte()) {
    scibyte = scia_get_byte();
    switch (scibyte) {
    case 's':
        state = STATESTOP;
        shouldfail = 0;
        EPWMset(1, 1, 0);
        break;
    case 'g':
        state = STATERUN;
        shouldfail = 0;
        fail = 0;
        EPWMset(1, 1, 0.5);
        while (adc_readingAvailable())
            popRingElem(); // flush old readings from the ring
        break;
    }
case 'd':
    state = STATEDUMP;
    shouldfail = 0;
    EPWMset(1, 1, 0);
    break;

case 'r': //record
    //state = STATETRAN;
    state = STATETRANS;
    shouldfail = 0;
    fail = 0;
    storagecounter = 0;
    break;

case 'f': //set to fail for first half of recording
    state = STATETRANS;
    shouldfail = 1;
    fail = 0;
    storagecounter = 0;
    //EPWMset(1, fail, 0.5);
    break;

default:
    break;

}

if (scibyte > '0' && scibyte < '9'){
    setvoltage = (float)(scibyte - '0'); //convert from ascii to decimal
    //EPWMset(1, setvoltage);
}

} //dump out all of our stored buffer to the computer through the serial port
if (STATEDUMP == state)
{
    for (storagecounter = 0; storagecounter < (0x010000/(sizeof(storage)));
        storagecounter++)
    {
        for (i = 0; i < 10; i++)
        {
            printDouble((mystorage[storagecounter].internalstates[i]),4);
            print(",");
        }
        printDouble((mystorage[storagecounter].internalstates[10]),4);
        //print times in higher resolution to computer
        print("\n");
        //storagecounter = 0;
        print("Section End\n");
        state = STATETRAN;
    }
}

//float testfloat;
//float ilprev;
//float vcprev;
//float Rrefaverage;
//float Cd;

interrupt void cpu_timer0_isr(void){
    //unsigned char smallcount;
    //toggle our TIMER0 gpio to show the word that we are running
    GpioDataRegs.GPBSET.bit.GPIO33 = 1;
    
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
    //////////////////////////////////////////////////////////////////////////
    //////////////////////////////////////////////////////////////////////////
    //Do all our heavy processing here....
    
    GpioDataRegs.GPCLEAR.bit.GPIO33 = 1;
}

int time;
interrupt void adc_isr(void) {
    adcringsample mysample;
    mysample.samples[0] = AdcMirror.ADCRESULT0;
    mysample.samples[1] = AdcMirror.ADCRESULT1;
    mysample.samples[2] = AdcMirror.ADCRESULT2;
    mysample.samples[3] = AdcMirror.ADCRESULT3;
    mysample.samples[4] = AdcMirror.ADCRESULT4;
    mysample.samples[5] = AdcMirror.ADCRESULT5;
    mysample.samples[7] = AdcMirror.ADCRESULT7;
    mysample.valid = missedsample;
    mysample.time = time ∗ 6.666667e−9;
    mysample.rawtime = time;
    //memcpy(adcring, , AdcMirror.ADCRESULT0, sizeof(AdcMirror.ADCRESULT0)∗8);
    insertRingElem(&mysample);
    //if the pwm triggered and missed a sample, we should register that to our datastructure, so down the road we can figure out what happened..
    if (TRUE == missedsample)
    {
        //TODO: Shove some crap into the ring buffer to show missing sample
        missedsample = FALSE;
    }
    adconverting = FALSE;
    // Reinitialize for next ADC sequence
    AdcRegs.ADCTRL2.bit.RST_SEQ1 = 1;    // Reset SEQ1
    AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1;  // Clear INT SEQ1 bit
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
    //update our pid control loop
    if (200 > time)
    {
        if (state == STATERUN)
        {
            EPWMset(fail, 1, Mathl(&Vcontroller, setvoltage = ADCVOLTOUT ∗ mysample.samples[1]));
        }
        //else
        //EPWMset(1, 1, 0);
    }
    return;
}

interrupt void pwm3_isr(void) {
    time = EPwm3Regs.TBCTR;
    //check if the adc is currently in the middle of a conversion
    //this happens if the pwm is set below 70% or above 90%
    if (FALSE == AdcRegs.ADCST.bit.SEQ1_BSY) {
        //start ADC conversion
        AdcRegs.ADCTRL2.bit.SOC_SEQ1 = 1;
        //Clear interrupt flag
        adconverting = TRUE;
    }
    else {
        //we missed a sample bc the adc is too slow o crap...probably the pwm duty is near 0% or 100%
        missedsample = TRUE;
    }
    //alternate what we trigger on, so if we triggered on CMPA next time we
    //want to trigger on CMPB next time
    if ( EPwm3Regs.ETSEL.bit.INTSEL == 0x01) {
      EPwm3Regs.ETSEL.bit.INTSEL = 0x04;
    }
    else {
      EPwm3Regs.ETSEL.bit.INTSEL = 0x01;
    }
    //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;
    return;
}
C.2 Adc2VA.h

/*
adc2va.h - Base class that provides raw adc values to voltages and currents
Copyright (c) 2011 Kieran T Levin. All right reserved.

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
*/
#endif
#define ADC2VAH
#define ADC2VAH
#include "DSP28x_Project.h" // Device Headerfile
#include "switcherlibs.h"

#define ADC2VOLT (0.001220703125) // (5/4096)
#define DCOFFSET (2.500)
#define ADC2VOLTOUT (0.009765625) // (40/4096)
#define ADC2VOLTIN (0.009765625) // (40/4096)

extern float INCHAN[16];

#define IOUT (0)
#define VOUT (1)
#define IIND (2)
#define VIN (4)

void processchannels(void);

float getchan(unsigned char chan);

#endif // end of definition

C.3 Adc2VA.c

/*
Adc2VA.c - Base class that provides conversions from raw adc to real world values
Copyright (c) 2011 Kieran T Levin. All right reserved.

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
*/
float INCHAN[16];

void process_channels(void)
{
    INCHAN[0] = VOLT2AMP * (DCOFFSET - (ADC2VOLT * RAWADC[0])); //IOUT
    INCHAN[1] = ADC2VOLTOUT * RAWADC[1]; //VOUT
    //Convert this to Vc as we are measuring Vc + Vser
    INCHAN[2] = VOLT2AMP * (DCOFFSET - (ADC2VOLT * RAWADC[2])); //I_L
    INCHAN[3] = VOLT2AMP * (DCOFFSET - (ADC2VOLT * RAWADC[3])); //I_in
    INCHAN[4] = ADC2VOLTOUT * RAWADC[4]; //VIN
    //INCHAN[5] = 0;
    //INCHAN[6] = 0;
    //INCHAN[7] = 0;
}

float getchan(unsigned char chan)
{
    return INCHAN[chan];
}

C.4 Adclib.h

/*
 Adclib.h – Base class that provides hardware interface to adc
 Copyright (c) 2011 Kieran T Levin. All right reserved.

 This library is free software; you can redistribute it and/or
 modify it under the terms of the GNU Lesser General Public
 License as published by the Free Software Foundation; either
 version 2.1 of the License, or (at your option) any later version.

 This library is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 Lesser General Public License for more details.

 You should have received a copy of the GNU Lesser General Public
 License along with this library; if not, write to the Free Software
 Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
 */
#ifndef ADCLIBS_H
#define ADCLIBS_H
#include "DS28x_Project.h" // Device Headerfile
#include "switcherlibs.h"

extern volatile double RAWADC[16];
extern volatile unsigned char processing;

void ADCInit(void);
double ADCRead(unsigned char chan);
void ADCsample(void);
void ADCEnChan(unsigned char chan);
void ADCDisChan(unsigned char chan);
#endif // end of definition
C.5 Adclib.c

/**
   Adclib.c - Base class that provides hardware setup for adc
   Copyright (c) 2011 Kieran T Levin. All right reserved.
   Based on TI examples...

   This library is free software; you can redistribute it and/or
   modify it under the terms of the GNU Lesser General Public
   License as published by the Free Software Foundation; either
   version 2.1 of the License, or (at your option) any later version.

   This library is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
   Lesser General Public License for more details.

   You should have received a copy of the GNU Lesser General Public
   License along with this library; if not, write to the Free Software
   Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
*/
#include "Adclib.h"
volatile double RAWADC[16];
volatile unsigned char processing;

// For now only converts on lower 8 channels
void ADC_Init(void)
{
    // ISR functions found within this file.
    EALLOW; // This is needed to write to EALLOW protected register
    // PaeVectTable.ADCINT = BADCInterrupt;
    // PaeVectTable.ADCINT = Dsimulation;
    EDIS; // This is needed to disable write to EALLOW protected registers

    InitAdc(); // For this example, init the ADC
    // Enable ADCINT in PIE
    PieCtrlRegs.PIEIER1.bit.INTx6 = 1;
    PIE |= INT1; // Enable CPU Interrupt 1
    EINT; // Enable Global interrupt INTM
    EIRM; // Enable Global real time interrupt DBGM

    // Configure ADC
    /*
    AdcRegs.ADCTRL3.bit.SMODE_SEL = 1; // setup for simultaneous mode
    AdcRegs.ADCTRL1.bit.CONT_RUN = 0; //setup for continuous conversion
    AdcRegs.ADCTRL1.bit.AQPS = 0x7; //acquisition window size
    AdcRegs.ADCTRL1.bit.SEQOVRD = 0;
    
    AdcRegs.ADCTRL2.bit.EPWM_SOCSEQ = 0; // disable epwm trigger SOC
    AdcRegs.ADCTRL2.bit.INT_ENA_SEQ1 = 1; //enable sequencer interrupt
    AdcRegs.ADCTRL2.bit.INT_MOD_SEQ1 = 0; //interrupt after each sequencer finishes
    AdcRegs.ADCTRL2.bit.EPWM_SOCSEQ1 = 0;
    AdcRegs.ADCTRL2.bit.EXT_SOCSEQ1 = 0;
    AdcRegs.ADCTRL2.bit.SOC_SEQ2 = 0; //SOC for seq2 clears trigger
    AdcRegs.ADCTRL3.bit.ADCLKPS = 0x01; //axis clock divider
    AdcRegs.ADCMAXCONV.all = 0x0004; // Setup 2 conv's on SEQ1
    */
    //AdcRegs.ADCTRL2.bit.INT_ENA_SEQ1 = 1; //enable sequencer interrupt
    //AdcRegs.ADCTRL2.bit.INT_MOD_SEQ1 = 0; //interrupt after each sequencer finishes

    AdcRegs.ADCTRL1.bit.SUSMOD = 0x0; // Emulation suspend ignored
    AdcRegs.ADCTRL1.bit.AQPS = 3; // Acquisition window size
    AdcRegs.ADCTRL1.bit.CPS = 0; // Core clock prescaler was 1
    AdcRegs.ADCTRL3.bit.ADCCLKPS = 3; // Core clock divider was 3
    AdcRegs.ADCRESSEL.bit.REF_SEL = 0; // Set Reference Voltage
    AdcRegs.ADCOFFTRIM.bit.OFFSET_TRIM = 0; // Set Offset Error Correctino Value
    AdcRegs.ADCTRL1.bit.CONT_RUN = 0; // 0: Start-Stop or continuous sequencer mode
    AdcRegs.ADCTRL3.bit.ADCBGRFDN = 0x3; // Bandgap and reference powered up

    */
AdcRegs.ADCCTRL1.bit.SM0DR_SEL = 0;  // 1: Simultaneous, 0: Sequential sampling
AdcRegs.ADCMAXCONV.bit.MAX_CONV1 = 4;  // Number of conversions in CONV2 when using B module
AdcRegs.ADCCTRL1.bit.SEQ_CONC = 0;  // 1: Cascaded, 0: Dual sequencer mode
AdcRegs.ADCCHSELSEQ1.all = 12816U;  // Channels for conversion
AdcRegs.ADCCTRL2.bit.RST_SEQ1 = 0x1;  // Reset SEQ1

AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0x0;  // Setup ADCINA3 as 1st SEQ1 conv.
AdcRegs.ADCCHSELSEQ1.bit.CONV01 = 0x1;  // Setup ADCINA2 as 2nd SEQ1 conv.
AdcRegs.ADCCHSELSEQ1.bit.CONV02 = 0x2;  // Setup ADCINA3 as 1st SEQ1 conv.
AdcRegs.ADCCHSELSEQ1.bit.CONV03 = 0x3;  // Setup ADCINA2 as 2nd SEQ1 conv.

AdcRegs.ADCCHSELSEQ2.bit.CONV04 = 0x4;  // Setup ADCINA2 as 2nd SEQ1 conv.
AdcRegs.ADCCHSELSEQ2.bit.CONV05 = 0x5;  // Setup ADCINA2 as 2nd SEQ1 conv.
AdcRegs.ADCCHSELSEQ2.bit.CONV06 = 0x6;  // Setup ADCINA2 as 2nd SEQ1 conv.
AdcRegs.ADCCHSELSEQ2.bit.CONV07 = 0x7;  // Setup ADCINA2 as 2nd SEQ1 conv.

AdcRegs.ADCCTRL2.bit.EPWM_SOCA_SEQ1 = 1;  // Enable SOCA from ePWM to start SEQ1
AdcRegs.ADCCTRL2.bit.INT_ENA_SEQ1 = 1;  // Enable SEQ1 interrupt (every EOS)
AdcRegs.ADCCTRL2.bit.INT_MOD_SEQ1 = 0;  // Interrupt after every conversion

// Assumes ePWM1 clock is already enabled in InitSysCtrl()
/* EPwm1Regs.ETSEL.bit.SOCAEN = 1;  // Enable SOC on A group
EPwm1Regs.ETSEL.bit.SOCASEL = 4;  // Select SOC from from CPMA on upcount
EPwm1Regs.ETPS.bit.SOCAPRD = 1;  // Generate pulse on 1st event
EPwm1Regs.CMPA half.CMPA = 0x0080;  // Set compare A value
EPwm1Regs.TBPRD = 0xFFFF;  // Set period for ePWM1
EPwm1Regs.TBCTL.bit.CTRMODE = 0;  // count up and start */

double ADCRead(unsigned char chan){
    if (chan > 15)
        return -1;
    return RAWADC[chan];
    // r e t u r n *(AdcRegs.ADCRESULT0+chan);
}

void ADCSample(void){
    // trigger sample
    AdcRegs.ADCCTRL2.bit.SOC_SEQ1 = 1;
    // while (AdcRegs.ADCST.bit.INT_SEQ1 == 0);
    // asm(" RPT#11 || NOP");
    // ADCInterrupt();
}

unsigned char ADCNumChanEn = 0;
void ADCEnChan(unsigned char chan){
    ADCNumChanEn++;
    AdcRegs.ADCMAXCONV.all = ADCNumChanEn;
    AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0x3;  // Setup ADCINA3 as 1st SEQ1 conv.
}

void ADCDisChan(unsigned char chan){
    ADCNumChanEn--;
    AdcRegs.ADCMAXCONV.all = ADCNumChanEn;
}
C.6 Epwmlib.h

/*
  epwmlib.h — Base class that provides hardware interface to pwm
Copyright (c) 2011 Kieran T Levin. All right reserved.

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
*/
#ifndef EPWMLIB_H
#define EPWMLIB_H

#include "DSP28x_Project.h" // Device Headerfile
#include "switcherlib.h"

#define PWM_MAX 7500+2
#define PWM_MIN 0

/** PWM Parameter Structure */
typedef struct_EPWMPARAMS {
  /*—- Time—Base (TB) Submodule ——*/
  Uint16 TBPRD;
  Uint16 TBCTL_CTRMODE;
  Uint16 TBCTL_SYNCOSEL;
  Uint16 TBCTL_PHESEN;
  Uint16 TBCTL_PHSDIR;
  Uint16 TBPHS;
  Uint16 TBCTL_HSPCLKDIV;
  Uint16 TBCTL_CLKDIV;

  /—— Counter—Compare (CC) Submodule ——*/
  Uint16 CMPCTL_LOADAMODE;
  Uint16 CMPCTL_LOADBMODE;
  Uint16 CMPA;
  Uint16 CMPB;

  /—— Action—Qualifier (AQ) Submodule ——*/
  Uint16 AQCTLA;
  Uint16 AQCTLB;
  Uint16 AQCSFRC_CSFA;
  Uint16 AQCSFRC_CSFB;
  Uint16 AQCSFRC_RLDCSF;

  /—— Dead—Band Generator (DB) Submodule ——*/
  Uint16 DBCTL_OUTMODE;
  Uint16 DBCTL_INMODE;
  Uint16 DBCTL_POLSEL;
  Uint16 DBRED;
  Uint16 DBRED;

  /—— Event—Trigger (ET) Submodule ——*/
  Uint16 ETSEL_SOCASEN;
  Uint16 ETSEL_SOCASEL;
  Uint16 ETPS_SOCAPRD;
  Uint16 ETSEL_SOCBEN;
  Uint16 ETSEL_SOCBSEL;
  Uint16 ETPS_SOCBPRD;
  Uint16 ETSEL_INTEN;
  Uint16 ETSEL_INSEL;
  Uint16 ETPS_INTPRD;
}
/∗−− PWM-Chopper (PC) Submodule −−∗/
Uint16 PCCTL_CHPEN;
Uint16 PCCTL_CHPFREQ;
Uint16 PCCTL_OSHIWT;
Uint16 PCCTL_CHPDUTY;

/*−− Trip-Zone (TZ) Submodule −−∗/
Uint16 TZSEL;
Uint16 TZCTL_TZA;
Uint16 TZCTL_TZB;
Uint16 TZEINT_OST;
Uint16 TZEINT_CBC;
} EPWMPARAMS;

void config_ePWM_GPIO ( void );

void EPWMinit ( void );

void EPWMset ( float chan1 , float chan2 , float chan3 );

void config_ePWMRegs ( volatile struct EPWM_REGS∗ EPwmRegs, EPWMPARAMS∗ EPwmParams );

#endif // end of definition

C.7 Epwmlib.c

// epwmlib.c
// READ SPRU791F for info about this……
/*
epwmlib.c - Base class that provides hardware interface to pwm
Copyright (c) 2011 Kieran T Levin. All right reserved.

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
*/

// based on ti examples
#include "Epwmlib.h"
void config_ePWM_GPIO ( void )
{
EALLOW;

/*−− Configure pin assignments for ePWM −−*/
GpioCtrlRegs.GPAPUD.bit.GPIO4 = 0; // Enable pull-up on GPIO4 (EPWM3A)
GpioCtrlRegs.GPAPUD.bit.GPIO5 = 0; // Enable pull-up on GPIO5 (EPWM3B)
GpioCtrlRegs.GPAMUX1.bit.GPIO4 = 1; // Configure GPIO4 as EPWM3A
GpioCtrlRegs.GPAMUX1.bit.GPIO5 = 1; // Configure GPIO5 as EPWM3B

/*−− Configure pin assignments for ePWM −−*/
GpioCtrlRegs.GPAPUD.bit.GPIO2 = 0; // Enable pull-up on GPIO2 (EPWM2A)
GpioCtrlRegs.GPAPUD.bit.GPIO3 = 0; // Enable pull-up on GPIO3 (EPWM2B)
GpioCtrlRegs.GPAMUX1.bit.GPIO2 = 1; // Configure GPIO2 as EPWM2A
GpioCtrlRegs.GPAMUX1.bit.GPIO3 = 1; // Configure GPIO3 as EPWM2B

54
/** Configure pin assignments for ePWM */
GpioCtrlRegs.GPAPUD.bit.GPIO0 = 0;  // Enable pull-up on GPIO0 (EPWMIA)
GpioCtrlRegs.GPAPUD.bit.GPIO1 = 0;  // Enable pull-up on GPIO1 (EPWMIB)
GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 1;  // Configure GPIO0 as EPWMIA
GpioCtrlRegs.GPAMUX1.bit.GPIO1 = 1;  // Configure GPIO1 as EPWMIB
GpioCtrlRegs.GPBFUD.bit.GPIO32 = 0;  // Enable pull-up on GPIO32 (zSYNCI).
GpioCtrlRegs.GPBFUD.bit.GPIO32 = 2;  // Configure GPIO32 as zSYNCI.
GpioCtrlRegs.GPBFUD.bit.GPIO33 = 0;  // Enable pull-up on GPIO33 (zSYNCO).
GpioCtrlRegs.GPBFUMUX1.bit.GPIO33 = 2;  // Configure GPIO33 as zSYNCO.

/** Configure pin assignments for TZn */

/* Function: configEPwMRegs */
void configEPwMRegs (volatile struct EPwmRegs *EPwmRegs,
volatile struct EPwmParams *EPwmParams)
{
  /*** Setup Time-Base (TB) Submodule ***/
EPwmRegs->TBPRD = EPwmParams->TBPRD;
EPwmRegs->CMPPHIB = EPwmParams->CMPPHIB;
EPwmRegs->CMPPHIF = EPwmParams->CMPPHIF;
EPwmRegs->CMPPHIB = EPwmParams->CMPPHIB;

  /*** Setup Counter, Compare (CC) Submodule ***/
EPwmRegs->CMPPCTL.bit.SWDWMODE = CC_SSHADOW;  // Always enable shadow mode, no immediate
EPwmRegs->CMPPCTL.bit.SWDWMODE = CC_SSHADOW;  // Always enable shadow mode, no immediate
EPwmRegs->CMPPCTL.bit.HSPCLKDIV = EPwmParams->CMPPCTRL.HSPCLKDIV;
EPwmRegs->CBCTR = 0x0000;  // Clear counter

  /*** Setup Action-Qualifier (AQ) Submodule ***/
EPwmRegs->AQCSFRC.bit.RLLCSF = EPwmParams->AQCSFRC.RLLCSF;
EPwmRegs->AQCSFRC.bit.CSPB = EPwmParams->AQCSFRC.CSPB;

  /*** Setup Dead-Band Generator (DB) Submodule ***/
EPwmRegs->DBRED = EPwmParams->DBRED;
EPwmRegs->DBRED = EPwmParams->DBRED;

  /*** Setup Event-Trigger (ET) Submodule ***/
EPwmRegs->ETSEL.bit.SOCKEN = EPwmParams->ETSEL.SOCKEN;
EPwmRegs->ETSEL.bit.SOCKASEL = EPwmParams->ETSEL.SOCKASEL;
EPwmRegs->ETSEL.bit.INTEN = EPwmParams->ETSEL.INTEN;
EPwmRegs->ETPS.bit.INTFPRD = EPwmParams->ETPS.INTFPRD;

  /*** Setup PWM-Chopper (PC) Submodule ***/
EPwmRegs->PCCTRL.bit.CHFREQ = EPwm Params->PCCTRL.CHFREQ;
EPwmRegs->PCCTRL.bit.CHFRQ = EPwm Params->PCCTRL.CHFRQ;

55
void EPWMinit(void)
{
/* Start for S-Function (c280xpwm): '<S>/dPWM' */

/** Initialize dPWM3 modules **/
{
    EPwmRegs EPwm3Params;
    /* Setup Time-Base (TB) Submodule */
    EPwm3Params.TBPRD = PWM_MAX;  // period of time base counter, sets frequency
    EPwm3Params.TBCTL.CTRMODE = 0;  // up count mode
    EPwm3Params.TBCTL.SYNCOSEL = 3;  // disable synchronization select
    EPwm3Params.TBCTL.PHSEN = 0;  // do not load from phase register
    EPwm3Params.TBCTL.PHSDIR = 0;  // phase direction set
    EPwm3Params.TBPHS = 0;  // sets the time base phase
    EPwm3Params.TBCTL.HSPCLKDIV = 0;  // high speed time base clock prescale = /1
    EPwm3Params.TBCTL.CLKDIV = 0;  // time base clock prescale = /1

    /* Setup Counter_Compare (CC) Submodule */
    EPwm3Params.CMPCTL.CTMODE = 0;  // load 0 into counter on compare true
    EPwm3Params.CMPCTL.LOADAMODE = 0;  // load 0 into counter on compare true
    EPwm3Params.CMPA = 0;  // The value in the active CMPA register is continuously compared to the time-base counter (TBCTR). When the values are equal, the counter-compare module generates a "time-base counter equal to counter compare A" event.
    EPwm3Params.CMPB = 32000;

    /* Setup Action_Qualifier (AQ) Submodule */
    EPwm3Params.AQCTLA = 0x24;  // was dec36 // set output high when compare equal and zero at timer overflow value
    EPwm3Params.AQCTLB = 0x108;  // was dec264
    EPwm3Params.AQCSFRQ.CSFA = 0;  //
    EPwm3Params.AQCSFRQ.CSFBO = 0;  //
    EPwm3Params.AQCSFRQ.RLDCSFP = 0;

    /* Setup Dead-Band Generator (DB) Submodule */
    EPwm3Params.DBCTL.OUT_MODE = 0;
    EPwm3Params.DBCTL.IN_MODE = 0;
    EPwm3Params.DBCTL.POLSEL = 0;
    EPwm3Params.DBFED = 0.0;
    EPwm3Params.DBFED = 0.0;

    /* Setup Event_Trigger (ET) Submodule */
    EPwm3Params.ETSEL.SOCASEL = 1;
    EPwm3Params.ETSEL.SOCASEL = 0;
    EPwm3Params.ETPS.SOCAPRD = 1;
    EPwm3Params.ETPS.SOCAPRD = 0;
    EPwm3Params.ETSEL.SOCHEN = 0;
    EPwm3Params.ETSEL.SOCHEN = 1;
    EPwm3Params.ETPS.SOCPRD = 1;
    EPwm3Params.ETPS.SOCPRD = 0;
    EPwm3Params.ETPS.INTEN = 1;
    EPwm3Params.ETPS.INTEN = 0;
    EPwm3Params.ETPS.INTPRD = 1;

    /* Setup PWM-Chopper (PC) Submodule */
    EPwm3Params.PCCTL.CHPEN = 0;
    EPwm3Params.PCCTL.CHPFREQ = 0;
    EPwm3Params.PCCTL.CHPTWTH = 0;
    EPwm3Params.PCCTL.CHPDUTY = 0;

    /* Setup Trip-Zone (TZ) Submodule */
EPwm3Params.TZSEL = 0;
EPwm3Params.TZCTL.TZA = 3;
EPwm3Params.TZCTL.TZB = 3;
EPwm3Params.TZEINT.OST = 0;
EPwm3Params.TZEINT.CBC = 0;

/** Initial ePWM3 **/
    config_ePWMRegs(&EPwm3Regs, &EPwm3Params);
} }

/* Start for S-Function (c380xpwm): '<S>/ePWM1' */

/***** Initialize ePWM2 modules *****/

EPWM_PARAMS EPwm2Params;

/**< Setup Time-Base (TB) Submodule **/
EPwm2Params.TBPRD = PWM_MAX;
EPwm2Params.TBCTL.CTRMODE = 0;
EPwm2Params.TBCTL.SYNCOSEL = 3;
EPwm2Params.TBCTL.PHSEN = 0;
EPwm2Params.TBCTL.PHSDIR = 0;
EPwm2Params.TBPHS = 0;
EPwm2Params.TBCTL.JSFCLKDIV = 0;
EPwm2Params.TBCTL.CLKDIV = 0;

/**< Setup Counter_Compare (CC) Submodule **/
EPwm2Params.CMPCTL.LOADAMODE = 0;
EPwm2Params.CMPCTL.LOADBMODE = 0;
EPwm2Params.CMPA = PWM_MAX;
EPwm2Params.CMPB = 32000;

/**< Setup Action_Qualifier (AQ) Submodule **/
EPwm2Params.AQCTLA = 36;
EPwm2Params.AQCTLB = 264;
EPwm2Params.AQCSFRC_CSFA = 0;
EPwm2Params.AQCSFRC_CSFB = 0;
EPwm2Params.AQCSFRC_RLDCSF = 0;

/**< Setup Dead-Band Generator (DB) Submodule **/
EPwm2Params.DBCTL.OUTMODE = 0;
EPwm2Params.DBCTL.INMODE = 0;
EPwm2Params.DBPOLSEL = 0;
EPwm2Params.DBRED = 0.0;
EPwm2Params.DBFED = 0.0;

/**< Setup Event_Trigger (ET) Submodule **/
EPwm2Params.ETSEL.SOCASEL = 1;
EPwm2Params.ETPS.SOCAPRD = 1;
EPwm2Params.ETSEL.SOCBSEL = 1;
EPwm2Params.ETPS.SOCBPRD = 1;
EPwm2Params.ETSEL.INTEN = 0;
EPwm2Params.ETSEL.INTSEL = 1;
EPwm2Params.ETPS.INTPRD = 1;

/**< Setup PWM-Chopper (PC) Submodule **/
EPwm2Params.PCCTL.CHPEN = 0;
EPwm2Params.PCCTL.CHPFREQ = 0;
EPwm2Params.PCCTL.CHPFWD = 0;
EPwm2Params.PCCTL.CHPREV = 0;

/**< Setup Trip-Zone (TZ) Submodule **/
EPwm2Params.TZSEL = 0;
EPwm2Params.TZCTL.TZA = 1;
EPwm2Params.TZCTL.TZB = 1;
EPwm2Params.TZEINT.OST = 0;
EPwm2Params.TZEINT.CBC = 0;

/**< Initial ePWM3 **/
    config_ePWMRegs(&EPwm2Regs, &EPwm2Params);
} */
Start for S-Function (c280xpwmm): '<S6>/sPWM2' */

**** Initialize ePWM1 modules ****/
{
  EPWM1Params EPwm1Params;

  /--- Setup Time-Base (TB) Submodule ---/
  EPwm1Params.TBPRD = PWM_MAX;
  EPwm1Params.TBCTL.CTRMODE = 0;
  EPwm1Params.TBCTL.SYNOSEL = 3;
  EPwm1Params.TBCTL.PHSEN = 0;
  EPwm1Params.TBCTL.PHSDIR = 0;
  EPwm1Params.TBPHS = 0;
  EPwm1Params.TBCTL.RSPCLKDIV = 0;
  EPwm1Params.TBCTL.CLKDIV = 0;

  /--- Setup Counter_Compare (CC) Submodule ---/
  EPwm1Params.CMPCTL.LOADAMODE = 0;
  EPwm1Params.CMPCTL.LOADBMODE = 0;
  EPwm1Params.CMPA = PWM_MAX;
  EPwm1Params.CMPB = 32000;

  /--- Setup Action_Qualifier (AQ) Submodule ---/
  EPwm1Params.AQCTRLA = 36;
  EPwm1Params.AQCTRLB = 264;
  EPwm1Params.AQCSFRACSF = 0;
  EPwm1Params.AQCSFRACSF = 0;

  /--- Setup Dead_Band Generator (DB) Submodule ---/
  EPwm1Params.DBCTRL.OUTMODE = 0;
  EPwm1Params.DBCTRL.INMODE = 0;
  EPwm1Params.DBCTRL.POLSEL = 0;
  EPwm1Params.DBCTRL.RLDCCSF = 0;

  /--- Setup Event_Trigger (ET) Submodule ---/
  EPwm1Params.ETSEL.SOCASEL = 1;
  EPwm1Params.ETSEL.SOCASEL = 0;
  EPwm1Params.ETSEL.SOCBSEL = 1;
  EPwm1Params.ETSEL.SOCBSEL = 0;
  EPwm1Params.ETSEL.INTEN = 0;
  EPwm1Params.ETSEL.INTSEL = 1;

  /--- Setup PWM-Chopper (PC) Submodule ---/
  EPwm1Params.PCCTRL.CHPPEN = 0;
  EPwm1Params.PCCTRL.CHPFREQ = 0;
  EPwm1Params.PCCTRL.GSHTWTH = 0;
  EPwm1Params.PCCTRL.GSPDUTY = 0;

  /--- Setup Trip-Zone (TZ) Submodule ---/
  EPwm1Params.TZSEL = 0;
  EPwm1Params.TZCTRL.TZC = 1;
  EPwm1Params.TZCTRL.TZB = 1;
  EPwm1Params.TZCTRL.TZEINT_OST = 0;
  EPwm1Params.TZCTRL.TZEINT_CBC = 0;

  /--- Initial ePWM ---/
  config_ePWMRegs(&EPwm1Regs, &EPwm1Params);
}

void EPWMset(float chan1, float chan2, float chan3){

}
```c
if (chan1 > 1)
  chan1 = 1;

if (chan1 < 0)
  chan1 = 0;

if (chan2 > 1)
  chan2 = 1;

if (chan2 < 0)
  chan2 = 0;

if (chan3 > 1)
  chan3 = 1;

if (chan3 < 0)
  chan3 = 0;

chan1 = chan1 * PWMMAX;
chan2 = chan2 * PWMMAX;
chan3 = chan3 * PWMMAX;

//--- Update CMPA value for ePWM3 ---/
{   EPwm3Regs.CMPA_half.CMPA = (Uint16)(chan3);
}

//--- Update CMPB value for ePWM3 ---/
{   EPwm3Regs.CMPB = (Uint16)(chan3);
}

/* S-Function (c280xpwm): '<S6>/ePWM1' */
//--- Update CMPA value for ePWM2 ---/
{   EPwm2Regs.CMPA_half.CMPA = (Uint16)(chan2);
}

//--- Update CMPB value for ePWM2 ---/
{   EPwm2Regs.CMPB = (Uint16)(chan2);
}

/* S-Function (c280xpwm): '<S6>/ePWM2' */
//--- Update CMPA value for ePWM1 ---/
{   EPwm1Regs.CMPA_half.CMPA = (Uint16)(chan1);
}

//--- Update CMPB value for ePWM1 ---/
{   EPwm1Regs.CMPB = (Uint16)(chan1);
}
```

C.8 LCD.h

/*
 * LCD.h - Base class that provides hardware interface to LCD
 * Copyright (c) 2011 Kieran T Levin. All right reserved.
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 */
/ * based on arduino hd44780 library */
ifndef LCDLIBS_H
#define LCDLIBS_H
#include "DSP28x_Project.h" // Device Headerfile
#include "switcherlibs.h"

// commands
#define LCD_CLEARDISPLAY 0x01
#define LCD_RETURNHOME 0x02
#define LCD_ENTRYMODESET 0x04
#define LCD_DISPLAYCONTROL 0x08
#define LCD_CURSORSHIFT 0x10
#define LCD_FUNCTIONSET 0x20
#define LCD_SETCGRAMADDR 0x40
#define LCD_SETDDRAMADDR 0x80

// flags for display entry mode
#define LCD_ENTRYRIGHT 0x00
#define LCD_ENTRYLEFT 0x02
#define LCD_ENTRYSHIFTINCREMENT 0x01
#define LCD_ENTRYSHIFTDECREMENT 0x00

// flags for display on/off control
#define LCD_DISPLAYON 0x04
#define LCD_DISPLAYOFF 0x00
#define LCD_CURSORON 0x02
#define LCD_CURSOROFF 0x00
#define LCD_BLINKON 0x01
#define LCD_BLINKOFF 0x00

// flags for display/cursor shift
#define LCD_DISPLAYMOVE 0x08
#define LCD_MOVELEFT 0x00
#define LCD_MOVERIGHT 0x04

// flags for function set
#define LCD_8BITMODE 0x10
#define LCD_4BITMODE 0x00
#define LCD_2LINE 0x08
#define LCD_1LINE 0x00
#define LCD_5x10DOTS 0x04
#define LCD_5x8DOTS 0x00

void LCDcommand(unsigned char value);
void LCDwrite(unsigned char value);
void LCDwrite8bits(unsigned char value);
void LCDpulseEnable(void);
void LCDsend(unsigned char value, unsigned char mode);
void LCDinit(unsigned char cols, unsigned char lines);
void LCDclear();
void LCDhome();
void LCDnoDisplay();
void LCDdisplay();
void LCDPrint(char *mline1, char *mline2);
void LCDupdate();
#endif // end of definition
C.9 LCD.c

/*
 * LCD.c - Base class that provides hardware interface to LCD
 * Copyright (c) 2011 Kieran T Levin. All right reserved.
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
 *
 */

#include "LCD.h"

unsigned char _LCDdisplaycontrol;
unsigned char _LCDdisplayfunction;
unsigned char _LCDdisplaymode;

void LCDcommand(unsigned char value) {
    LCDsend(value, 0);
}

void LCDwrite(unsigned char value) {
    LCDsend(value, 1);
}

int LCDposition;
char line1[20];
char line2[20];

void LCDPrint(char *mline1, char *mline2){
    memcpy(line1, mline1, sizeof(line1));
    memcpy(line2, mline2, sizeof(line2));
}

void LCDupdate(){
    if (LCDposition < 20){
        LCDwrite(line1[LCDposition]);
    }
    else if (LCDposition == 20){
        LCDcommand(LCDSETDDRAMADDR | 0x40);
    }
    else if (LCDposition < 41){
        LCDwrite(line2[LCDposition - 21]);
    }
    else if (LCDposition == 41){
        LCDcommand(LCDSETDDRAMADDR | 0x00);
        LCDposition = -1;
    }
    ++LCDposition;
}

/* void LCDsetCursor(uint8_t col, uint8_t row) */
/* { */
/*    int row_offsets[] = { 0x00, 0x40, 0x14, 0x54 ]; */
/*    if ( row > _numlines ) { */
/*        row = _numlines-1; // we count rows starting w/0 */
/*    } */
/*    LCDcommand(LCDSETDDRAMADDR | (col + row_offsets[row])); */
/* } */

void LCDwrite8bits(unsigned char value) {
    value = value & 0x00FF;
}

61
value = ((value \times 0x80200802ULL) \& 0x0884422110ULL) \times 0x0101010101ULL >> 32;
GpioDataRegs.GPBCLEAR.all = 0xFF000000;  // clear the top pins
GpioDataRegs.GPBSET.all = 0xFF000000 & ((unsigned long)value << 24);
LCDpulseEnable();
}

void LCDpulseEnable(void) {
    // GpioDataRegs.GPBCLEAR.bit.GPIO53 = 1;
    //DELAY_US(1);
    GpioDataRegs.GPBSET.bit.GPIO53 = 1;  // digitalWrite(_enable.pin, HIGH);
    //DELAY_US(1);  // enable pulse must be >450ns
    GpioDataRegs.GPBCLEAR.bit.GPIO53 = 1;  // digitalWrite(_enable.pin, LOW);
    //DELAY_US(100);  // commands need > 37us to settle
}

void LCDsend(unsigned char value, unsigned char mode) {
    GpioDataRegs.GPBDAT.bit.GPIO55 = mode;
    // digitalWrite(_rs.pin, mode);
    // if there is a RW pin indicated, set it low to Write
    //digitalWrite(_rw.pin, LOW);
    GpioDataRegs.GPBCLEAR.bit.GPIO54 = 1;
    LCDwrite8bits(value);
}

void LCDinit(unsigned char cols, unsigned char lines) {
    _LCDdisplayfunction = LCD_2LINE | LCD_4BITMODE | LCD_5x8DOTS;
    // SEE PAGE 45/46 FOR INITIALIZATION SPECIFICATION!
    // according to datasheet, we need at least 40ms after power rises above 2.7V
    // before sending commands. Arduino can turn on way before 4.5V so we'll wait 50
    DELAY_US(50000);
    // Now we pull both RS and R/W low to begin commands
    //digitalWrite(_rs.pin, LOW);
    GpioDataRegs.GPBCLEAR.bit.GPIO55 = 1;
    //digitalWrite(_enable.pin, LOW);
    GpioDataRegs.GPBCLEAR.bit.GPIO53 = 1;
    // digitalWrite(_rw.pin, LOW);
    GpioDataRegs.GPBCLEAR.bit.GPIO54 = 1;
    DELAY_US(150);
    // this is according to the hitachi HD44780 datasheet
    // page 45 figure 23
    // Send function set command sequence
    LCDcommand(LCD_FUNCTIONSET | _LCDdisplayfunction);
    DELAY_US(4500);  // wait more than 4.1ms
    // second try
    LCDcommand(LCD_FUNCTIONSET | _LCDdisplayfunction);
    DELAY_US(150);
    // third go
    LCDcommand(LCD_FUNCTIONSET | _LCDdisplayfunction);
    DELAY_US(150);
    // finally, set # lines, font size, etc.
    LCDcommand(LCD_FUNCTIONSET | _LCDdisplayfunction);
    DELAY_US(150);
    // turn the display on with no cursor or blinking default
    _LCDdisplaycontrol = LCD_DISPLAYON | LCD_CURSOROFF | LCD_BLINKOFF;
    LCDdisplay();
    DELAY_US(150);
    // clear it off
    LCDclear();
    DELAY_US(150);
    // initialize to default text direction (for romance languages)
    _LCDdisplaymode = LCD_ENTRYLEFT | LCD_ENTRYSHIFTDECREMENT;
    // set the entry mode
    LCDcommand(LCD_ENTRYMODESET | _LCDdisplaymode);
    DELAY_US(150);
}
```c
void LCDclear()
{
    LCDcommand(LCD_CLEARDISPLAY); // clear display, set cursor position to zero
    DELAY_US(2000); // this command takes a long time!
}

void LCDhome()
{
    LCDcommand(LCD_RETURNHOME); // set cursor position to zero
    DELAY_US(2000); // this command takes a long time!
}

// Turn the display on/off (quickly)
void LCDnoDisplay()
{
    LCDdisplaycontrol &= ~LCD_DISPLAYON;
    LCDcommand(LCD_DISPLAYCONTROL | LCDdisplaycontrol);
}

void LCDdisplay()
{
    LCDdisplaycontrol |= LCD_DISPLAYON;
    LCDcommand(LCD_DISPLAYCONTROL | LCDdisplaycontrol);
}

C.10 Mathlib.h

/∗
Mathlib.h – Base class that provides PID and matrix multiplication operations in floating point
Copyright (c) 2011 Kieran T Levin. All right reserved.

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software
Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
*/

#ifndef MATHLIB_H
#define MATHLIB_H

#include "DSP28x_Project.h" // Device Headerfile
#include "switcherlib.h"

#define real_T double

typedef struct PIDstruct{
    float Pgain;
    float Igain;
    float Dgain;
    double Integral;
    float Deurrent;
    float Dprev;
}PIDstruct;

typedef struct Istruct{
    float Igain;
    double Integral;
    float InputPrev;
    char saturation;
    float satHI;
}Istruct;
```
typedef struct FIRstruct {
  float prevsamples [4];
} FIRstruct;

float MathPID(PIDstruct * PID, float P, float I, float D);

float MathI(Integral * Integral, float I);

float MathSat(float input, float max, float min);

float MathFIR4(FIRstruct * FIR, float sample);

void Mmpy2x2(real T A[4], real T B[4], real T out[4]);

void Mmpy2x1(real T A[4], real T B[2], real T out[2]);

void Mmpy2xs(real T A[4], real T s, real T out[4]);

void Sq2x2(real T A[4], real T out[4]);

void Mmpydot2x2(real T A[4], real T B[4], real T out[4]);

void Mexp2x2(real T A[4], real T out[4]);

void Minv2x2(real T A[4], real T out[4]);

void Macc2x2(real T A[4], real T out[4]);

void Macc2x1(real T A[2], real T out[2]);

#define VCZONEPLUS 0x01
#define VCZONEZERO 0x02
#define VCZONEMINUS 0x04
#define ILZONEPLUS 0x10
#define ILZONEZERO 0x20
#define ILZONEMINUS 0x40

unsigned char Error2Zone(real T it, real T vc);

// end of definition

C.11 Mathlib.c

#include "Mathlib.h"
#include "math.h"

float MathPID(PIDstruct * PID, float P, float I, float D)(
  // this is a really noisy derivative so it should not really be used...
```c
float derivative;
derivative = (D - (PID->Dprev)) / SAMPLETIME;
(PID->Dprev) = D;
PID->Integral += (1 + (PID->Igain)) * SAMPLETIME;

return (float)((PID->Integral + (PID->Pgain) * P + derivative) * (PID->Dgain));
} // trapezoidal integration

float MathI(Istruct * Integral, float I)
{
    Integral->Integral += (double)(Integral->Integral * InputPrev / 2) * SAMPLETIME;
    if (Integral->saturation)
    {
        if (Integral->Integral > Integral->satHI)
            Integral->Integral = Integral->satHI;
        if (Integral->Integral < Integral->satLO)
            Integral->Integral = Integral->satLO;
    }
    Integral->InputPrev = I;
    return (float)(Integral->Integral);
}

float MathSat(float input, float max, float min)
{
    if (input > max)
        input = max;
    if (input < min)
        input = min;
    return input;
}

float MathFIR4(FIRstruct * FIR, float sample)
{
    float temp;
            FIR->prevsamples[3]) / 5;
    FIR->prevsamples[2] = FIR->prevsamples[1];
    FIR->prevsamples[1] = FIR->prevsamples[0];
    FIR->prevsamples[0] = sample;
    return temp;
}

void Mmpy2x2(real_t T A[4], real_t T B[4], real_t T out[4])
{
}

void Mmpy2x1(real_t T A[4], real_t T B[2], real_t T out[2])
{
}

void Mmpy2x1(real_t T A[4], real_t T B[2], real_t T out[2])
{
}

void Mmpy2x1(real_t T A[4], real_t T B[2], real_t T out[2])
{
}

void Sq2x2(real_t T A[4], real_t T out[4])
{
}

65
```
// dot multiply, multiply element wise two matrices together

void Mmpydot2x2(real T A[4], real T B[4], real T out[4])
{
    out[0] = A[0]*B[0];
}

#define ITER (6.00)

void Mexp2x2(real T A[4], real T out[4])
{
    real T X[4];
    real T A2[4];
    real T T1[4];
    real T cX[4];
    real T mxc = 0.50;
    real T nm;
    real T s;
    real T norm;
    real T E[4] = {1,0,0,1}; //make eye(2)
    real T D[4] = {1,0,0,1};
    int logtwo;
    int i, j, k;

    //Find the inf norm of the matrix A (max row sum)
        norm = (A[0] + A[1]);
    } else {
    }

    //find the log2 of this
    frexp(norm, &logtwo);
    logtwo += 1;
    if (logtwo < 0) {
        logtwo = 0;
        memcpy(A2, A, sizeof(A2));
    } else {
        //scale A to some exponent of log2
        for (i = 0; i < 4; i++){
            A2[i] = ldexp(A[i], -logtwo); // divide by 2^logtwo which is same as x*2^logtwo so we can use optimized instructions for this
        }
    }

    //copy A to X
    memcpy(X, A2, sizeof(X));
    mxc = 0.5;
    //do E = I + C*A
    for (i = 0; i < 4; i++){
        E[i] = E[i] + ldexp(A2[i], -1);
        D[i] = D[i] - ldexp(A2[i], -1);
    }

    for (k = 2; k <= ITER; k++){
        mxc = mxc * (ITER - k + 1)/(k * ((2 * ITER) - k + 1));
        Mmpy2x2(A2, X, A);
        memcpy(X, A, sizeof(X)); //copy results back into X
        //Do E = E + C*X
        for (i = 0; i < 4; i++){
            cX[i] = mxc * X[i]; //ldexp(X[i], -1); // multiply by 1/2
            E[i] = E[i] + cX[i];
        }
        if (k & 0x01) {
            for (i = 0; i < 4; i++){
                D[i] = D[i] - cX[i];
            }
        } else {
            for (i = 0; i < 4; i++){
                D[i] = D[i] + cX[i];
            }
        }
    }
}
//now we are done iterating, and need to do some crazy $E = D \cdot E$
memcpy(X, E, sizeof(X));  // copy results back into X

if (abs(D[1]) > abs(D[0])){
    i = 2;
    j = 1;
} else {
    i = 1;
    j = 2;
}

nm = D[j - 1] / D[i - 1];
s = D[j + 1] - nm * D[i + 1];
// dont test for bad things like zero entries here. just assume good

// manually unrolled below
// for (k = 0; k < 2; k++) {
//    E[1 + (k << 1)] = (X[(j - 1) + (k << 1)] - X[(i - 1) + (k << 1)] * nm) / s;
//    E[k << 1] = (X[(i - 1) + (k << 1)] - E[1 + (k << 1)] * D[i + 1]) / D[i - 1];
// }
E[0] = (X[(i - 1)] - E[1] * D[i + 1]) / D[i - 1];
E[1] = (X[(j - 1)] - X[(i - 1)] * nm) / s;
E[2] = (X[(i - 1) + 2] - E[3] * D[i + 1]) / D[i - 1];
E[3] = (X[(j - 1) + 2] - X[(i - 1) + 2] * nm) / s;

// undo scaling by repeated squaring
for(i = 0; i < logtwo; i++){
    Sq2x2(E, T1);
    memcpy(E, T1, sizeof(E));
}
// copy result from E to output
memcpy(out, E, sizeof(E));

void Minv2x2(real_T A[4], real_T out[4]){
    real_T temp;
}

void Macc2x2(real_T A[4], real_T out[4]){
    out[0] += A[0];
    out[1] += A[1];
}

void Macc2x1(real_T A[2], real_T out[2]){
    out[0] += A[0];
    out[1] += A[1];
}

unsigned char Error2Zone(real_T il, real_T vc){
    unsigned char temp = 0;
    if(il > NOISEI){
        temp = ILZONEPLUS;
    } else if (il > -NOISEI){
        temp = ILZONEZERO;
    } else {
        temp = ILZONEMINUS;
    }
}
if (vc > NOISEI) {
  temp |= VCZONEPLUS;
} else if (vc > -NOISEI) {
  temp |= VCZONEZERO;
} else {
  temp |= VCZONEMINUS;
}
return temp;

C.12 Obsparams.h

/*
ObsParams.h — Values used for observer parameters
Copyright (c) 2011 Kieran T Levin. All right reserved.

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
*/

/*
#define A11 (-2.0040e0)
#define A12 (-2.0040e3)
#define A21 (1.72410e3)
#define A22 (0e0)

#define B11 (2.0040e3)
#define B22 (-1.72410e3)
#endif
#endif
#define L11 (7.998e3)
#define L12 (2.5004e3)
#define L21 (1.7241e3)
#define L22 (8e3)
*/

#define SIGMA (100e0)
#define RESR (0.082e0)
#define RDSON (0.022e0)
#define RDIODE (0.1e0)
#define RINDUCTOR (0.04e0)
#define INDUCTOR (500.9e-6)
#define CAPACTOR (534e-6)
#define VDIODE (0.5137e0)

// Inverses so the compiler doesn’t do divides
#define SIGMA (100e0)
#define RESR (0.082e0)
#define RDSON (0.022e0)
#define RDIODE (0.1e0)
#define RINDUCTOR (0.04e0)
#define INDUCTOR (500.9e-6)
#define CAPACTOR (534e-6)
#define VDIODE (0.5137e0)

// Also let them see the value
#define SIGMA (100e0)
#define RESR (0.082e0)
#define RDSON (0.022e0)
#define RDIODE (0.1e0)
#define RINDUCTOR (0.04e0)
#define INDUCTOR (500.9e-6)
#define CAPACTOR (534e-6)
#define VDIODE (0.5137e0)

// div = + Cd/INDUCTOR*getchan(Vc)
C.13 Print.h

/*
Print.h – Base class that provides print() and println()
Copyright (c) 2008 David A. Mellis. All right reserved.

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1234 USA
*/

#ifndef Print_h
#define Print_h
#include "DSP28x_Project.h" // Device Headerfile
#include "switcherlibs.h"

#define uint8_t ( unsigned char )
#define DEC 10
#define HEX 16
#define OCT 8
#define BIN 2
#define BYTE 0

void printNumber(unsigned long n, unsigned char base);
void printFloat(double number, unsigned char digits);
void print(char str[]);
void printN8S(char, int);
void printN8U(unsigned char, int);
void printIntS(int, int);
void printIntU(unsigned int, int);
void printLongS(long, int);
void printLongU(unsigned long, int);
void printDouble(double, int); // second param is 2
void println(void);
#endif

C.14 Print.c

/*
Print.cpp – Base class that provides print() and println()
Copyright (c) 2008 David A. Mellis. All right reserved.

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.

69
This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Modified 23 November 2006 by David A. Mellis

#include <stdio.h>
#include <string.h>
#include <math.h>

#include "Print.h"

// Public Methods ////////////////////////////////////////////////////

/* default implementation: may be overridden */
void write(const char *str)
{
    while (*str){
        scia_xmit(*str);
        str++;
    }
}

/* default implementation: may be overridden */
void writeN(char *buffer, unsigned char size)
{
    while (size--)
    { scia_xmit(*buffer++);
    }

void print(char str[])
{
    write(str);
}

void printN8S(char c, int base)
{
    printLongS((long) c, base);
}

void printN8U(unsigned char b, int base)
{
    printIntU((unsigned long) b, base);
}

void printIntS(int n, int base)
{
    printLongS((long) n, base);
}

void printIntU(unsigned int n, int base)
{
    printIntU((unsigned long) n, base);
}

void printLongS(long n, int base)
{
    if (base == 0) {
        printLongS(n, DEC);
    } else if (base == 10) {
        if (n < 0) {
            print("-");
            n = -n;
        }
        printNumber(n, 10);
    } else {
        printNumber(n, base);
    }
void printLongU(unsigned long n, int base) {
  if (base == 0) writeN((char *)n, 4);
  else printNumber(n, base);
}

void printDouble(double n, int digits) {
  printFloat(n, digits);
}

void println(void) {
  print("\r");
  print("\n");
}

// Private Methods //////////////////////////////////////////////////////////////////////
void printNumber(unsigned long n, unsigned char base) {
  unsigned char buf[8 * sizeof(long)]; // Assumes 8-bit chars.
  unsigned long i = 0;
  if (n == 0) {
    print("0");
    return;
  }
  while (n > 0) {
    buf[i++] = n % base;
    n /= base;
  }
  for (; i > 0; i--) scia_xmit((char) (buf[i - 1] < 10 ? '0' + buf[i - 1] : 'A' + buf[i - 1] - 10));
}

void printFloat(double number, unsigned char digits) {
  unsigned char i;
  double rounding = 0.5;
  unsigned long intPart;
  double remainder;
  int toPrint;
  // Handle negative numbers
  if (number < 0.0) {
    print("- ");
    number = -number;
  }
  // Round correctly so that print(1.999, 2) prints as "2.00"
  for (i=0; i<digits; ++i)
    rounding /= 10.0;
  number += rounding;
  // Extract the integer part of the number and print it
  int_part = (unsigned long)number;
  remainder = number - (double)int_part;
  printLongU(int_part, DEC);
  // Print the decimal point, but only if there are digits beyond
  if (digits > 0)
    print(".");
}
// Extract digits from the remainder one at a time
while (digits > 0) {
    remainder *= 10.0;
    toPrint = (int) remainder;
    printIntS(toPrint, DEC);
    remainder -= toPrint;
}
}

C.15 Ring.h

/*
 * Ring.h - Base class that provides a buffered reader and writer using rings
 * Copyright (c) 2011 Kieran T Levin. All right reserved.
 *
 * This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
 *
 */
#ifndef RINGLIBS_H
#define RINGLIBS_H

#include "DSP28x_Project.h" // Device Headerfile
#include "swtchrlibs.h"

#define RINGSIZE 4

typedef struct adcringsample {
    float samples[8];
    unsigned char valid;
    float time;
    unsigned short rawtime;
} adcringsample;

typedef struct adcringdef {
    adcringsample sample[RINGSIZE];
    int head;
    int tail;
    int size;
} adcringdef;

void initRingbuf(void);

adcringsample * popRingElem(void);

void insertRingElem(adcringsample *);

int adcringavailable(void);
#endif // end of definition
C.16 Ring.c

/*
ring.c - Base class that provides a buffered reader and writer using rings
Copyright (c) 2011 Kieran T Levin. All right reserved.

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software
Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
*/
#include "Ring.h"

adcringdef adcring;

void initRingbuf(void)(
    adcring.head = 0;
    adcring.tail = 0;
    adcring.size = 0;
}

void insertRingElem(adcringsample * element){
    int i = (adcring.head + 1) % RINGSIZE;
    if( i != adcring.tail)
    { 
        memcpy(&(adcring.sample[adcring.head]), element, sizeof(adcringsample));
        adcring.head = i;
    } else {
//TODO: we have overflowed the buffer and should flag it here:
asm("NOP");
    }
}

adcringsample * popRingElem(void){
    if(adcring.head == adcring.tail)
    { 
        return (adcringsample *)0;
    }
    adcring.tail = (unsigned int)(adcring.tail + 1)% RINGSIZE;
    return &(adcring.sample[adcring.tail]);
}

int adcringavailable(void){
    return (unsigned int)(RINGSIZE + adcring.head - adcring.tail) % RINGSIZE;
}

C.17 Scilib.h

/*
scilib.h - Base class that provides hardware interface to serial port
Copyright (c) 2011 Kieran T Levin. All right reserved.

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
*/
version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

*/
#define SCILIB_H
#include "DSP28x_Project.h" // Device Headerfile

#include "Scilib.h" // Test 1,SCIA DLB,8-bit word, baud rate 0x000F, default, 1 STOP bit, no parity
#define LSPCLK 37500000
#define BAUDRATE 115200
#define BAUDREG (LSPCLK/(BAUDRATE*8) − 1)

void scia_echo_back_init(void);
void scia_fifo_init(void);
void scia_xmit(int a);
void scia_msg(const char *msg);
void scia_rx_isr(void);
unsigned char scia_has_byte(void);
unsigned char scia_get_byte(void);

C.18 Scilib.c

/*
scilib.c – Base class that provides hardware interface to serial port
Copyright (c) 2011 Kieran T Levin. All right reserved.

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

*/

#define SCILIB

#include "Scilib.h"

// Test 1,SCIA DLB, 8-bit word, baud rate 0x000F, default, 1 STOP bit, no parity
#define LSPCLK 37500000
#define BAUDRATE 115200
#define BAUDREG (LSPCLK/(BAUDRATE*8) − 1)
void scia_echo_back_init()
{
    // Note: Clocks were turned on to the SCIA peripheral
    // in the InitSysCtrl() function
    SciaRegs.SCICCR.all =0x0007; // 1 stop bit, No loopback
    SciaRegs.SCICTL1.all =0x0003; // No parity,8 char bits,
    SciaRegs.SCICTL2.all =0x0003; // asyn mode, idle−line protocol
    SciaRegs.SCICTL1.bit.TXINTENA =1;


```c
SciaRegs.SCICTL2.bit.RXINTENA = 1;

/* #if (CPU_FRQ_150MHZ)
   SciaRegs.SCIHBAUD = 0x0001; // 9600 baud @LSPCLK = 37.5MHz.
   SciaRegs.SCILBAUD = 0x00E7;
#endif
#ifdef (CPU_FRQ_100MHZ)
   SciaRegs.SCIHBAUD = 0x0001; // 9600 baud @LSPCLK = 20MHz.
   SciaRegs.SCILBAUD = 0x0044;
#endif

SciaRegs.SCIHBAUD = (BAUDREG >> 8) & 0x00FF;
SciaRegs.SCILBAUD = (BAUDREG & 0x00FF);
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.SCTXBUF = a;
}

void scia_msg(const char * msg)
{
    int i = 0;
    while (msg[i] != '\0')
    {
        scia_xmit(msg[i]);
        i++;
    }
}

// Initialize the SCI FIFO
void scia_fifo_init()
{
    SciaRegs.SCIFFTX.all = 0xE040;
    SciaRegs.SCIFFRX.all = 0x204F;
    SciaRegs.SCIFFCT.all = 0x0;
}

unsigned short rxbyte;
unsigned short receivedbyte;

// Interrupt void scia_rx_isr()
{
    rxbyte = SciaRegs.SCRXBUF.all;
    receivedbyte = 1;
    SciaRegs.SCIFFRX.bit.RXPOVRCLR = 1; // Clear Overflow flag
    SciaRegs.SCIFFRX.bit.RXFFINTCLR = 1; // Clear Interrupt flag
    PIEControlRegs.PIEACK.all |= 0x100; // Issue PIE ack
}

unsigned char scia_has_byte()
{
    if (SciaRegs.SCIFFRX.bit.RXFFST)
    {
        return 1;
    }
    return 0;
}

unsigned char scia_get_byte()
{
    // receivedbyte = 0;
    // return rxbyte;
    return SciaRegs.SCRXBUF.all;
}
```

C.19 switcherlibs.h

/*
switcherlibs.h - header file to build project
Copyright (c) 2011 Kieran T Levin. All right reserved.

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
*/

#ifndef SWITCHERLIBS_H
#define SWITCHERLIBS_H
#include "DSP28x_Project.h" // Device Header file
#include "math.h" // serial port custom library
#define FALSE ( 0 )
#define TRUE ( 1 )
#define REALTIME
#define SAMPLERATE ( 1.0E-005 * 1000000 )
#define SAMPLETIME ( 1.0E-005 )
#include "obsparams.h"
#include "Scilib.h"
#include "Adclib.h"
#include "Print.h"
#include "Adc2VA.h"
#include "Epwmlib.h"
#include "Mathlib.h"
#include "LCD.h"
#include "Ring.h"
#endif // end of DSP28x_PROJECT_H definition

76
APPENDIX D

MATLAB SOURCE CODE

D.1 serialreadtofiles.m

```matlab
%serial to Mfile recorder....
clear

if (0 == isValid(s))
    s = serial('COM4');
    set(s,'BaudRate',115200,'DataBits',8,'Parity','none');
    fopen(s);
end

% Find a GPIB object.
obj1 = instrfind('Type','gpib','BoardIndex',7,'PrimaryAddress',1,'Tag','');

% Create the GPIB object if it does not exist
% otherwise use the object that was found.
if isempty(obj1)
    obj1 = gpib('AGILENT', 7, 1);
else
    fclosr(obj1);
    obj1 = obj1(1)
end

% Connect to instrument object, obj1.
fopen(obj1);
fprintf(obj1, 'MODE:RES ');
fprintf(obj1, 'RES 10 ');

%timer function so we delay
T = timer('StartDelay', 5, 'TimerFcn',...
    @(x,y)disp('Waiting for converter results'));

%flush the serial buffers
N = s.BytesAvailable();
while (N>0)
    fread(s,N);
    N = s.BytesAvailable();
end
Ts = 1e-5;
displayframes = 2050;
x = (0:T):(displayframes)*Ts - Ts;

for resistance = [2.5,10,15,20]
    output = zeros(8,displayframes,11);
    for voltage = 1:8
        fprintf(obj1, 'RES %u',resistance);
        %set the output voltage setpoint
        fprintf(s,'%u',voltage);
        %start the experiment by sending run
        fprintf(s,'g ');
        start(t);
        wait(t);
        fprintf(s,'r ');
        start(t);
        wait(t);
        fprintf(s,'d ');
    end
end
```
tline = fgetl(s);
counter = 1;
while (1 != strncmp(tline, 'Section End', 9))
    C = textscan(tline, '
' % f32 % f32 % f32 % f32 % f32 % f32 % f32 % f32 % f32 % f32'', 'delimiter', ',');
    output(voltage, counter, :) = cell2mat(C);
counter = counter + 1;
tline = fgetl(s);
end

name = sprintf('%dAutoRunResults_data_INDUCTOR=5_Rload=%u.mat', resistance);
save(name, 'output', 'Ts', 'x');
end

% turn off the converter
fprintf(s, 's');
close(s)
delete(s)
clear s
close(obj1);

x = (0:Ts:(4096)*Ts - Ts);

% subplot(3,3,9)
% hold on
% plot(output(:,1), output(:,2), 'color', 'blue');
% xlabel('I amps');
% ylabel('Vc Volts');
% title('Difference Equations XY');

% displayframes = 1800; % drop the empty frames
% output = output(1:displayframes,:);

% load('results_data.mat');

% subplot(6,2,1)
% plot(x, output(:,5));
% title(['Voltage In. Avg ', num2str(mean(output(:,5))))];
% ylabel('V');
%
% subplot(6,2,2)
% plot(x, output(:,4));
% title(['Current In. Avg ', num2str(mean(output(:,4))))];
% ylabel('A');
%
% subplot(6,2,3)
% plot(x, output(:,3));
% title(['Inductor Current. Avg ', num2str(mean(output(:,3))))];
% ylabel('A');
%
% subplot(6,2,4)
% plot(x, output(:,2));
% title(['Voltage Out. Avg ', num2str(mean(output(:,2))))];
% ylabel('V');
%
% subplot(6,2,5)
% plot(x, output(:,1));
% title(['Current Out. Avg ', num2str(mean(output(:,1))))];
% ylabel('A');
%
% subplot(6,2,6)
% plot(x, output(:,6));
% title('Fault Trigger');
% ylabel('Tf');
%
% subplot(6,2,7)
% plot(x, output(:,7));
% title('Observer II');
% ylabel('A');

78
D.2 displaygraphs.m

```matlab
% This is the most important one......
% subplot(6,2,12)
% plot(x,(outdata(:,1)+outdata(:,2)));
% title(['Power out data. Avg ',num2str(mean((outdata(:,1)+outdata(:,2))))]);
% ylabel('W');
% plot(x,(output(:,2)-output(:,8)));

Ts = 1e-5;
x = (0:Ts:(displayframes)*Ts-Ts);
for channel = 1:11
    subplot(4,3,channel)
    plot(x,output(:,channel));
    meanout = mean(output(:,channel));
    stddevout = std(output(:,channel));
    axis([0 time (meanout-2*stddevout) (meanout+2*stddevout)]);
    xlabel('t');
    switch channel
    case 1 %error IL
        title(['IL Error. Avg ',num2str(mean(output(:,channel))))]);
        ylabel('I');
    case 2 %error VC
        title(['VC Error. Avg ',num2str(mean(output(:,channel))))]);
    end
end
```

79
ylabel('V');
case 3 %configuration (if switch 1 or 2 is on)
title(['System Configuration. Avg ',num2str(mean(output(run,:,channel)))]);
ylabel('\sigma');
case 4 %Inductor
title(['I. Inductor. Avg ',num2str(mean(output(run,:,channel)))]);
ylabel('I');
case 5 % Capacitor
title(['V Capacitor. Avg ',num2str(mean(output(run,:,channel)))]);
ylabel('V');
case 6 % observer
title(['Observer 1. Avg ',num2str(mean(output(run,:,channel)))]);
ylabel('V');
case 7 % Observer Vc
title(['Observer Vc. Avg ',num2str(mean(output(run,:,channel)))]);
ylabel('V');
case 8 %Vinput
title(['Voltage In. Avg ',num2str(mean(output(run,:,channel)))]);
ylabel('V');
case 9 %output
title(['Voltage Out. Avg ',num2str(mean(output(run,:,channel)))]);
ylabel('V');
case 10 %output
title(['Current Out. Avg ',num2str(mean(output(run,:,channel)))]);
ylabel('I');
case 11 %deltatime
title(['DeltaTime. Avg ',num2str(mean(output(run,:,channel)))]);
ylabel('sec');
end

D.3 generateErrors.m

clear;

%load('results_data_Nominal_Rload4.mat');
%load('AutoRunResults_data_Nominal_Rload=2.mat');

for error = [1 2 3 4 5 10 15] %for capacitor
    for error = [1 2 3 4 5 10] %for inductor
        load(['AutoRunResults_data_RESR=',num2str(error), '_Rload=',num2str(rload),'.mat']);
        load(['AutoRunResults_data_Nominal_Rload=',num2str(rload),'.mat']);
        load(['AutoRunResults_data_R1INDUCTOR=',num2str(error), '_Rload=',num2str(rload),'.mat']);
        run = 5;
        startframe = 100; %if we want to adjust start index
        displayframes = 1000; %drop the empty frames
        output = output(:,startframe:displayframes+startframe-1,:);
        Ts = 1e-5;
        x = (0:Ts:(displayframes)*Ts - Ts);
        %deltatime is in channel 11....
        time = 0;
        for i = 1:displayframes
            for j = 1:11
                if (output(run,i,j)) > 6 | (output(run,i,j)) < -6
                    output(run,i,j) = median(output(run,:,j));
                end
            end
            time = time + output(run,i,11);
            x(i) = time;
        end
        %generate state values
        %outputstate1 = zeros(250,11);
        %outputstate2 = zeros(250,11);
        for j = 1:8
            outputstatelogical = (output(run,:,3) < -1);
            outputstatelogical = outputstatelogical(1:250);
            for i = 1:11
                outputstatelogical = (output(run,:,3) > 6);
                outputstatelogical = outputstatelogical(1:250);
                for i = 1:11
                    outputstatelogical = (output(run,:,3) < -6);
                    outputstatelogical = outputstatelogical(1:250);
                    for i = 1:11
                        outputstatelogical = (output(run,:,3) > 6);
                        outputstatelogical = outputstatelogical(1:250);
                        for i = 1:11
                            outputstatelogical = (output(run,:,3) < -6);
                            outputstatelogical = outputstatelogical(1:250);
                            for i = 1:11
                                outputstatelogical = (output(run,:,3) > 6);
                                outputstatelogical = outputstatelogical(1:250);
                                for i = 1:11
                                    outputstatelogical = (output(run,:,3) < -6);
                                    outputstatelogical = outputstatelogical(1:250);
                                    for i = 1:11
                                        outputstatelogical = (output(run,:,3) > 6);
                                        outputstatelogical = outputstatelogical(1:250);
                                        for i = 1:11
                                            outputstatelogical = (output(run,:,3) < -6);
                                            outputstatelogical = outputstatelogical(1:250);
                                            for i = 1:11
                                                outputstatelogical = (output(run,:,3) > 6);
                                                outputstatelogical = outputstatelogical(1:250);
                                                for i = 1:11
                                                    outputstatelogical = (output(run,:,3) < -6);
                                                    outputstatelogical = outputstatelogical(1:250);
                                                    for i = 1:11
                                                        outputstatelogical = (output(run,:,3) > 6);
                                                        outputstatelogical = outputstatelogical(1:250);
                                                        for i = 1:11
                                                            outputstatelogical = (output(run,:,3) < -6);
                                                            outputstatelogical = outputstatelogical(1:250);
                                                            for i = 1:11
                                                                outputstatelogical = (output(run,:,3) > 6);
                                                                outputstatelogical = outputstatelogical(1:250);
                                                                for i = 1:11
                                                                    outputstatelogical = (output(run,:,3) < -6);
                                                                    outputstatelogical = outputstatelogical(1:250);
                                                                    for i = 1:11
                                                                        outputstatelogical = (output(run,:,3) > 6);
                                                                        outputstatelogical = outputstatelogical(1:250);
                                                                        for i = 1:11
                                                                            outputstatelogical = (output(run,:,3) < -6);
                                                                            outputstatelogical = outputstatelogical(1:250);
                                                                            for i = 1:11
                                                                                outputstatelogical = (output(run,:,3) > 6);
                                                                                outputstatelogical = outputstatelogical(1:250);
                                                                                for i = 1:11
                                                                                    outputstatelogical = (output(run,:,3) < -6);
                                                                                    outputstatelogical = outputstatelogical(1:250);
                                                                                    for i = 1:11
                                                                                        outputstatelogical = (output(run,:,3) > 6);
                                                                                        outputstatelogical = outputstatelogical(1:250);
                                                                                        for i = 1:11
                                                                                            outputstatelogical = (output(run,:,3) < -6);
                                                                                            outputstatelogical = outputstatelogical(1:250);
                                                                                            for i = 1:11
                                                                                                outputstatelogical = (output(run,:,3) > 6);
                                                                                                outputstatelogical = outputstatelogical(1:250);
                                                                                                for i = 1:11
                                                                                                    outputstatelogical = (output(run,:,3) < -6);
                                                                                                    outputstatelogical = outputstatelogical(1:250);
                                                                                                    for i = 1:11
                                                                                                        outputstatelogical = (output(run,:,3) > 6);
                                                                                                        outputstatelogical = outputstatelogical(1:250);
                                                                                                        for i = 1:11
                                                                                                            outputstatelogical = (output(run,:,3) < -6);
                                                                                                            outputstatelogical = outputstatelogical(1:250);
                                                                                                            for i = 1:11
                                                                                                                outputstatelogical = (output(run,:,3) > 6);
                                                                                                                outputstatelogical = outputstatelogical(1:250);
                                                                                                                for i = 1:11
                                                                                                                    outputstatelogical = (output(run,:,3) < -6);
                                                                                                                    outputstatelogical = outputstatelogical(1:250);
                                                                                                                    for i = 1:11
                                                                                                                        outputstatelogical = (output(run,:,3) > 6);
                                                                                                                        outputstatelogical = outputstatelogical(1:250);
                                                                                                                        for i = 1:11
                                                                                                                            outputstatelogical = (output(run,:,3) < -6);
                                                                                                                            outputstatelogical = outputstatelogical(1:250);
                                                                                                                            for i = 1:11
                                                                                                                                outputstatelogical = (output(run,:,3) > 6);
                                                                                                                                outputstatelogical = outputstatelogical(1:250);
                                                                                                                                for i = 1:11
                                                                                                                                    outputstatelogical = (output(run,:,3) < -6);
                                                                                                                                    outputstatelogical = outputstatelogical(1:250);
                                                                                                                                    for i = 1:11
                                                                                                                                        outputstatelogical = (output(run,:,3) > 6);
                                                                                                                                        outputstatelogical = outputstatelogical(1:250);
                                                                                                                                        for i = 1:11
                                                                                                                                            outputstatelogical = (output(run,:,3) < -6);
                                                                                                                                            outputstatelogical = outputstatelogical(1:250);
                                                                                                                                            for i = 1:11
                                                                                                                                                outputstatelogical = (output(run,:,3) > 6);
                                                                                                                                                outputstatelogical = outputstatelogical(1:250);
                                                                                                                                                for i = 1:11
                                                                                                                                                    outputstatelogical = (output(run,:,3) < -6);
                                                                                                                                                    outputstatelogical = outputstatelogical(1:250);
                                                                                                                                                    for i = 1:11
                                                                                                                                                        outputstatelogical = (output(run,:,3) > 6);
                                                                                                                                                        outputstatelogical = outputstatelogical(1:250);
                                                                                                                                                        for i = 1:11
                                                                                                                                                            outputstatelogical = (output(run,:,3) < -6);
                                                                                                                                                            outputstatelogical = outputstatelogical(1:250);
                                                                                                                                                            for i = 1:11
                                                                                                                                                                outputstatelogical = (output(run,:,3) > 6);
                                                                                                                                                                outputstatelogical = outputstatelogical(1:250);
                                                                                                                                                                for i = 1:11
                                                                                                                                                                    outputstatelogical = (output(run,:,3) < -6);
                                                                                                                                                                    outputstatelogical = outputstatelogical(1:250);
                                                                                                                                                                    for i = 1:11
                                                                                                                                                                        outputstatelogical = (output(run,:,3) > 6);
                                                                                                                                                                        outputstatelogical = outputstatelogical(1:250);
                                                                                                                                                                        for i = 1:11
                                                                                                                                                                            outputstatelogical = (output(run,:,3) < -6);
                                                                                                                                                                            outputstatelogical = outputstatelogical(1:250);
                                                                                                            for i = 1:11
                                                for i = 1:11
                            for i = 1:11
                        for i = 1:11
                    for i = 1:11
                for i = 1:11
            for i = 1:11
        for i = 1:11
    for i = 1:11

80
outputstate1(run,:,i) = −output(run,logical(outputstate1logical),i);  
outputstate2(run,:,i) = −output(run,logical(1−outputstate1logical),i);  
end

%end  
yoffset = 0.04;  
xoffset = 0.0;  
%arrow = '\leftarrow';  
arrow = '>';  
%plot the Il(x) vs Vc(y) error  
subplot(2,1,1);  
plot(outputstate1(run,:,1),outputstate1(run,:,2));  
plot(median(outputstate1(run,:,1)),median(outputstate1(run,:,2)),'o','MarkerSize'  
     ,12,'MarkerFaceColor' ,'.49 1 .63');  
hold on;  
% Transform from data space to figure space  
%axis([min(-1,min(-0.2+outputstate1(run,:,1))) max(1,0.2+max(outputstate1(run,:,1))))  
min(-1,-0.2+min(outputstate1(run,:,2))) max(1,0.2+max(outputstate1(run,:,2))))])  
;  
content = sprintf('R_{%d} = %d ', error);  
%content = ' ';  
% Plot anno text centered at the tail of the arrow  
text(xoffset+mean(outputstate1(run,:,1)) , yoffset+mean(outputstate1(run,:,2)) , [arrow ,  
content])  
title('Error dynamics in state 1');  
ylabel('Error V_{c} (A)');  
xlabel('Error I_{L} (A)');  
 subplot(2,1,2);  
plot(outputstate2(run,:,1),outputstate2(run,:,2) , 'red');  
plot(median(outputstate2(run,:,1)),median(outputstate2(run,:,2)) , '-mo' , 'MarkerSize'  
     ,12,'MarkerFaceColor' ,'.49 1 .63');  
hold on;  
%axis([min(-1,-0.2+min(outputstate2(run,:,1))) max(1,0.2+max(outputstate2(run,:,1))))  
min(-1,-0.2+min(outputstate2(run,:,2))) max(1,0.2+max(outputstate2(run,:,2))))])  
;  
text(xoffset+mean(outputstate2(run,:,1)) , yoffset+mean(outputstate2(run,:,2)) , [arrow ,  
content])  
title('Error dynamics in state 2');  
ylabel('Error V_{c} (V)');  
xlabel('Error I_{L} (A)');  
end;

D.4 Buckconverter.m

function [IlOut VcOut Vout] = Buckconverter(Il , Vc , Iload , Iin , Vin , Time , DutyState ,  
iteration , f , failure)

%Converter Parameters  
RESR = (0.082 e0);  
RDSON = (0.022 e0);  
RDIODE = (0.1 e0);  
RINDUCTOR = (0.04 e0);  
INDUCTOR = (500.9 e−6);  
CAPACITOR = (534 e−6);  
VDIODE = (0.5137 e0);  
F_NONE = 0;  
F_CAPACITOR = 1;  
F_INDUCTOR = 2;  
F_RESR = 3;  
F_RINDUCTOR = 4;  
F_DSON = 5;  
F_RDIODE = 6;

switch failure  
case F_CAPACITOR  
    CAPACITOR = (534e−6) − (534e−6)*f;  
case F_INDUCTOR  
    INDUCTOR = (500.9e−6) − (500.9e−6)*f;  
case F_RESR  
    RESR = (0.082e0) + (10)*f;  
case F_RINDUCTOR

81
RINDUCTOR = (0.04e0) + (1)*f;
case F_DSON
RDSON = (0.022e0) + (1)*f;
case F_RDIODE
RDIODE = (0.1e0) + (1)*f;
otherwise
end
if iteration > 500
% RINDUCTOR = 2*RINDUCTOR;
% CAPACITOR = 0.6*CAPACITOR;
% INDUCTOR = 0.6*INDUCTOR;
% RESR = 5*RESR;
end
Rload = Vc/Iload;
cd = Rload/(Rload + RESR);
if DutyState
A = [((-ORDSON - RINDUCTOR - cd*RESR)/INDUCTOR) (-cd/INDUCTOR) ; (cd/CAPACITOR) (-cd/Rload/CAPACITOR)];
B = [(Vin/INDUCTOR) ; (0)];
else
A = [((-ORDIODE - RINDUCTOR - cd*RESR)/INDUCTOR) (-cd/INDUCTOR) ; (cd/CAPACITOR) (-cd/Rload/CAPACITOR)];
B = [(-VDIODE/INDUCTOR) ; (0)];
end
X = expm(A*Time)*[Iload;Vc] + A'-1*(expm(A*Time) - eye(2))*B;
Vout = X(2) + RESR*(X(1) - Iload);
IlOut = X(1);
VcOut = X(2);
end

D.5 BuckObserver.m

function [IlOutObs VcOutObs VoutObs] = BuckObserver(Ii, Vc, Ilobs, Vcobs, Iload, Iin,
Vin, Time, DutyState)
% Converter Parameters
ORESR = (0.082e0);
ORDSON = (0.022e0);
ORDIODE = (0.1e0);
ORINDUCTOR = (0.04e0);
OINDUCTOR = (500.9e-6);
OCAPACITOR = (534e-6);
OVDIODE = (0.5137e0);
Rload = Vcobs/Iload;
cd = Rload/(Rload + ORESR);
if DutyState
A = [((-ORDSON - ORINDUCTOR - cd*ORESR)/OINDUCTOR) (-cd/OINDUCTOR) ; (cd/OCAPACITOR) (-cd/Oload/OCAPACITOR)];
B = [(Vin/OINDUCTOR) ; (0)];
else
A = [((-ORDIODE - ORINDUCTOR - cd*ORESR)/OINDUCTOR) (-cd/OINDUCTOR) ; (cd/OCAPACITOR) (-cd/Oload/OCAPACITOR)];
B = [(-VDIODE/OINDUCTOR) ; (0)];
end
% Debugging stuff
X0 = expm(A*Time)*[Ilobs;Vcobs];
X1 = expm(20*eye(2)*Time)*[Ii;Vc] - [Ilobs;Vcobs];
X2 = A' - 1*expm(A*Time) - eye(2)*B;
X = X0 + X1 + X2;
% X = [expm(A*Time)*[Ilobs;Vcobs] + A' - 1*(expm(A*Time) - eye(2))*B + expm(20*eye(2)*Time)* X0 + X1 + X2]*[Ii;Vc] - [Ilobs;Vcobs];
VoutObs = X(2) + ORESR*(X(1) - Iload);
IlOutObs = X(1);
VcOutObs = X(2);
D.6 Bucksimulator.m

```matlab
%let s imulate a buck converter now
RESR = (0.082e0);
RDSON = (0.022e0);
RDIODE = (0.1e0);
RINDUCTOR = (500.9e-6);
INDUCTOR = (534e-6);
CAPACITOR = (534e-6);
VDIODE = (0.5137e0);

%[IOut VcOut] = Buckconverter(Il, Vc, Iload, Iin, Vin, Time, DutyState)
%clear;
%clc;
%setup plot data
LOOPSIZE = 6000;
f = 0; %level of failure 0 no fail to 1 component fully failed...
output = zeros (LOOPSIZE, 8);
Vset = 5;
Vc = 5;
Vout = 5.0;
Il = 2;
Vcobs = 5;
Ilobs = 2;
Rload = 2.5;
Iload = Vc/Rload;
Vin = 10;
DutyState = 1;
Time = 1/10000/2;
Period = 1/10000;
lin = 0;
x = (0:Time:LOOPSIZE∗Time−Time);
CurrentTime = 0;

for k = 1:LOOPSIZE
[IlOut VcOut Vout] = Buckconverter(Il, Vc, Iload, Iin, Vin, Time, DutyState,k,f);
[IlOutObs VcOutObs VoutObs] = BuckObserver(Il, Vc, Ilobs, Vcobs, Iload, Iin, Vin,
Time, DutyState);

x(k) = CurrentTime;

%do a voltage setpoint change to see how the converter reacts
if k == 500
Vset = 7;
end

if DutyState
Time = Period − Pcontrol∗Period;
DutyState = 0;
else
Pcontrol = Pcontrol + 30∗((Vset − VcOut))∗Period;
if Pcontrol >= 0.97
Pcontrol = 0.97;
else if Pcontrol < 0.00
Pcontrol = 0.00;
end
end
Time = Pcontrol∗Period;
DutyState = 1;
end

CurrentTime = CurrentTime + Time;
Il = IlOut;
Vc = VcOut;
Ilobs = IlOutObs;
Vcobs = VcOutObs;
Iload = Vc/Rload;
output(k,1) = Il;
output(k,2) = Vc;
output(k,3) = Vout;
output(k,4) = Vcobs;
output(k,5) = Ilobs;
end

83
D.7 errordynamics.m

%syms 'c' 'lambdac' 'lambdaesr' 'l' 'rdson' 'rl' 'rear' 'lambdaL' 'real';
%syms 'rd' 'lambdarad' 'lambdaradson' 'real';
RESR = (0.082e0);
RDSON = (0.022e0);
RDIODE = (0.1e0);
RINDUCTOR = (0.04e0);
INDUCTOR = (500.9e-6);
CAPACITOR = (534e-6);
VDIODE = (0.5137e0);
x = [1; 5];
u = [10; 2];
lambdac = CAPACITOR*0;
lambdal = INDUCTOR*0;
lambdaESR = RESR*2;
lambdaRL = RINDUCTOR*0;
lambdaRDSON = RDSON*0;

%output cap fault
Ea1c = [0 0; (1/(CAPACITOR + lambdac) - 1/CAPACITOR) (-1/(CAPACITOR+lambdac)*0)];
Ea2c = Ea1;
\[ Eb_{1c} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \frac{1}{(\text{CAPACITOR} + \lambda_C)} & 0 \end{bmatrix} \];
\[ Eb_{2c} = Eb_{1c} \];

% A matrix error at time t
\[ t = \frac{1}{10000} \times 0.5; \]
\[ \expm(E\_{a1c} \times t) \times x; \]
\[ \expm(E\_{b1c} \times t) \times u; \]

% ESR Fault
\[ E_{a1esr} = \begin{bmatrix} -\lambda_{ESR}/\text{INDUCTOR} & 0 & 0 \end{bmatrix}; \]
\[ E_{a2esr} = E_{a1esr}; \]
\[ E_{b1esr} = \begin{bmatrix} 0 & -\lambda_{ESR}/\text{INDUCTOR} & 0 \end{bmatrix}; \]
\[ E_{b2esr} = E_{b1esr}; \]
\[ \expm(E_{a1esr} \times t) \times x; \]
\[ \expm(E_{b1esr} \times t) \times u; \]

D.8 parametervariations.m

% lets simulate a buck converter now
% capacitor noise profile = 0.0076;
% inductor noise profile = 0.0323;

F\_NONE = 0;
F\_CAPACITOR = 1;
F\_INDUCTOR = 2;
F\_RESR = 3;
F\_RINDUCTOR = 4;
F\_DSON = 5;
F\_RDIODE = 6;

% \[ [I_{Out}, V_{cOut}] = \text{Buckconverter}(I_l, V_c, I_{load}, I_{in}, Vin, Time, DutyState) \]
% clear;
% setup plot data
LOOPSIZE = 3000;
failure = 1; % set this to the type of component failure that will happen
Time = 1/10000/2;
Period = 1/10000;
output = zeros(LOOPSIZE, 8);
x = (0 : Time : LOOPSIZE \times Time - Time);
for failure = 1 : 1 : 6
    for f = 0 : 0.1 : 0.9
        Vset = 5;
        Vc = 5;
        Vout = 5.0;
        I_l = 2;
        Vcobs = 5;
        Ilobs = 2;
        Rload = 2.5;
        Vin = 10;
        DutyState = 1;
        Rload = 2;
        I_{in} = 0;
        CurrentTime = 0;
        Pcontrol = 0.5429;
        for Vset = 2 : 1 : 8
            for Rload = 1 : 1 : 10
                I_{load} = Vc/Rload;
                for k = 1 : LOOPSIZE
                    ...
x(k) = CurrentTime;

if DutyState
  Time = Period + PIcontrol*Period;
  DutyState = 0;
else
  PIcontrol = PIcontrol + 30*((Vset - VcOut))*Period;
  if PIcontrol >= 0.97
    PIcontrol = 0.97;
  elseif PIcontrol < 0.00
    PIcontrol = 0.00;
  end
end

Time = PIcontrol*Period;
DutyState = 1;

CurrentTime = CurrentTime + Time;
Il = IlOut;
Vc = VcOut;
Ilobs = IlOutObs;
Vcobs = VcOutObs;
Iload = Vc/Rload;
output(k,1) = Il;
output(k,2) = Vc;
output(k,3) = Vout;
output(k,4) = Vcobs;
output(k,5) = Ilobs;
output(k,6) = VoutObs;
end

subplot(1,2,1)
hold on;
%plot(output(LOOPSIZE-1,1) - output(LOOPSIZE-1,5),output(LOOPSIZE-1,3) - output(LOOPSIZE-1,6),'*','Color',[1-1 f 0],'MarkerSize',11-Rload);
%plot(output(LOOPSIZE-2,1) - output(LOOPSIZE-2,5),output(LOOPSIZE-2,3) - output(LOOPSIZE-2,6),'+','Color',[1-1 f 0],'MarkerSize',11-Rload);
%plot(output(LOOPSIZE-1,1) - output(LOOPSIZE-1,5),output(LOOPSIZE-1,3) - output(LOOPSIZE-1,6),'.','Color',[1-1 f 0],'MarkerSize',Vset);
%plot(output(LOOPSIZE-2,1) - output(LOOPSIZE-2,5),output(LOOPSIZE-2,3) - output(LOOPSIZE-2,6),'.','Color',[1-1 f 0],'MarkerSize',Vset);
plot(output(LOOPSIZE-1,1) - output(LOOPSIZE-1,5),output(LOOPSIZE-1,3) - output(LOOPSIZE-1,6),'.','Color',[failure/6-failure/6 0],'MarkerSize',5);
plot(output(LOOPSIZE-2,1) - output(LOOPSIZE-2,5),output(LOOPSIZE-2,3) - output(LOOPSIZE-2,6),'.','Color',[failure/6-failure/6 0],'MarkerSize',5);
end

figure

% title('Capacitor fault Difference Equations');
% subplot(1,2,2)
% hold on;
% xxlabel('Il Amps');
% ylabel('Vc Volts');
% title('Difference Equations signs 2');
end

circle = rsmak('circle');
ellipse = fncmb(circle,[inductornoiseprofile 0.0 capacitornoiseprofile]);
fnplt(ellipse);
xlabel('Il Amps');
ylabel('Vc Volts');
switch failure
  case F_CAPACITOR
    title('Capacitor fault Difference Equations');
  case F_INDUCTOR
    title('Inductor fault Difference Equations');
  case F_RESR
    title('ESR fault Difference Equations');
  case F_RINDUCTOR
    title('R Inductor fault Difference Equations');
  case F_DSON
    title('RDSOn fault Difference Equations');
  case F_RDIODE
    title('R Diode fault Difference Equations');
  otherwise
end
REFERENCES


