Summer 2012

A digitally implemented practical photovoltaic simulator with a double current mode controller

Jie Ang Zhao

Department:

A DIGITALLY IMPLEMENTED PRACTICAL PHOTOVOLTAIC SIMULATOR WITH A DOUBLE CURRENT MODE CONTROLLER

by

JIE ANG ZHAO

A THESIS

Presented to the Faculty of the Graduate School of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

2012

Approved by

Jonathan W. Kimball, Advisor
Keith Corzine
Mehdi Ferdowsi

ABSTRACT

This thesis presents a microcontroller- and DC-DC converter-based photovoltaic (PV) simulator which emulates the output characteristic of a PV module. The current-voltage (IV) characteristic of a PV module is implemented as a look-up-table that generates an output reference current based on a measured output voltage. The control mechanism is based on a double current mode controller that consists of a predictive current mode controller and a proportional-integral (PI) controller arranged in an inner loop and outer loop configuration. The PI controller is designed based on classical phase margin and gain margin criteria to ensure system stability. The performance of the portable PV simulator prototype of 85 W is examined in terms of its steady state IV curve matching capability and the convergence time corresponding to step changes in load and insolation levels. The result shows a well behaved and responsive PV simulator that can be treated as a real PV module in most situations.

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Jonathan Kimball for his guidance and support which allow me to conduct this research. Without Dr. Kimball's vast knowledge and dedication in the electrical engineering field, this research will not come to fruition.

I would also like to acknowledge the members of my committee, Dr. Keith Corzine and Dr. Mehdi Ferdowsi, for their interest in my research topic and their instructions in electric drive machines and power electronics.

I want to express my deepest gratitude to my family for their support and understanding. Despite all the hardships and difficulties that my parents have in their lives, they are still able to stand strong and put faith in me.

I would also like to thank my friends for their helps and advises.

Last, I would like to thank the National Science Foundation and the University of Missouri research board for their financial support. This project was funded in part by NSF grant ECCS-0900940.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Page

LIST OF TABLES

Page

NOMENCLATURE

Symbol	Description
PV	Photovoltaic
MPPT	Maximum Power Point Tracking
LUT	Look-Up-Table
PWM	Pulse-Width Modulation
ADC	Analog-to-Digital Converter
PI	Proportional-Integral
DSP	Digital Signal Processor
I_D	Diode Current
V_D	Diode Voltage
I_0	Reverse Saturation Current
V_t	Thermal Voltage
A	Ideality Factor
k	Boltzmann's Constant
T	Absolute Temperature
q	Elementary Charge
I_{PVout}	Output Current of a PV Module
V_{PVout}	Output Voltage of a PV Module
R_s	Series Resistance of a PV Cell Model
R_{sh}	Shunt Resistance of a PV Cell Model
N	Number of Series Connected Cells of a PV Module
I_{ph}	Photo-generated Current
$I_{ph(max)}$	Photo-generated Current at Maximum Insolation
α	Insolation Percentage
Λ	Irradiance
MPP	Maximum Power Point
$V_{PVout}[i]$	Digital Output Voltage
V_{OC}	Open Circuit Voltage of a PV Module
A_{size}	Size of a Look-Up-Table

RAM	Random Access Memory
$I_{PVout}[i]$	Digital Output Reference Current
I_{SC}	Short Circuit Current of a PV Module
CCM	Continuous Conduction Mode
DCM	Discontinuous Conduction Mode
ESR	Equivalent Series Resistance
$G_{vd}(s)$	Small Signal Transfer Function Of Output Voltage Over Duty Ratio
$G_{id}(s)$	Small Signal Transfer Function Of Inductor Current Over Duty Ratio
$G_{iv}(s)$	Small Signal Transfer Function Of Output Current Over Output Voltage
L	Inductance
C	Capacitance
R_{ESR}	ESR of a Capacitor
R_L	DC Resistance of an Inductor
R	Load Resistance
$d[n+1]$	Predicted Duty Ratio for the Next Switch Cycle
$d[n]$	Duty Ratio for the Current Switch Cycle
$I_L[n]$	Measured Inductor Current
$I_{ref}[n]$	Reference Inductor Current for the Current Switch Cycle
T_s	Switching Period
$F_m(z)$	Predictive Current Mode Small Signal Transfer Function
$T(z)$	Discrete Time Loop Gain of the Plant
$G_{vd}(z)$	Discrete Time Equivalent of $G_{vd}(s)$
$G_{id}(z)$	Discrete Time Equivalent of $G_{id}(s)$
$G_{iv}(z)$	Discrete Time Equivalent of $G_{iv}(s)$
$F_m(s)$	Continuous Time Equivalent of $F_m(z)$
$G_c(s)$	Compensator
$G_c(s)$	Discrete Time Equivalent of $G_c(s)$
$T_{sys}(s)$	Loop Gain of Whole System
G_{LUT}	Small Signal Gain from LUT
V_{bat}	Input Battery Voltage
D_b	Boost Converter Duty Ratio

V_b	Output Voltage of Boost Converter
V_{out}	Output Voltage
D_{bk}	Buck Converter Duty Ratio
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
ΔI_L	Peak-to-Peak Inductor Current Ripple
f_{sw}	Switching Frequency
ΔV_{out}	Peak-to-Peak Capacitor Voltage Ripple
PCB	Printed Circuit Board

1. INTRODUCTION

Solar energy is one of the few renewable energy sources that have gained broad popularity around the world due to its high predictability and availability. One way to harness solar energy is by converting it to electrical energy using photovoltaic (PV) modules. One challenge when interfacing devices to PV modules is that the outputs of PV modules are non-linear. This non-linear output characteristic of PV modules requires devices that interface with them to have maximum power point tracking (MPPT) in order to maximize the efficiency of PV modules [1, 2]. Field testing devices with PV modules can be very challenging. First, PV modules are still expensive despite the fact that their price has been on a steady decline over the past decade. Installation of PV modules is also an expensive and time-consuming process. Second, the outputs of PV modules are dependent on the insolation level and temperature, which vary depending on the time of year and weather of the day. As a result, test conditions cannot be controlled, and repeatable test results cannot be obtained.

An alternative to field testing a PV system is to replace the PV module with a PV simulator, which is a device that emulates the output characteristic of a PV module. A PV simulator can be used within a lab environment at any time of the year. A few methods have been devised to simulate a PV module. The first method is to amplify the output of a PV cell or photodiode using analog amplifier circuits [3]. This requires the use of a device that can simulate natural sunlight. The second method is to build an equivalent circuit of a PV module using transistor and resistor networks and to amplify the signal from the equivalent circuit with analog circuits and a DC-DC converter [4, 5]. The last method is to digitally implement the output characteristic of a PV module through a look-up-table (LUT) that resides in the memory of a microcontroller and convert the digital signal to power output though a DC-DC converter [6-10]. This last method has gained increased popularity due to the ever increasing speed and capabilities of microcontrollers and their declining cost. The major advantage of digitally implemented PV simulators is that they provide a controlled environment where users can set conditions such as temperature, insolation level, the type of PV modules and the shading scenarios.

Digital control of a pulse-width modulation (PWM) converter is made possible with the inclusion of a microcontroller in the design. A digital system has the advantage of high noise immunity, immunity to analog component variations, fast design process and programmability for multiple applications, as compared to an analog counterpart [11-13]. With digital control, more complex control schemes can be easily achieved, such as the double current mode controller described in the present work. However, digital control limits the bandwidth of a system due to calculations delays and zero order hold actions originated from microcontrollers and sampling delays from analog-to-digital converters (ADCs) [14, 15]. Direct implementation of analog controllers in the digital domain would require relatively high switching frequency in order to get similar performance, which is not practical due to limited calculation capabilities of microcontrollers and excessive power loss in DC-DC converters. Different types of predictive control methods have been proposed to address calculation delay limitations of microcontrollers [11, 12, 14, 16]. They have been proven to alleviate the negative effect that calculation delay has on system bandwidth.

In this thesis, a PV simulator is implemented using a microcontroller and a DC-DC converter. The IV characteristic of a PV module is implemented as a LUT. The DC-DC converter is controlled by a double current mode controller that consists of an inner predictive current mode controller and an outer proportional-integral (PI) controller. This is a new approach to create a stable and reliable PV simulator.

In Chapter 2, the mathematical model of a PV module is derived and two methods to implement the mathematical model are proposed. In Chapter 3, the small signal transfer functions are derived, and the inner predictive current mode controller and the outer PI controller are designed to satisfy gain margin and phase margin requirements. Computer simulation of the PV simulator is done in Chapter 4. In Chapter 5, the hardware setup of the PV simulator is described in terms of the DC-DC converter topology, component selection and the requirements of the digital signal processor (DSP). The experimental results of the PV simulator prototype are shown in Chapter 6.

2. PV MODULE IV CURVE IMPLEMENTATION

2.1. PV MODULE MODELING

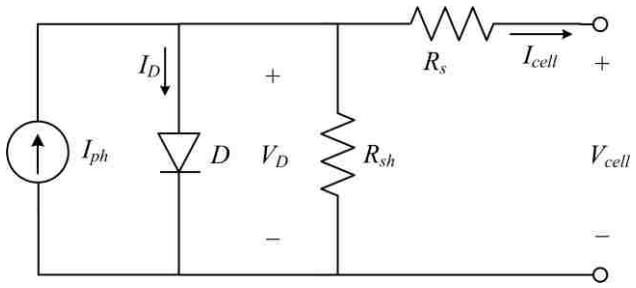

Figure 2.1. Equivalent circuit model of a PV cell [17]

A PV Cell can be modeled as a network that consists of a current source, a diode, a shunt resistor and a series resistor configured as shown in Fig. 2.1 [17]. The diode equation is given as [18]

$$I_D = I_0(e^{\frac{V_D}{V_t}} - 1) \tag{1}$$

$$V_t = \frac{AkT}{q} \tag{2}$$

where I_D is the current though the diode, V_D is the voltage across the diode, I_0 is reverse saturation current, V_t is thermal voltage, A is ideality factor, k is Boltzmann's constant, T is absolute temperature and q is elementary charge. A PV module is made of many series-connected PV cells as shown in Fig. 2.2. As a result, the voltage across each cell is the total output voltage of the PV module divided by the number of series-connected cells, and the current through each PV cell is the same as the output current of the PV module.

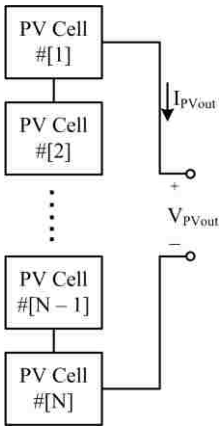

Figure 2.2. A PV module consists of series-connected PV cells

The output equation that relates the output current and output voltage of a PV module can be derived as

$$I_{PVout} = I_{ph} - I_0 \left(e^{\frac{\frac{V_{PVout}}{N} + I_{PVout} R_s}{V_t}} - 1 \right) - \frac{\frac{V_{PVout}}{N} + I_{PVout} R_s}{R_{sh}} \tag{3}$$

$$I_{ph} = \alpha \cdot I_{ph(\max)} \tag{4}$$

$$\alpha = \frac{\Lambda}{1000 \, W/m^2} \tag{5}$$

where I_{PVout} is the output current of the PV module, V_{PVout} is the output voltage of the PV module, R_s is the series resistance of the PV cell model, R_{sh} is the shunt resistance of the PV cell model, N is the number of series-connected cells, I_{ph} is the photo-generated current which is proportional to the amount of insolation received by the PV module as

shown in (4) and (5), $I_{ph(max)}$ is the photo-generated current at 100% insolation, and Λ is the irradiance from the sun received by the PV module, which is normalized to standard test condition (STC) of 1000 W/m^2 to find α, the insolation percentage. From (1)-(3), the IV curve of a PV module is characterized by R_s, R_{sh}, N, I_{ph}, A, T and I_0. By varying these parameters, one can model many different kinds of PV modules under different environmental conditions.

However, solar panel datasheets generally do not specify the parameters used in (3) explicitly, so these parameters must be extracted from other specified parameters from the datasheets. One technique requires only three values: the open circuit voltage, the short circuit current and the maximum power point (MPP) voltage and current [19]. The core of this technique involves stepping through all possible values of R_s and R_{sh} until the requirement that the derivative of power over voltage is zero is satisfied at the MPP. The Matlab[1] code to implement this technique is attached in Appendix C. The datasheet parameters and the extracted equivalent circuit model parameters have been compiled in Table 1 for the PV module SW-S85P from SunWize in Chapter 6.

2.2. IV CURVE IMPLEMENTATION

The PV module IV characteristic is represented by a LUT, which is updated whenever the controller receives a command to change the operating conditions, such as a different insolation level. In order to store the IV curve as a LUT in memory, a set number of matching output voltage and output current pairs has to be determined according to (3). For the double current mode controller to be presented in this paper, an output current reference is generated according to a measured output voltage. For simplicity, the output voltage corresponding to the i^{th} element of the LUT is set as

$$V_{PVout}[i] = i\frac{V_{OC}}{A_{size}}, i = 0,1,2,\cdots,(A_{size}-1) \tag{6}$$

[1] Matlab is a registered trade mark of The Math Works, Inc.

where V_{OC} is the open circuit voltage of the IV curve and A_{size} is the size of the LUT. A_{size} is limited by the random access memory (RAM) size of the microcontroller and can be increased if higher resolution is necessary and RAM permits. The i^{th} element of the LUT, which contains the output current reference $I_{PVout}[i]$, can be calculated according to (3). However, (3) has no analytical solution because I_{PVout} is on both sides of the equation and in a transcendental function, so Newton's method is used to find the numeric solutions. Equation (3) can be rearranged into

$$f\left(I_{PVout}[i]\right) = I_{PVout}[i] - I_{ph} + I_0\left(e^{\frac{\frac{V_{PVout}[i]}{N} + I_{PVout}[i]\cdot R_s}{V_t}} - 1\right) + \frac{\frac{V_{PVout}[i]}{N} + I_{PVout}[i]\cdot R_s}{R_{sh}} \quad (7)$$

with the output current reference $I_{PVout}[i]$ as its independent variable. The i^{th} LUT entry is the root of $f(I_{PVout}[i])$=0. Newton's method is given as [20]

$$I_{PVout}[i][n+1] = I_{PVout}[i][n] - \frac{f\left(I_{PVout}[i][n]\right)}{f'\left(I_{PVout}[i][n]\right)} \quad (8)$$

where

$$f'\left(I_{PVout}[i][n]\right) = I_0\left(e^{\frac{\frac{V_{PVout}[i]}{N} + I_{PVout}[i][n]\cdot R_s}{V_t}}\right)\frac{R_s}{V_t} + \frac{R_s}{R_{sh}} + 1 \quad (9)$$

.

$I_{PVout}[i][n+1]$ is the approximation to the real root of $f(I_{PVout}[i])$=0 after $n+1$ iterations. The initial guess $I_{PVout}[i][0]$ can be set to the short circuit current of the IV curve.

Once the array is filled up with voltage and current pairs, it may be used to represent the PV module. The index i to the array is calculated according to

$$i = round\left(A_{size}\frac{V_{out}}{V_{OC}}\right) \quad (10)$$

where V_{out} is the output voltage of the PV simulator.

Newton's method gives accurate solutions and only the output voltage is required to calculate the corresponding output current, which means only the output voltage needs to be measured to generate the index *i*. However, setting up the LUT requires a considerable amount of computing power from the microcontroller, which is a limited resource in embedded systems. This makes real time changes to the IV curve implausible due to large calculation delays.

Another technique to calculate the voltage and current pairs involves expressing the output current as a function of the diode voltage as

$$I_{PVout} = I_{ph} - I_0\left(e^{\frac{V_D}{V_t}} - 1\right) - \frac{V_D}{R_{sh}} \tag{11}$$

$$V_D = \frac{V_{PVout}}{N} + I_{PVout}R_s \tag{12}$$

V_D becomes the indexed voltage to the LUT, and its value corresponding to the i^{th} element of the LUT is set as

$$V_D[i] = i\frac{\frac{V_{OC}}{N} + I_{SC}R_S}{A_{size}} \tag{13}$$

where I_{SC} is the short circuit current of the IV curve. Then $I_{PVout}[i]$ is calculated by

$$I_{PVout}[i] = I_{ph} - I_0(e^{\frac{V_D[i]}{V_t}} - 1) - \frac{V_D[i]}{R_{sh}} \tag{14}$$

The index *i* to the LUT is generated according to

$$i = round(A_{size} \frac{V_{out} + I_{out} R_s N}{V_{OC} + I_{SC} R_s N}) \quad (15)$$

One drawback of this technique is that it requires the knowledge of both the output voltage and output current, which requires an extra current sensor besides an output voltage sensor, to generate the index *i*. However, the extra current sensor is already in place because the output current is regulated. This method does not require iterations of Newton's method and therefore drastically reduces the calculation time required to setup the LUT. The implementation of the PV simulator described below incorporates the abilities to simulate real time insolation and temperature changes, which is achieved with the fast method of (11) to (15). The process to setup the LUT is outlined in Fig. 2.3. The final current values that are stored in the LUT and their corresponding voltage values are shown in Fig. 2.4.

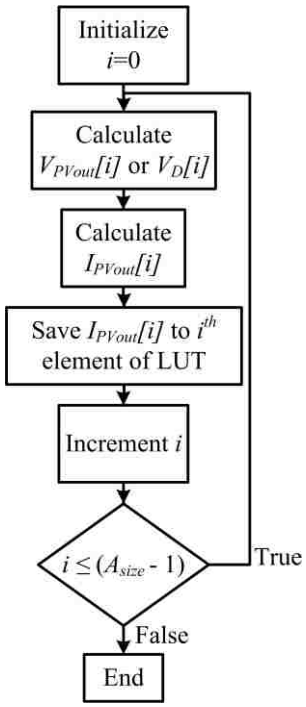

Figure 2.3. The LUT setup process

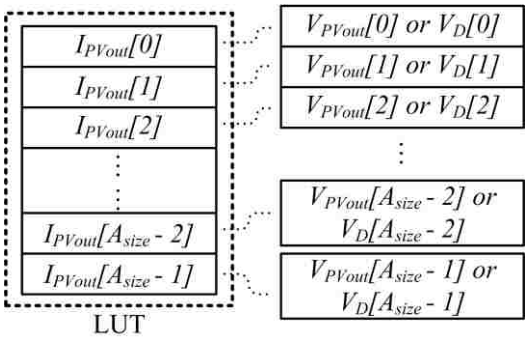

Figure 2.4. Final current values of the LUT and their corresponding voltage values

The IV curves of the PV module SW-S85P corresponding to 100%, 80%, 60%, 40% and 20% insolation levels along with their MPPs are shown in Fig. 2.5 with the diode voltage method. The same IV curves can also be obtained with Newton's method.

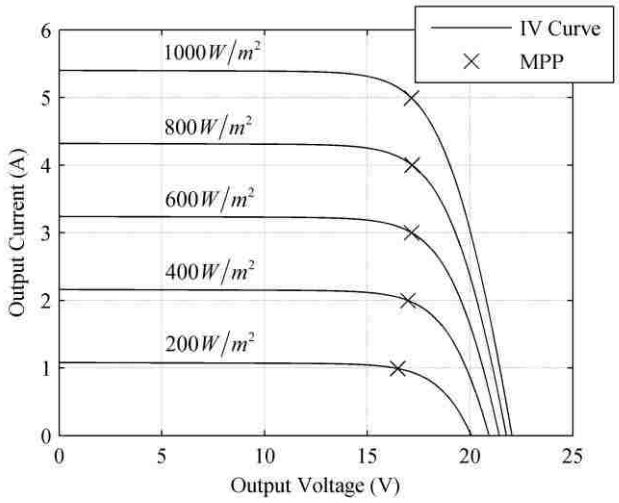

Figure 2.5. IV curves of the PV module SW-S85P using equivalent circuit modeling

To implement real time insolation and temperature changes to the IV curve, the LUT must be modified in real time; however, the LUT cannot be generated within one switching cycle, so the reference current signal would come from a partially modified LUT with discontinuities in it. This problem can be solved by using two separate LUTs. One LUT is used to generate the reference current and the other LUT is modified by the microcontroller in a background process to reflect insolation and temperature changes as shown in Fig. 2.6. Both processes shown in Fig. 2.6 run simultaneously. The variable *S* is used to determine the role of each LUT, and the symbol % is used to denote modulus operation.

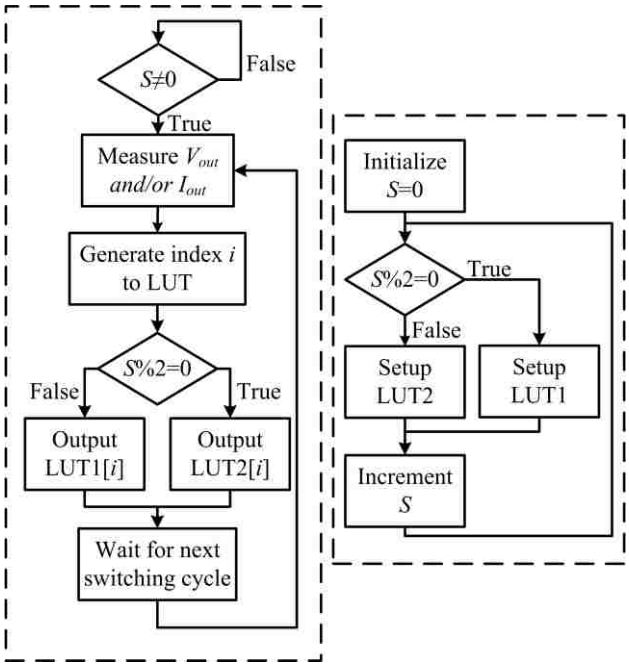

Figure 2.6. The IV curve implementation process

3. CONTROLLER IMPLEMENTATION

3.1. DC-DC CONVERTER IMPLEMENTATION

To achieve a variable output voltage, a synchronous buck converter topology is utilized as shown in Fig. 3.1. This topology has high efficiency over a wide operating range, a key feature given the need to reach both short-circuit and open-circuit conditions.

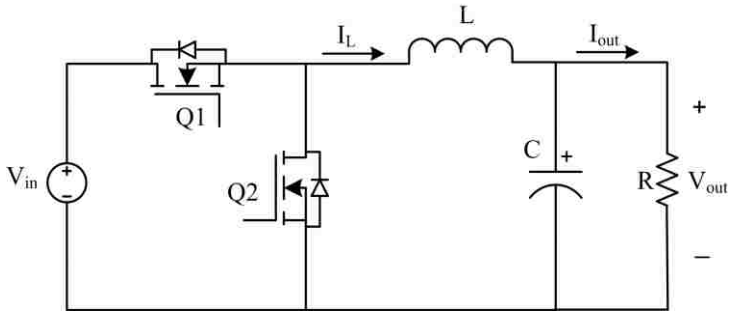

Figure 3.1. A synchronous buck converter

3.2. CONVERTER SMALL SIGNAL MODELING

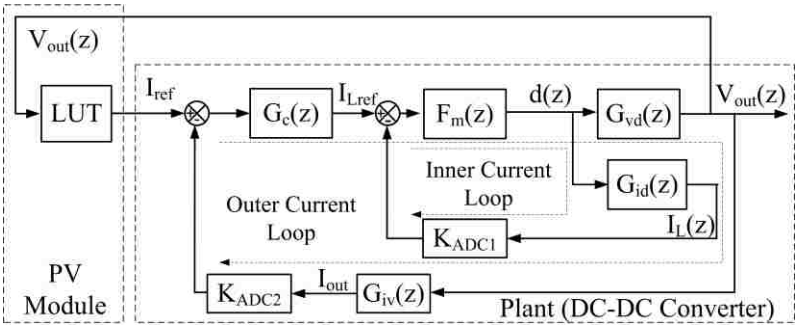

Figure 3.2. Control block diagram of the PV simulator

The control block diagram is shown in Fig. 3.2. In order to design a controller for the simulator, the small signal transfer functions of the buck converter must be derived. Here only the small signal transfer functions for the buck converter operating in continuous conduction mode (CCM) are needed because the buck converter is synchronous and will never run into discontinuous conduction mode (DCM). These small signal transfer functions can be derived using averaged switch modeling or state space averaging; both methods produce the same transfer functions [21]. The process accounts for the inductor DC resistance and equivalent series resistance (ESR) of the capacitor. Assuming a pure resistive load and that the inductor DC resistance and the ESR of the capacitor are much less than the load resistance, the continuous time small signal transfer functions are derived as

$$G_{vd}(s) = \left.\frac{\tilde{v}_{out}(s)}{\tilde{d}(s)}\right|_{\tilde{v}_{in}(s)=0}$$

$$= \frac{V_{in}R\left(1+CR_{esr}\cdot s\right)}{LC\left(R+R_{esr}\right)s^2+\left[L+CR_{esr}R_L+CR\left(R_{esr}+R_L\right)\right]s+R+R_L}$$

$$\approx \frac{V_{in}R\left(1+CR_{esr}\cdot s\right)}{LCR\cdot s^2+\left[L+CR\left(R_L+R_{esr}\right)\right]s+R+R_L} \qquad (16)$$

$$G_{id}(s) = \left.\frac{\tilde{i}_L(s)}{\tilde{d}(s)}\right|_{\tilde{v}_{in}(s)=0}$$

$$= \frac{V_{in}\left[1+C\left(R_{esr}+R\right)s\right]}{LC\left(R+R_{esr}\right)s^2+\left[L+CR_{esr}R_L+CR\left(R_{esr}+R_L\right)\right]s+R+R_L}$$

$$\approx \frac{V_{in}\left(1+CR\cdot s\right)}{LCR\cdot s^2+\left[L+CR\left(R_L+R_{esr}\right)\right]s+R+R_L} \tag{17}$$

$$G_{iv}(s)=\frac{\tilde{i}_{out}(s)}{\tilde{v}_{out}(s)}=\frac{1}{R} \tag{18}$$

where $G_{vd}(s)$ is the small signal transfer function of output voltage over duty ratio, $G_{id}(s)$ is the small signal transfer function of inductor current over duty ratio, $G_{iv}(s)$ is the small signal transfer function of output current over output voltage, L is the inductance, C is the capacitance, R_{ESR} is the ESR of the output capacitor, R_L is the DC resistance of the inductor, V_{in} is the input voltage, and R is the load resistance.

3.3. INNER PREDICTIVE CURRENT MODE CONTROLLER

Predictive current control is an accurate digital control technique that is based on inductor current predicted by sampled inductor current and output voltage. There are three kinds of predictive current mode controls – peak current control, average current control and valley current control. There are four kinds of modulation methods – trailing edge, leading edge, trailing triangle and leading triangle. Each of the three current control methods must be paired with the correct modulation method in order to be stable over the whole range of the duty ratio. Here average current control is used to increase noise immunity, and it is paired with trailing triangle modulation to give stability over the whole duty ratio range of the buck converter. The switching diagram is shown in Fig. 3.3. Sampling occurs at the beginning of each switching period. By the end of the second switching period, the average inductor current reaches the reference current set at the beginning of the first switching period. Regardless of the current control method, the predictive current mode control law for a buck converter is given as [12]

$$d\left[n+1\right]=-d\left[n\right]+\frac{L}{V_{in}T_s}\left(I_{ref}\left[n\right]-I_L\left[n\right]\right)+\frac{2V_{out}}{V_{in}} \tag{19}$$

where *d[n+1]* is the predicted duty ratio for the next switch cycle, *d[n]* is the duty ratio of the current switch cycle, $I_L[n]$ is the measured inductor current for the current switch cycle, $I_{ref}[n]$ is the reference inductor current for the current switch cycle, and T_s is the switching period. The small signal discrete time transfer function of duty ratio over current error is given as [13]

$$d(z) = F_m(z)[I_{ref}(z) - I_L(z)] \tag{20}$$

where

$$F_m(z) = \frac{L}{V_{in}T_s}\frac{1}{z+1} \tag{21}$$

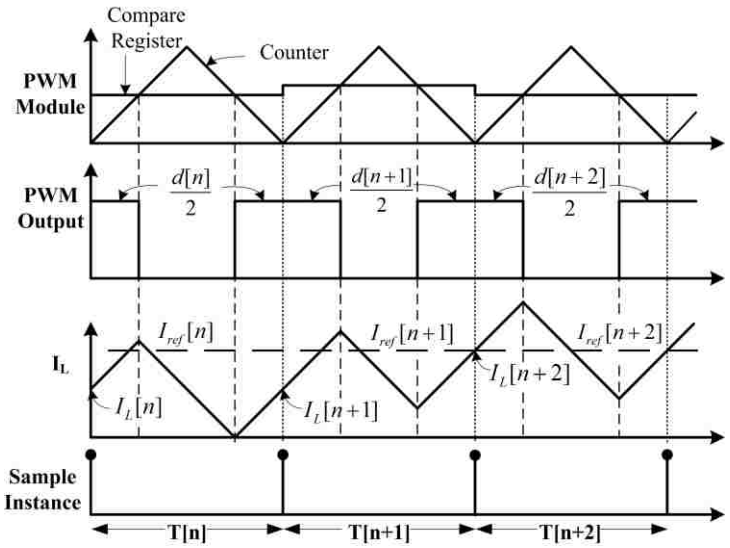

Figure 3.3. Average predictive current mode switch diagram

3.4. OUTER PI CONTROLLER DESIGN

From the block diagram shown in Fig. 3.2, the discrete time loop gain of the plant without the compensator $G_c(z)$ can be found using loop reduction or Mason's Law. Assuming that the gains from the ADCs have been adjusted to unity, the discrete time loop gain of the plant is found as

$$T(z) = \frac{F_m(z)G_{vd}(z)G_{iv}(z)}{1 + F_m(z)G_{id}(z)} \tag{22}$$

where $G_{vd}(z)$, $G_{id}(z)$ and $G_{iv}(z)$ are the discrete time equivalents of $G_{vd}(s)$, $G_{id}(s)$ and $G_{iv}(s)$ respectively. To design the compensator in the continuous time domain, the loop gain from (22) is converted to the continuous time domain by converting $F_m(z)$ to the continuous time domain $F_m(s)$ using bilinear transform as

$$F_m(s) = F_m(z)\Big|_{z=\frac{1+\frac{T_s}{2}s}{1-\frac{T_s}{2}s}} = \frac{L\left(1-\frac{T_s}{2}s\right)}{2T_sV_{in}} \tag{23}$$

Then the loop gain becomes

$$T(s) = \frac{F_m(s)G_{vd}(s)G_{iv}(s)}{1 + F_m(s)G_{id}(s)} = \frac{2L\left(1+CR_{esr}\cdot s\right)\left(1-\frac{T_s}{2}s\right)}{D(s)} \tag{24}$$

where

$$D(s) = 3CLRT_s\cdot s^2 + \left(4CRR_LT_s + 4CRR_{esr}T_s + 3LT_s + 2CLR\right)s + 2L + 4RT_s + 4R_LT_s. \tag{25}$$

The compensator $G_c(s)$ is designed based on the common PI controller

$$PI = K_P + \frac{K_I}{s} = \frac{K_p\left(s + \frac{K_I}{K_P}\right)}{s} \quad (26)$$

The small signal transfer functions of the buck converter are dependent on the load condition. To eliminate the need for mode switching or gain scheduling in the controller, control gain must ensure stability under the load conditions that place the most stringent requirements on the PI controller. If the load resistance is assumed to range from 0.25 Ω to 200 Ω, then a PI controller can be designed to satisfy both end point load conditions. Using Matlab, a range of operating conditions can be considered so that a PI controller can be produced to satisfy the requirements that the phase margin is greater than 75° and gain margin is greater than 6 dB. The zero of the PI controller is determined primarily by the light load condition, while the gain is determined primarily by the heavy load condition. Considering both, a stable controller may be found for the entire operating range. The PI controller is chosen to be

$$G_c(s) = \frac{2000(1 + 0.0039s)}{s} \quad (27)$$

The discrete time equivalent of $G_c(s)$ is transformed as

$$G_c(z) = G_c(s)\Big|_{s=\frac{2(z-1)}{T_s(z+1)}} = \frac{7.61z - 7.59}{z - 1} \quad (28)$$

The bandwidth of the system increases as load resistance decreases. The Bode plots of the compensated loop gains when the system is operating at minimum load resistance, maximum load resistance and MPP load resistance are shown in Fig. 3.4.

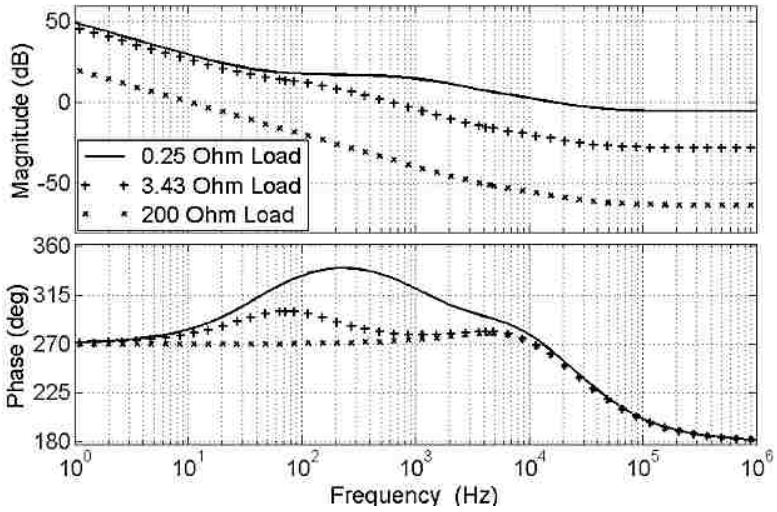

Figure 3.4. Compensated plant loop gains for multiple load conditions

3.5. SYSTEM STABILITY ANALYSIS

The continuous time loop gain of the whole closed system, which includes the LUT and the plant, is found to be

$$T_{sys}(s) = -G_{LUT}\frac{T(s)G_c(s)}{1+T(s)G_c(s)} \tag{29}$$

G_{LUT} is the small signal gain from the LUT. From (29), the stability of the whole system depends on the gain of the LUT since the plant is designed to be stable. The IV curve of the PV module to be simulated in Chapter 6 is shown in Fig. 3.5 along with the corresponding G_{LUT}. From Fig. 3.5, the absolute value of the IV curve slope increases as the voltage increases, and the worst case G_{LUT} is -1.797 S. At this worst case G_{LUT}, the loop gain of the whole closed system is shown in Fig. 3.6. As shown in Fig.3.6, the system still has adequate gain and phase margins and is stable.

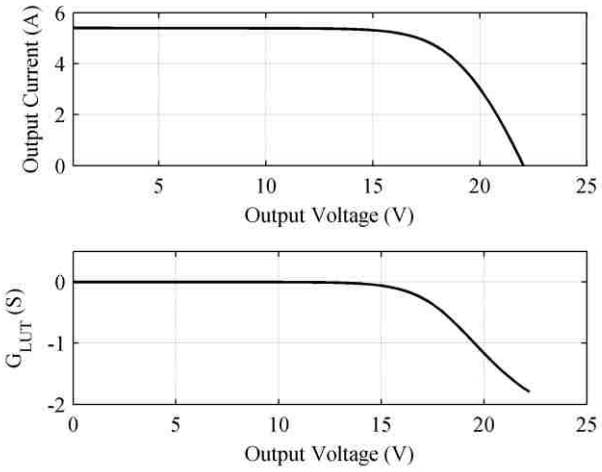

Figure 3.5. PV module IV curve and gain

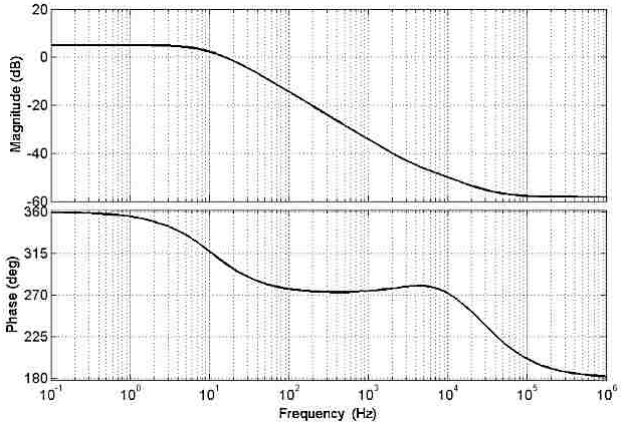

Figure 3.6. Bode plot of worst case system loop gain

4. SIMULATION RESULTS

Simulation is done in Matlab Simulink[2] with the help of the toolset PLECS[3]. The top level model schematic is shown in Fig. 4.1.

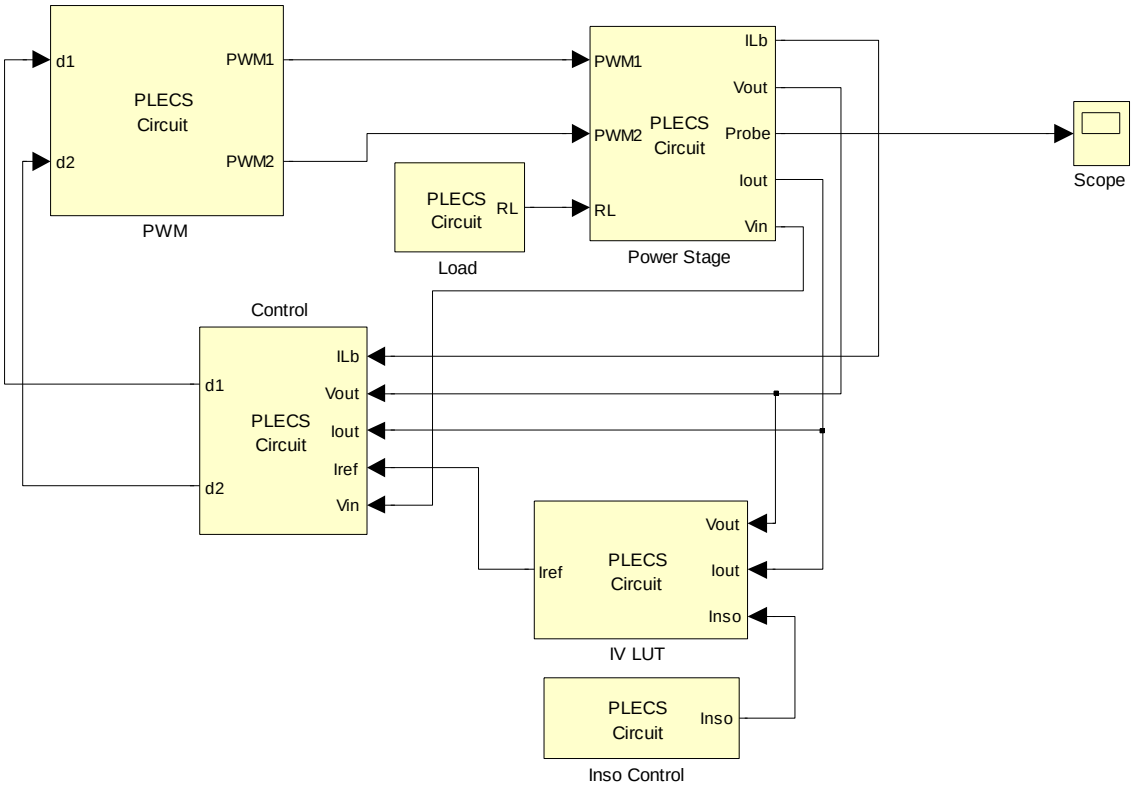

Figure 4.1. Top level schematic of the PV simulator simulation model

The "Power Stage" block shown in Fig. 4.2 houses the two stage DC-DC converter to be discussed in Chapter 5.

---

[2] Matlab Simulink is a registered trade mark of The Math Works, Inc.

[3] PLECS is a registered trade mark of Plexim GmbH.

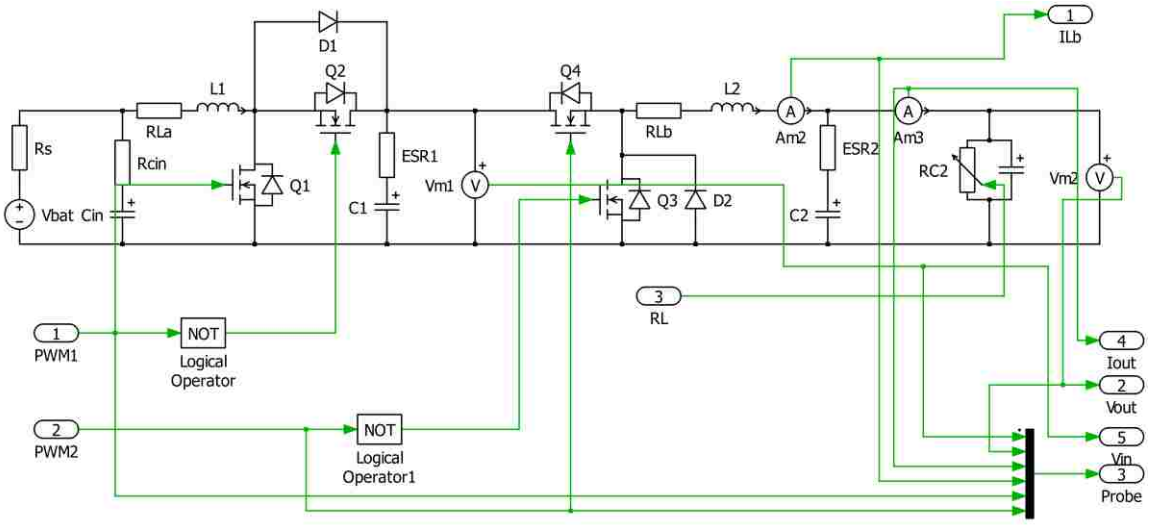

Figure 4.2. Power stage simulation model of the PV simulator

The circuit parameters used in the power stage simulation model are from those shown in Chapter 6 to match the experimental setup. The "IV LUT" block houses the IV curve LUT of the PV module. It is implemented through the "C-script" block available from the PLECS toolset as shown in Fig. 4.3.

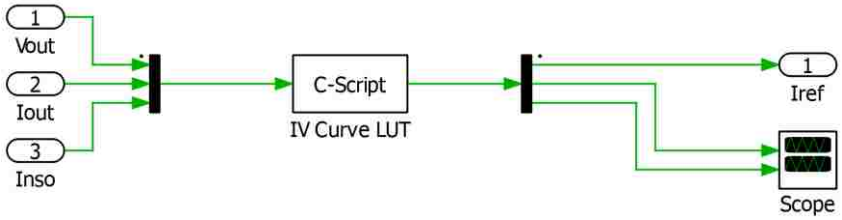

Figure 4.3. Inside of "IV LUT" block

The “C-script” block allows users to implement algorithms in C code. This greatly increases the capabilities and flexibilities of Simulink. The C code that goes in this “C-script” block can be extracted from the C codes shown in Appendix B. The “Control” block is implemented in a similar manner as shown in Fig. 4.4. The “Delay” block before the output “d2” is to simulate the zero-order hold action of a microcontroller.

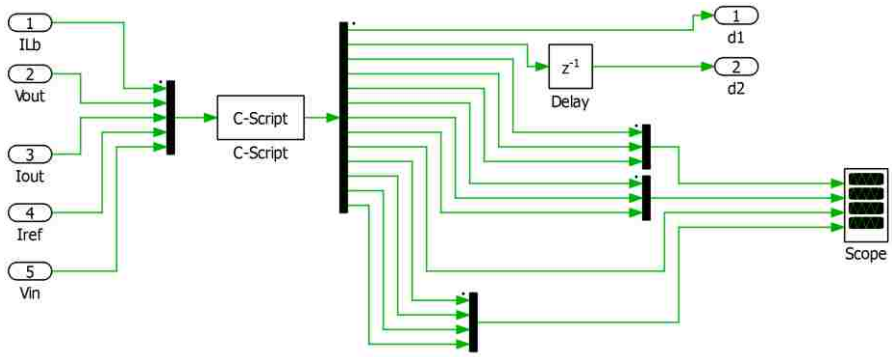

Figure 4.4. Control block simulation model for PV simulator

The steady state IV curve matching capability of the simulator is shown in Fig. 4.5 at insolation levels of 100%, 60% and 20%. The dynamic load step responses of the simulator at five different operating points on the IV curve as shown in Fig. 4.6 are shown from Fig. 4.7 to Fig. 4.11. The dynamic responses corresponding to insolation changes for three fixed load conditions are shown from Fig. 4.12 to Fig. 4.14.

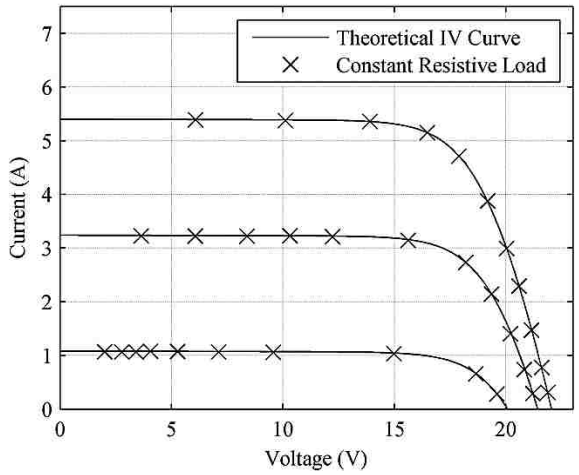

Figure 4.5. Computer simulated performance of the PV simulator

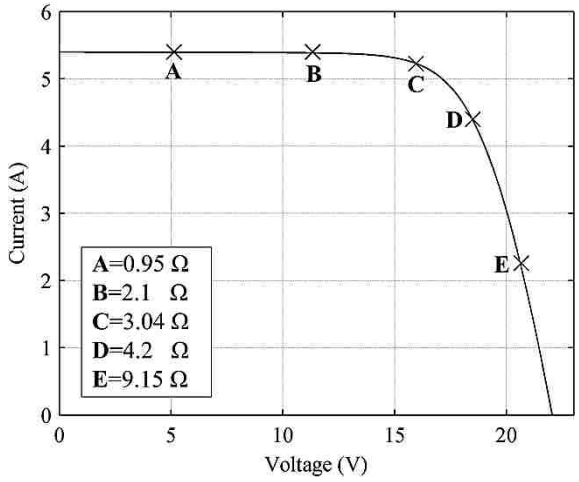

Figure 4.6. Dynamic load step response test operating points

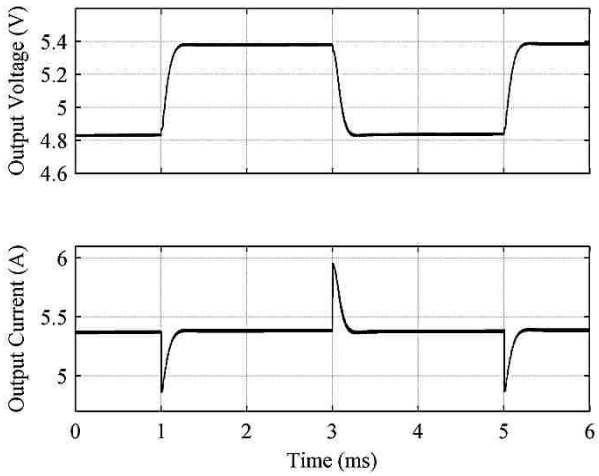

Figure 4.7. Computer simulated output waveforms of the PV simulator when load changes between 1.0Ω and 0.9Ω (operating point A)

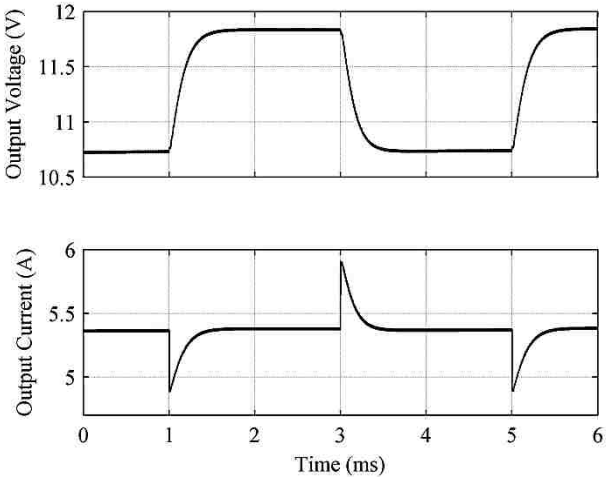

Figure 4.8. Computer simulated output waveforms of the PV simulator when load changes between 2.2Ω and 2Ω (operating point B)

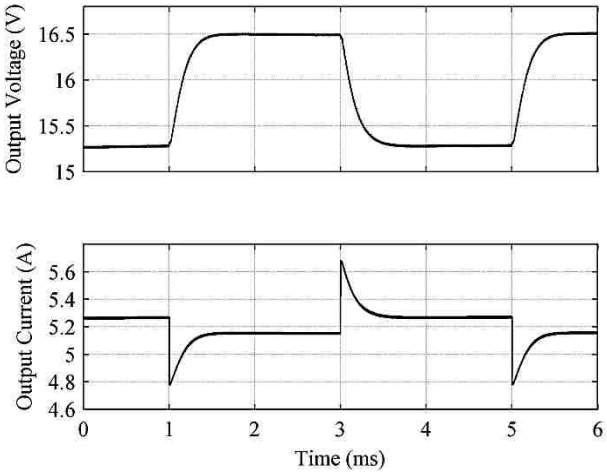

Figure 4.9. Computer simulated output waveforms of the PV simulator when load changes between 3.2Ω and 2.9Ω (operating point C)

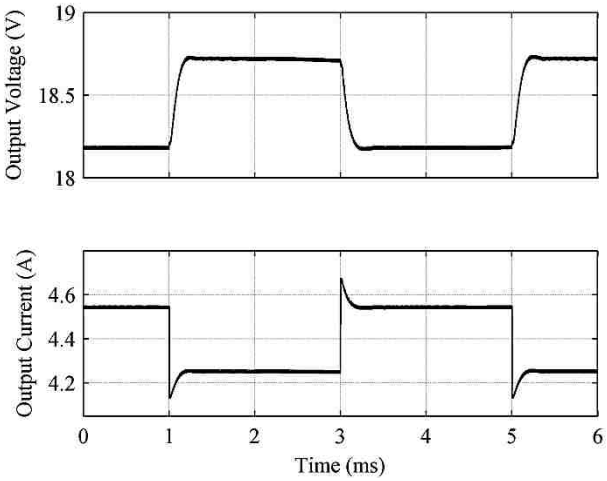

Figure 4.10. Computer simulated output waveforms of the PV simulator when load changes between 4.4Ω and 4Ω (operating point D)

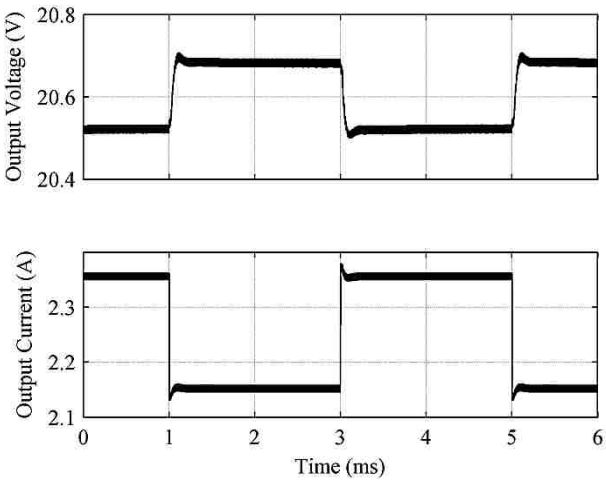

Figure 4.11. Computer simulated output waveforms of the PV simulator when load changes between 9.6Ω and 8.7Ω (operating point E)

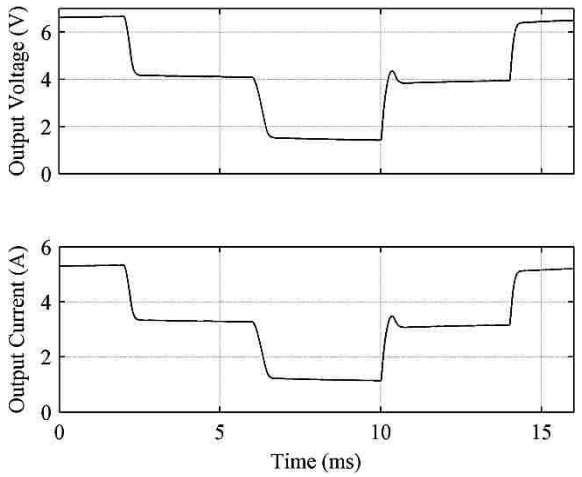

Figure 4.12. Computer simulated insolation change step response with load of 1.25Ω

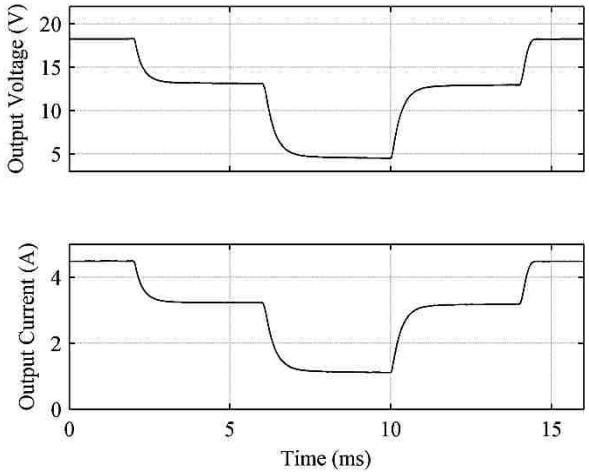

Figure 4.13. Computer simulated insolation change step response with load of 4.07Ω

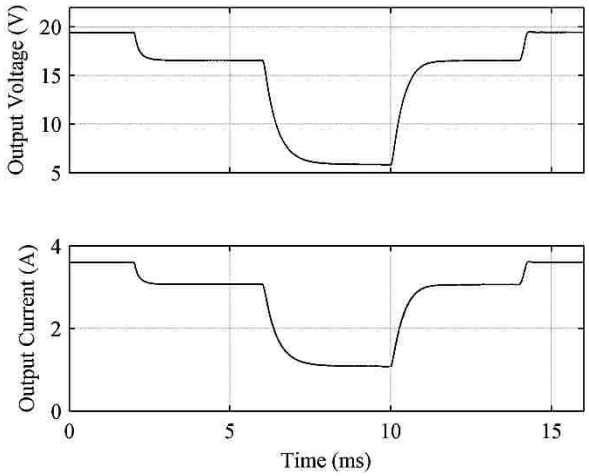

Figure 4.14. Computer simulated insolation change step response with load of 5.4Ω

5. HARDWARE IMPLEMENTATION

5.1. POWER STAGE OF THE PV SIMULATOR

The simulator is powered by a Li-ion battery pack to create a portable device; however, it can also be powered by a fixed or programmable DC power supply when they are available. A battery or DC power supply with output voltage higher than the open circuit voltage of the PV module to be simulated might not be readily available, so a two-stage synchronous DC-DC converter is devised as shown in Fig. 5.1.

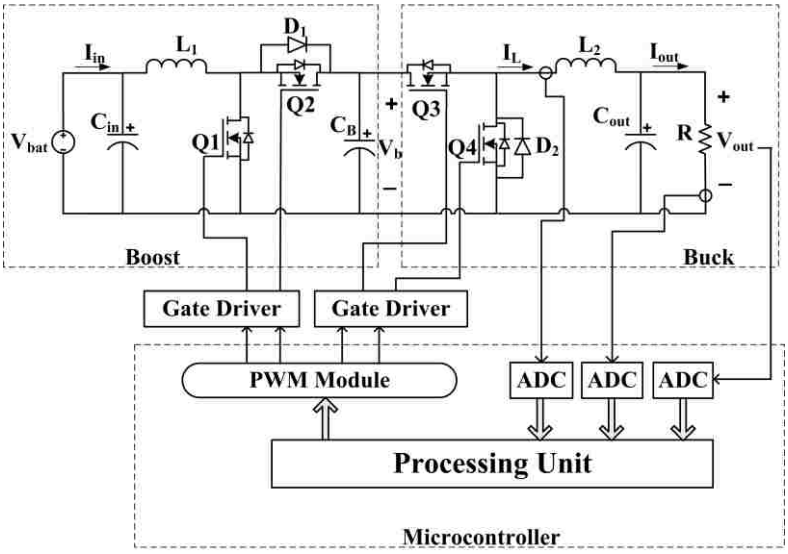

Figure 5.1. PV simulator hardware block diagram

The first stage is a step-up synchronous boost converter whose steady state output voltage is approximately

$$V_b = \frac{V_{bat}}{1 - D_b} \tag{30}$$

where V_{bat} is the input voltage from the battery or an equivalent power source, D_b is the boost converter duty ratio and V_b is the output voltage of the boost converter. For the prototype built, the boost converter outputs a nominal voltage of 30 V from a nominal input voltage of 14.8 V from the Li-ion battery pack. The second stage is a step-down synchronous buck converter whose output voltage is approximately

$$V_{out} = D_{bk} V_b \tag{31}$$

where V_{out} is the output voltage and D_{bk} is the duty ratio of the buck converter. For the prototype built, the buck converter has an output voltage range of 0 V to 30 V from a nominal input voltage of 30 V, which means D_{bk} will range from zero to unity. One major advantage of using a synchronous topology is high efficiency, which is essential for portable applications. Another advantage of using a synchronous topology here is that the DC-DC converter will never run into DCM. This allows for a much simpler controller implementation that does not need to account for both CCM and DCM, which are qualitatively different in nature and have different dynamic models.

However, with a synchronous topology, the number of MOSFETs used is increased from two to four, which means the required number of PWM signals also increases from two to four. Dead-bands must be introduced between Q1 and Q2, and between Q3 and Q4 to prevent short-circuiting of the filter capacitor C_B due to the inherent turn-on and turn-off delays of MOSFETs. Since the source nodes of the high-side MOSFETs Q2 and Q3 have to be floating, the gate drivers must be able to provide floating gate signals to these high-side MOSFETs. In the present design, the “low and high side” gate driver IR2110 from International Rectifier is used. This type of gate driver requires that the source node of the high-side MOSFETs be connected to the drain of the low-side MOSFETs. This means that the body diodes of Q2 and Q3 will be temporarily conducting whenever Q2 and Q3 are not fully turned on and are required to conduct current. The body diode of a MOSFET is inefficient because it has

comparatively high forward voltage and reverse recovery charge, so the Schottky diodes D1 and D2 are used to bypass the body diodes in Q2 and Q3 to increase efficiency.

5.2. INDUCTOR AND CAPACITOR SELECTION

The inductor current waveform of the buck converter is shown in Fig. 5.2.

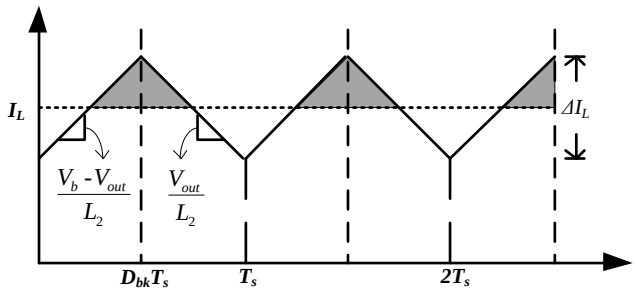

Figure 5.2. Buck converter inductor current

The ripple current then can be calculated by

$$\Delta I_L = \frac{D_{bk} T_s \left(V_b - V_{out}\right)}{L_2} \tag{32}$$

where ΔI_L is the inductor peak-to-peak current ripple, and L_2 is the buck converter inductor. Substituting (31) in (32) and solving for L_2 gives

$$L_2 = \frac{V_{out}\left(V_b - V_{out}\right)}{\Delta I_L f_{sw} V_b} \tag{33}$$

where f_{sw} is the switching frequency. When V_{out} is equal to half of V_b, L_2 is maximized. This leads to

$$L_2 = \frac{V_b}{4\Delta I_L f_{sw}}. \tag{34}$$

Equation (34) gives the minimum required inductance for a specified inductor peak-to-peak current ripple requirement for the proposed PV simulator. The minimum required output capacitor value to satisfy output voltage peak-to-peak ripple requirement is given as [21]

$$C_{out} = \frac{\Delta I_L}{8 f_{sw} \left(\Delta V_{out} - R_{ESR} \Delta I_L \right)}. \tag{35}$$

where ΔV_{out} is the peak-to-peak output voltage ripple. In the PV simulator prototype, the inductor value and capacitor value selected are to satisfy a peak-to-peak inductor current ripple of 1 A and a peak-to-peak output voltage ripple of 50 mV respectively.

Since the second stage buck converter acts as a low pass filter to the output voltage of the first stage boost converter, there is a lot of flexibility in the inductor and capacitor values for the boost converter. However, the boost converter acts as an input filter for the buck converter, so the output impedance of the boost converter must be less than the input impedance of the buck converter to ensure system stability. In order to avoid multi-stage instability, the filter capacitor C_b should be chosen to be as large as reasonable. In the PV simulator prototype, C_b is chosen to be twice as large as C_{out}, and L_1 is chosen to be the same as L_2.

5.3. THE DSP

The DSP is required to have three ADC channels, four PWM output channels, built-in or external RAM and a processing unit, preferably with floating point math capability. The ADCs are used to measure the output voltage, output current and the buck converter inductor current, as needed by the double current mode controller. The PWM signals that go into Q1 and Q2 should run in complementary mode, which means only one MOSFET can be turned on at a time, with preset dead-band. This also applies to the PWM signals that go into Q3 and Q4. For the PV simulator prototype, the DSP chosen is the 32-bit TMS320F28335 by Texas Instrument. The DSP runs at 150MHz and has a built-in floating-point-unit, 16 channels of 12-bit ADCs, and six PWM modules, each

with two PWM output channels that can run in complementary mode with preset dead-band.

6. EXPERIMENTAL RESULTS

6.1. PV SIMULATOR PROTOTYPE

The PV simulator has been built with the circuit parameters shown in Table 6.1. The physical layout of the circuit board is shown in Fig. 6.1.

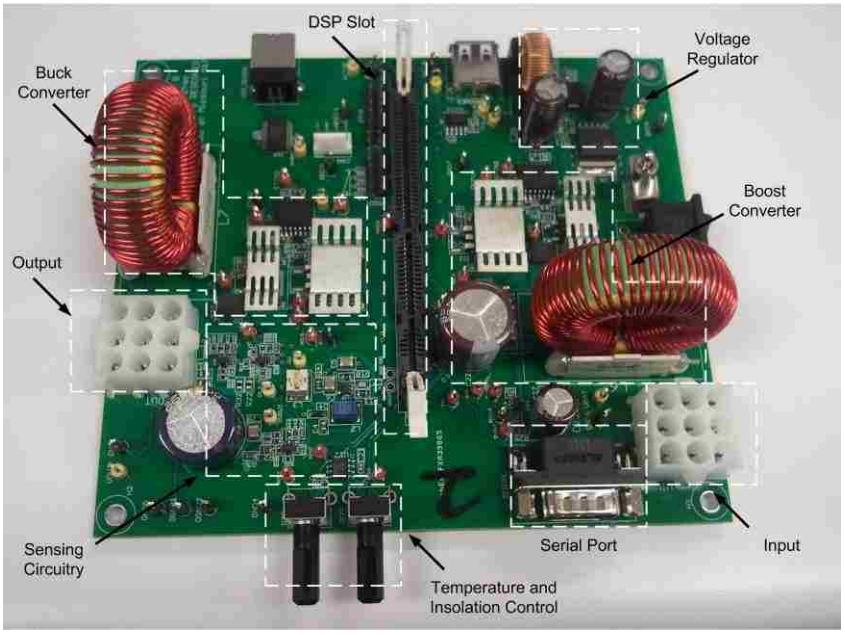

Figure 6.1. PV simulator prototype built

The main restriction to the frequency is the calculation speed of the microcontroller since it has to finish all required calculation before the next switching period starts. The PV simulator is set to simulate the multi-crystalline PV module SW-S85P from SunWize. Its datasheet parameters and the calculated parameters for the equivalent circuit model shown in Chapter 2 are given in Table 6.2. Note that the circuit parameters from the datasheet are taken under STC, where the temperature is 25 °C and the irradiance is 1000 W/m^2 (100% insolation).

Table 6.1. PV Simulator Parameter

DC-DC converter parameters	
Switching Frequency (f_{sw})	100 kHz
Inductor (L_1)	138 μH
Inductor (L_2)	138 μH
Inductor DC Resistance(R_{DC})	100 mΩ
Input Capacitor (C_{in})	560 μF
Filter Capacitor(C_B)	1 mF
Output Capacitor (C_{out})	560 μF
Output Capacitor ESR(R_{esr})	54 mΩ

Table 6.2. PV Module Parameters

Datasheet Parameters (STC)	
Number of Cells in Series (N)	72
Short Circuit Current (I_{sc})	5.4 A
Open Circuit Voltage (V_{oc})	22 V
Voltage at Max. Power (V_{mpp})	17.4 V
Current at Max. Power (I_{mpp})	4.9 A
Maximum Output Power (P_{mpp})	85 W
Model Parameters	
Internal Series Resistance (R_s)	342 mΩ
Internal Shunt Resistance (R_{sh})	1.115 KΩ
Reverse Saturation Current (I_0)	73.42 nA
Photo-generated Current (100% Insolation) (I_{ph})	5.402 A
Ideality Factor (A)	0.4728

The outputs of the PV simulator prototype when connected to different constant current loads are shown in Fig. 6.2 for 100%, 60% and 20% insolation levels. As shown in Fig. 6.2, the output of the PV simulator prototype follows the ideal IV curves consistently without running into stability issues.

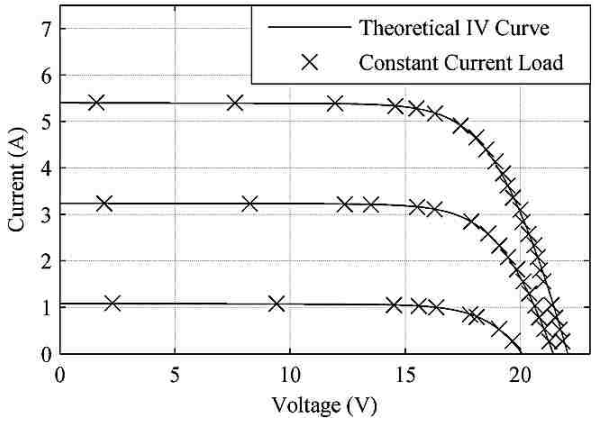

Figure 6.2. PV simulator prototype's performance

The experimental setup to test the dynamic response of the simulator is shown in Fig. 6.3, where R_f is the controlled load, Q_f is the controlled load switch and P is the gate control signal with a switching frequency of 100 kHz. The load step responses of the PV simulator prototype, corresponding to the five different operating points on the IV curve shown in Fig. 4.6, are shown from Fig. 6.4 to Fig. 6.8, where the top trace is the output voltage, the middle trace is the output current and the bottom trace is the load switch. Fig. 6.9 to Fig. 6.14 show the insolation step responses of the PV simulator prototype in the MPP region, where the top trace is the output current and the bottom trace is the output voltage. The load step response experimental results are comparable to the simulated load step responses shown in Chapter 4, as measured by rise time, fall time, settling time and overshoots. The maximum settling time of the prototype is about 500 μs. Therefore, the PV simulator prototype can be perturbed, as by a MPPT controller, with a sampling frequency of up to 2 kHz.

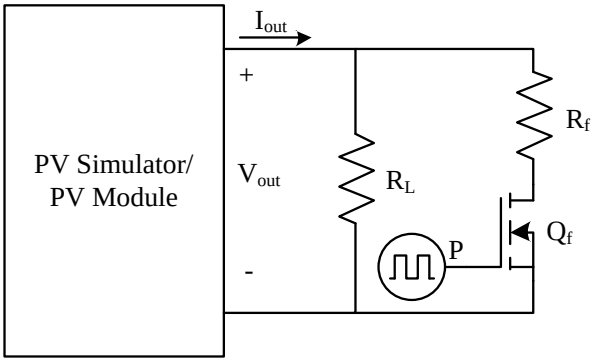

Figure 6.3. Experiment setup for the PV simulator/PV module load dynamic response

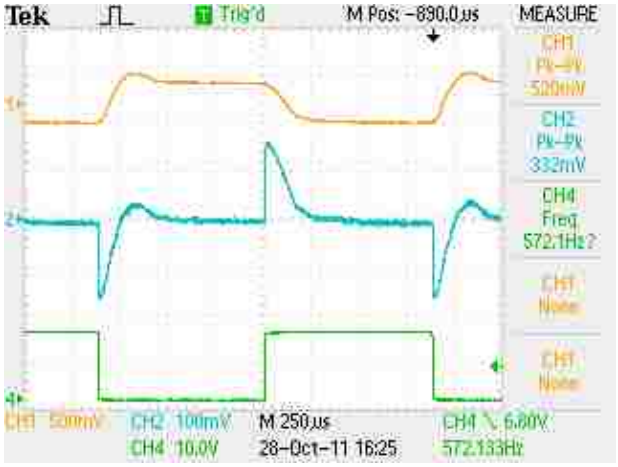

Figure 6.4. Step response when load changes between 1Ω and 0.9Ω.

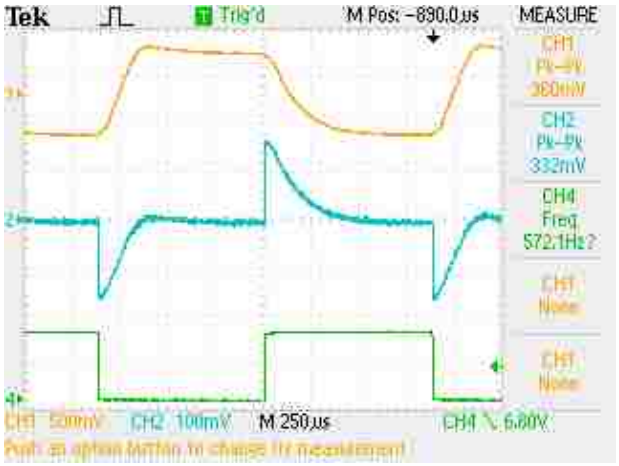

Figure 6.5. Step response when load changes between 2.22Ω and 2Ω.

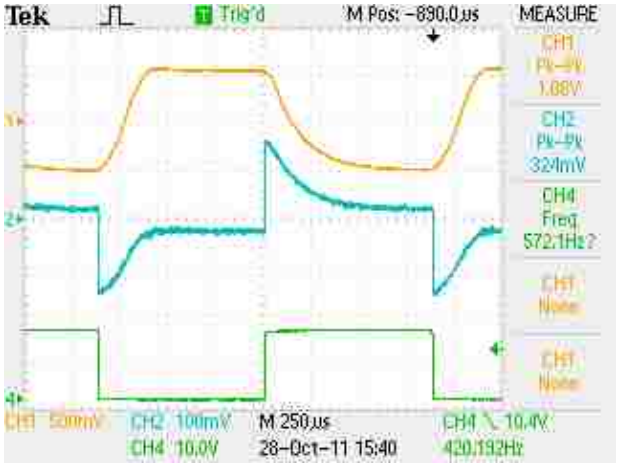

Figure 6.6. Step response when load changes between 3.2Ω and 2.9Ω.

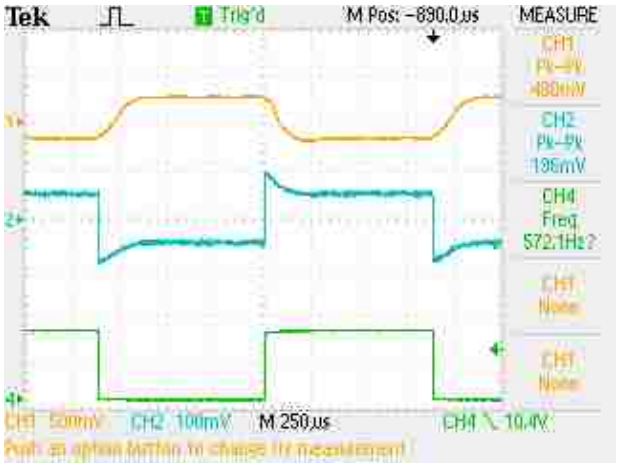

Figure 6.7. Step response when load changes between 4.4Ω and 4Ω.

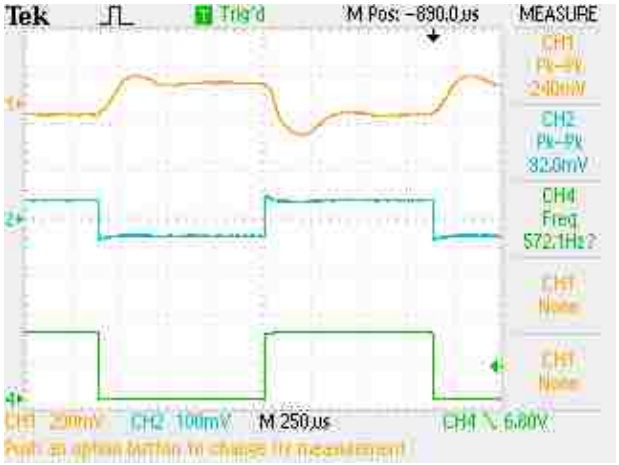

Figure 6.8. Step response when load changes between 9.6Ω and 8.7Ω.

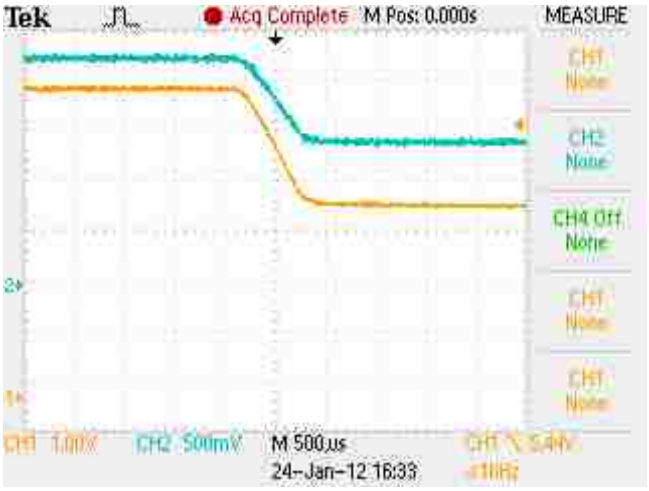

Figure 6.9. Insolation step change (Insolation 100% - 60%) (Load = 1.24Ω)

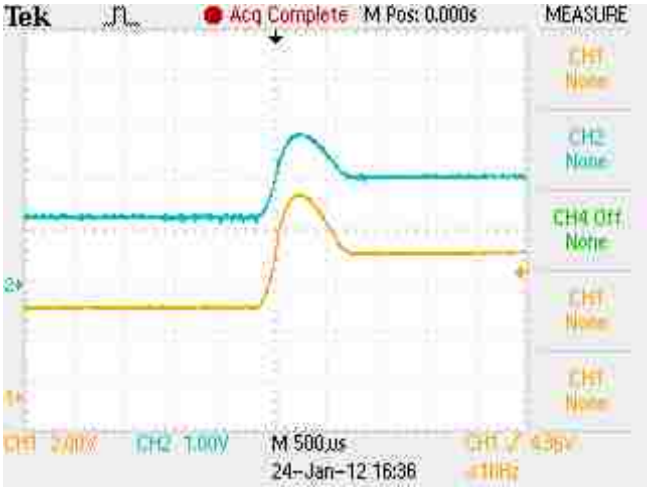

Figure 6.10. Insolation step change (Insolation 60% - 100%) (Load = 1.24Ω)

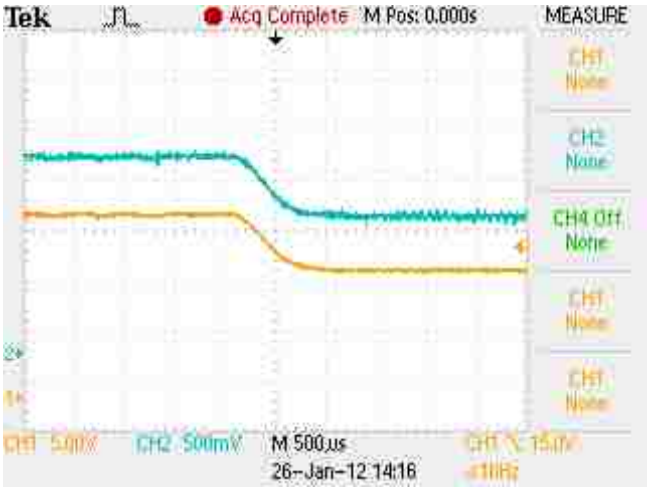

Figure 6.11. Insolation step change (Insolation 100% - 60%) (Load = 3.14Ω)

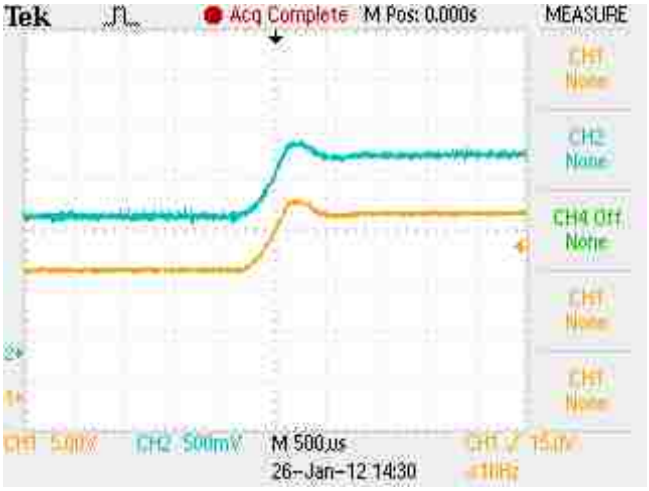

Figure 6.12. Insolation step change (Insolation 60% - 100%) (Load = 3.14Ω)

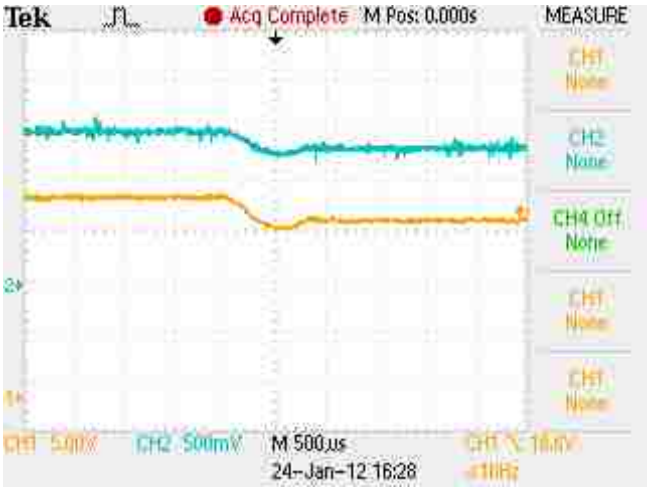

Figure 6.13. Insolation step change (Insolation 100% - 60%) (Load = 4.23Ω)

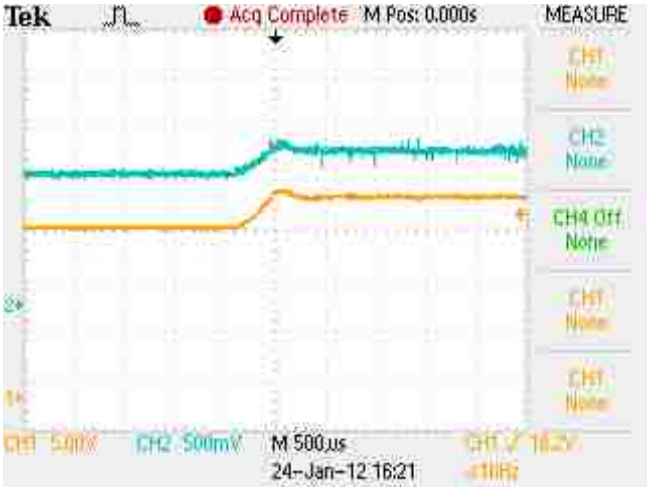

Figure 6.14. Insolation step change (Insolation 60% - 100%) (Load = 4.23Ω)

6.2. PV MODULE DYNAMIC RESPONSE

With the setup shown in Fig. 6.3, the load step responses of the PV module SW-S85P are determined and the results are shown from Fig. 6.15 to Fig. 6.17 with a load switching frequency of 100 kHz. The settling time increases as the output current

increases, and the output takes about 5 μs to settle for the worst case, which translates to a sampling frequency of 200 kHz.

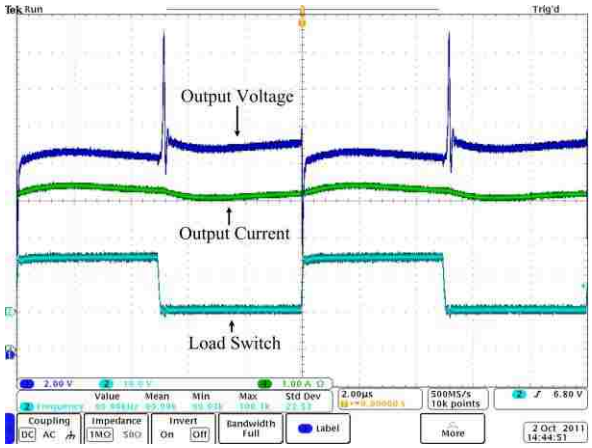

Figure 6.15. PV module step response when load changes between 2.34Ω and 2.14Ω

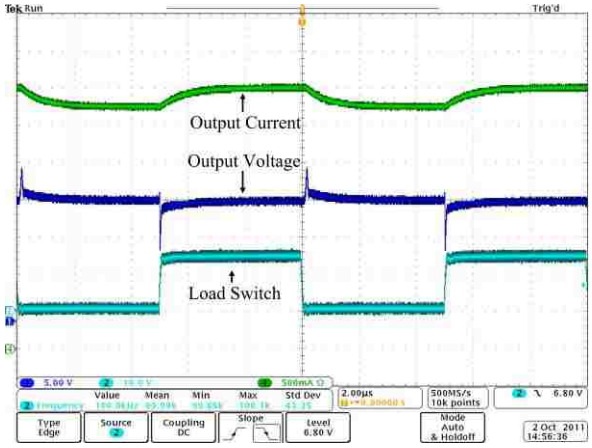

Figure 6.16. PV module step response when load changes between 4.23Ω and 3.67Ω

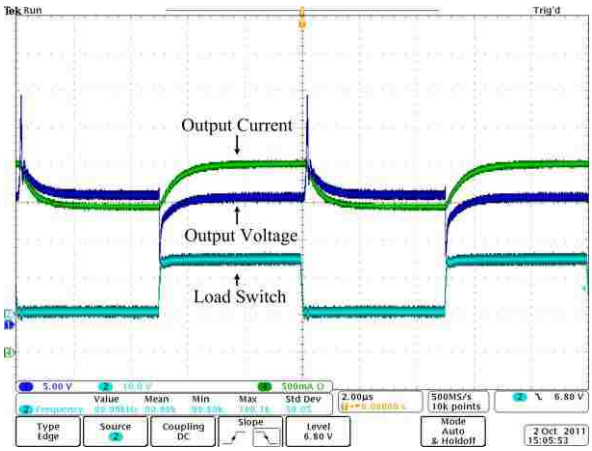

Figure 6.17. PV module step response when load changes between 8.46Ω and 7.33Ω

7. CONCLUSION

7.1. RESEARCH FINDINGS

This thesis presents a digitally implemented PV simulator which consists of a DC-DC converter and a DSP. The IV characteristic of the PV module of interest is implemented as a LUT that resides in the RAM of a microcontroller. A double current mode controller that consists of a PI controller and a predictive current mode controller is used to regulate the output current of the DC-DC converter to match the IV curve in the LUT. With a double current mode controller, the stability of the system is guaranteed over a wide range of load conditions. A portable PV simulator prototype of 85 Watts is built to demonstrate the effectiveness of the presented method. It is shown that the prototype has excellent IV curve matching capability, and it can be perturbed, as by a MPPT controller, with a sampling frequency of up to 2 kHz. The dynamic response of the physical PV module of interest is also examined, and it has a maximum settling time of approximately 5 μs, which translates to a maximum sampling frequency of 200 kHz. It is observed that the settling time of the PV simulator prototype is significantly more than the settling time of a real PV module. The effectiveness of the PV simulator prototype hence is dependent on the MPPT algorithm and its sampling frequency. Even though the PV simulator prototype falls short of the step response of a real PV module, this gap can be bridged by better component selection and faster microcontrollers.

7.2. FUTURE WORK

It is possible to improve the dynamic response of the simulator prototype by reducing the size of the output capacitor while at the same time satisfying the output voltage ripple requirement. For a fixed switching frequency, the selection of the output capacitor for the simulator prototype is dominated by the ESR, so the best way to reduce the size of the output capacitor is to replace it with a capacitor of a different material, such as Polymer Aluminum, which has lower ESR than Aluminum Electrolytic capacitors. Another way to reduce the size of the output capacitor is to reduce the inductor ripple by using large inductor values. However, larger inductor values mean lower rate of change of the inductor current, which could limit the system bandwidth. A

method to balance inductor and capacitor values to achieve maximum system bandwidth is of great value here. The last method to reduce the size of the output capacitor is to increase the switching frequency. However, higher frequency requires faster processors, better power management for the DC-DC converter and capacitors that are suitable for high frequency operation.

It is also beneficial to see if multiple PV simulators can be connected in a series or parallel configuration to enable emulation of higher power PV modules. This would greatly increase the flexibility of the PV simulator and give reasons for the mass production of a PV simulator of low output power, such as the prototype built.

A computer user interface, such as a graphic user interface or a command prompt, needs to be designed to let users input all the parameters about the PV module of interest and the environmental conditions of the simulation in order to run the PV simulator prototype in stand-alone mode. There is a serial port built into the PV simulator prototype, which is connected to the SCI channels of the DSP. This port will be connected to the RS-232 serial port of a computer. There is also a USB interface for the SCI of the DSP for computers that don’t have serial ports, as it is true for most modern computers.

APPENDIX A.
PRINTED CIRCUIT BOARD DESIGN

This appendix contains the EAGLE[4] schematics and printed circuit board (PCB) physical layout information for the photovoltaic simulator prototype built. Figures A.1 through A.4 show the schematics for the PV simulator prototype. The PCB consists of two layers and utilizes the ground plane construct technique. Figure A.5 and A.6 show the PCB physical top and bottom board layer of the PV simulator prototype. In the physical board layout, circles represent vias or through holes, and the dashed lines specify the ground plane borders.

[4] The EAGLE logo is a registered trademark of CadSoft Computer.

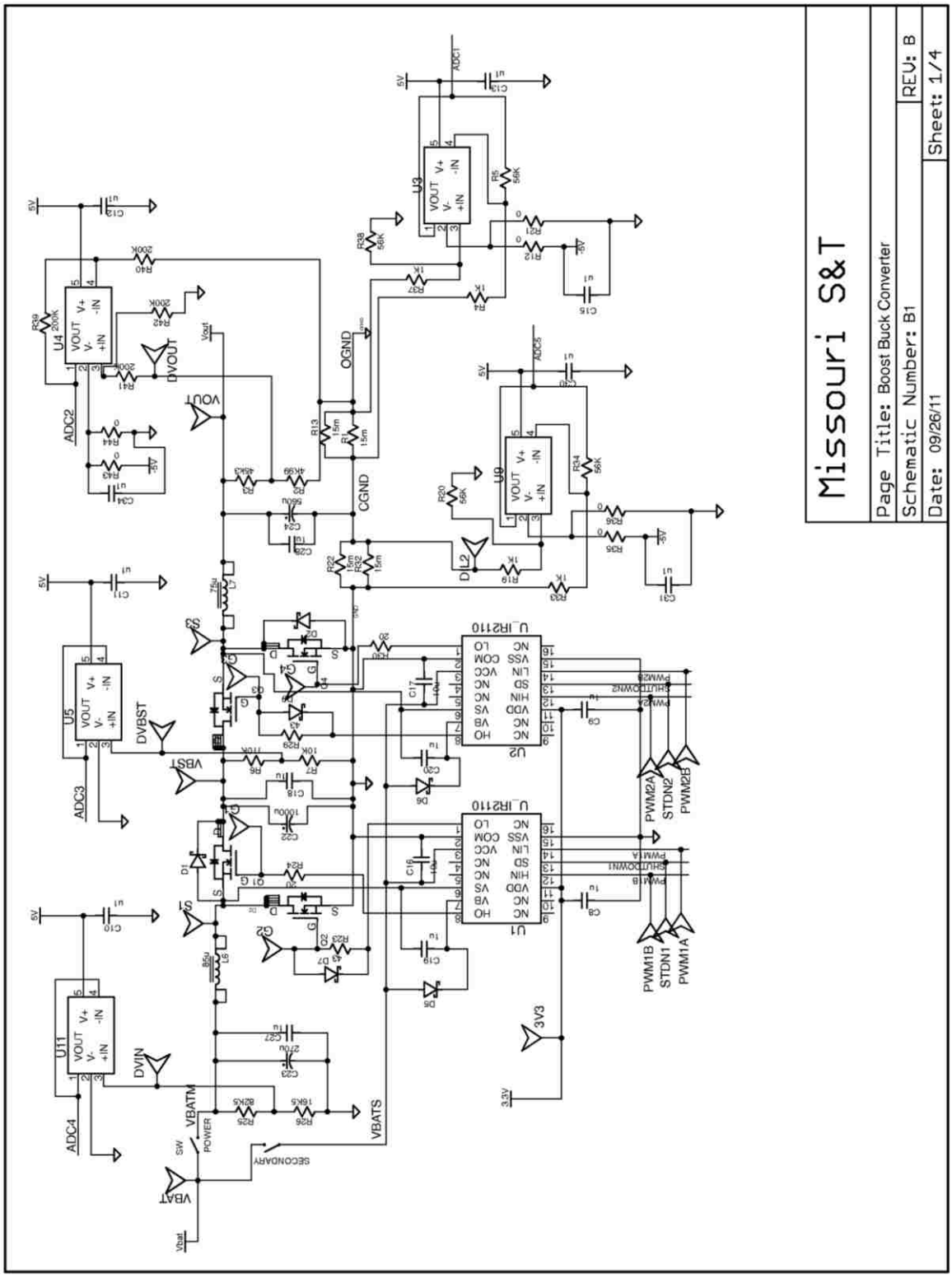

Figure A.1. PV simulator PCB schematic sheet one

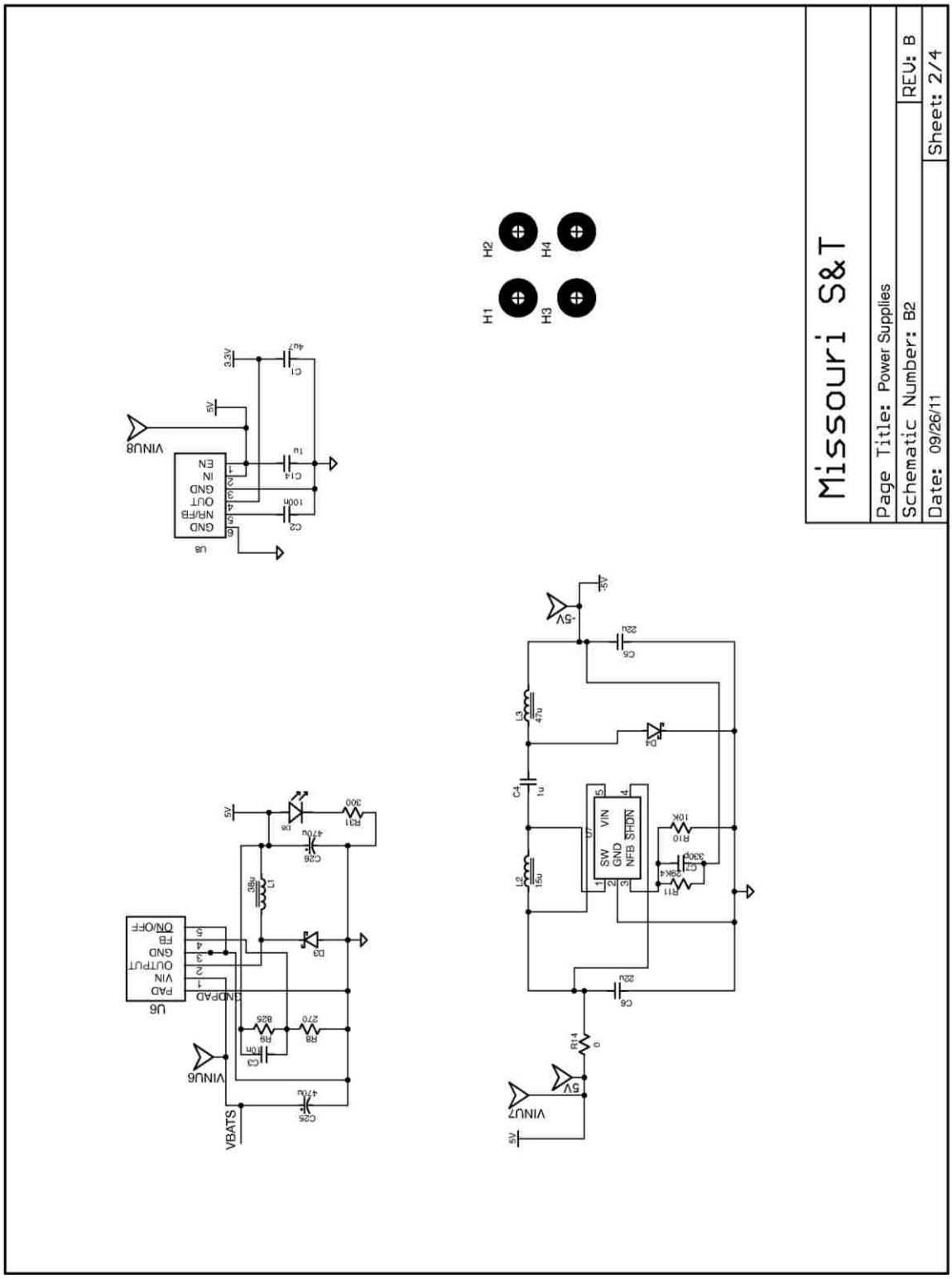

Figure A.2. PV simulator PCB schematic sheet two

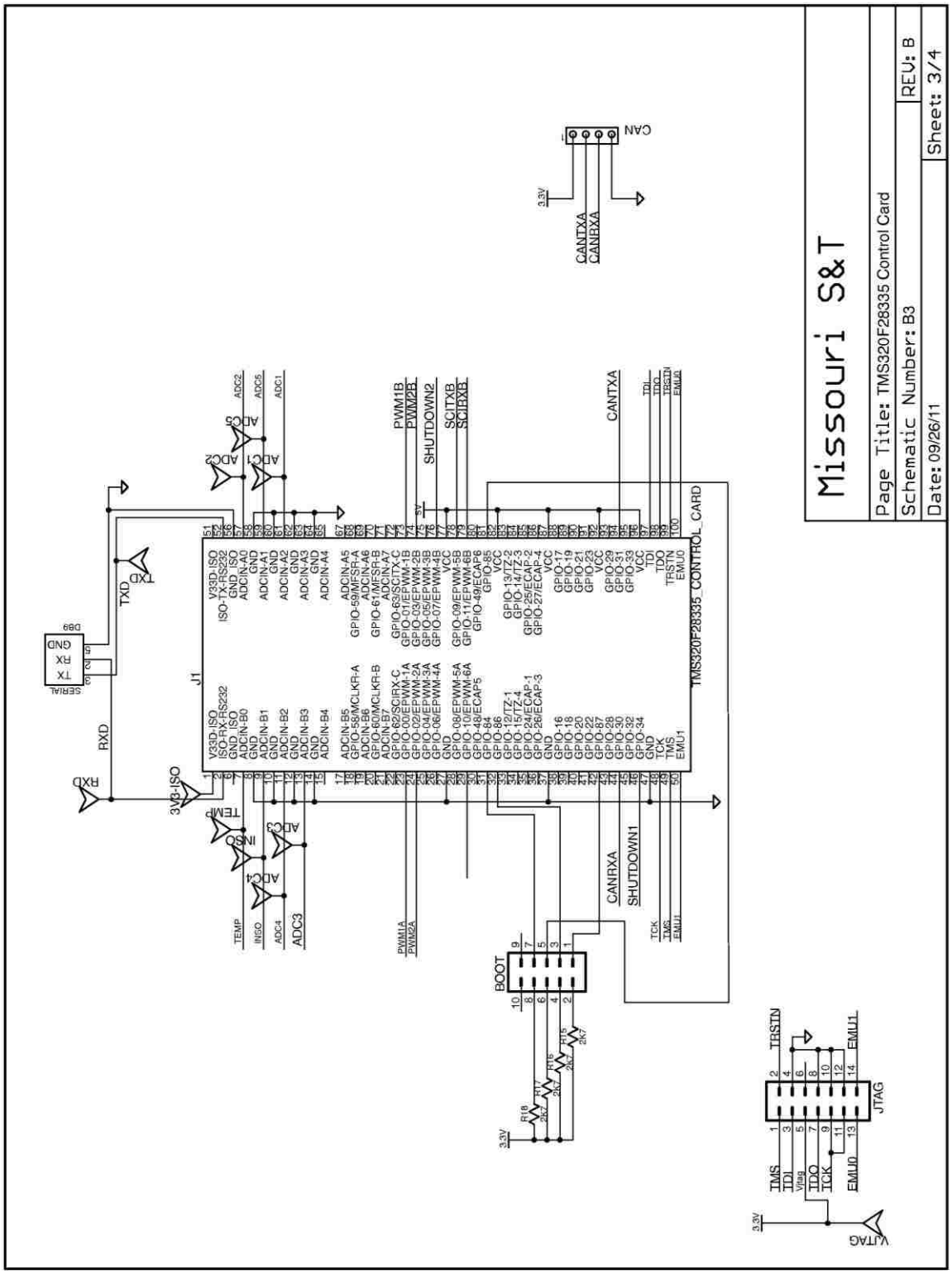

Figure A.3. PV simulator PCB schematic sheet three

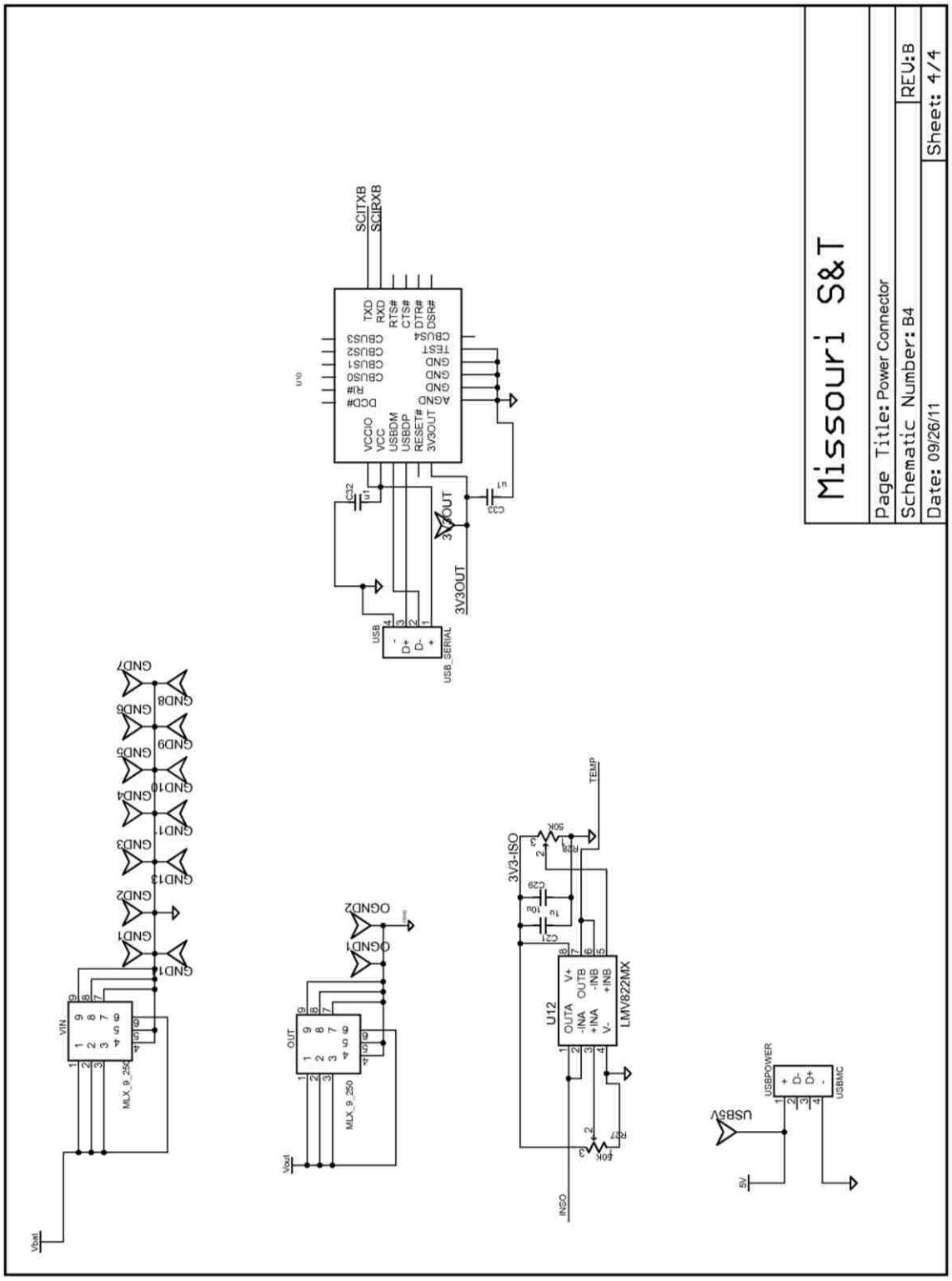

Figure A.4. PV simulator PCB schematic sheet four

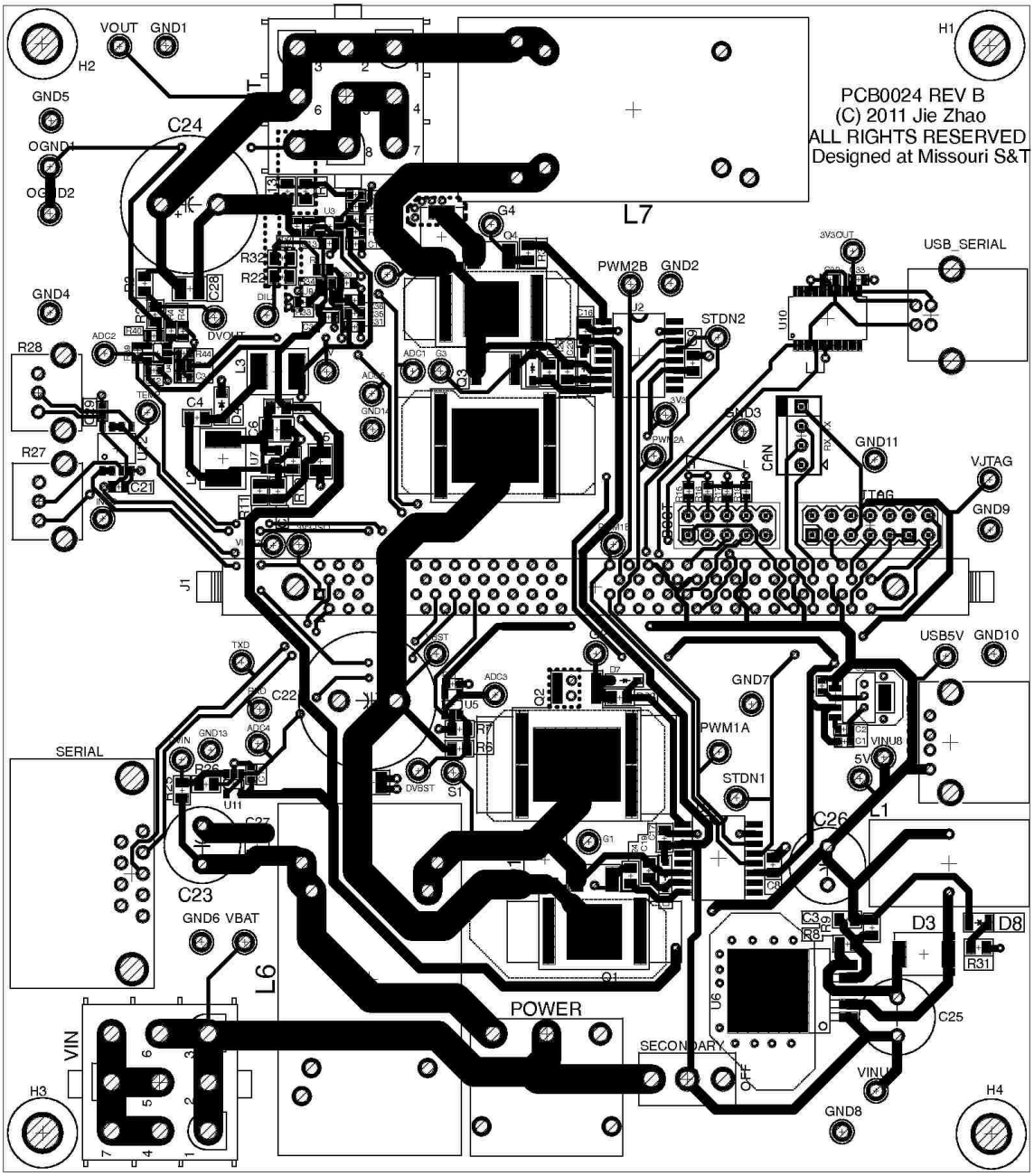

Figure A.5. Top layer of board physical layout

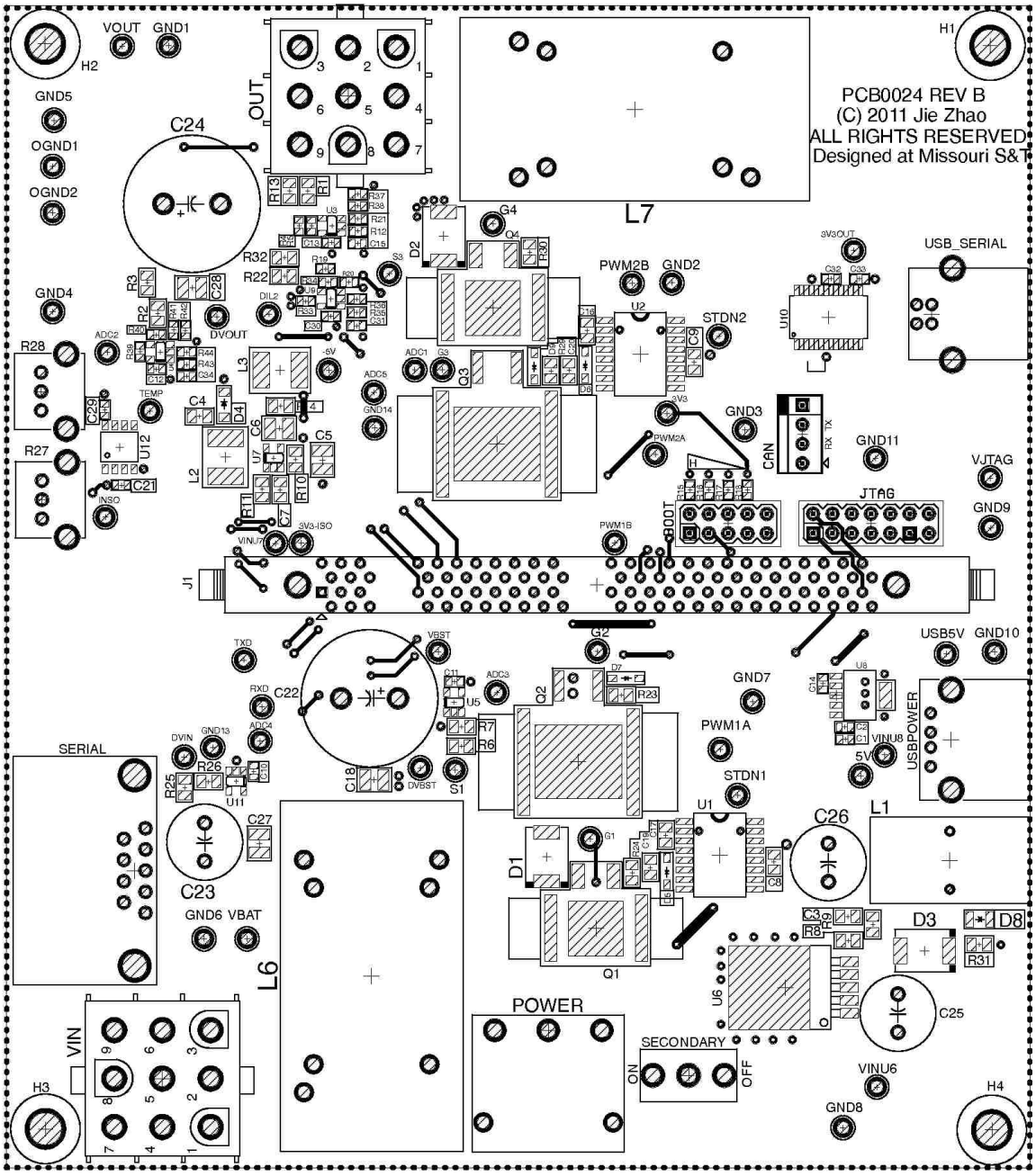

Figure A.6. Bottom layer of board physical layout

APPENDIX B.
C CODE FOR THE MICROCONTROLLER

This appendix contains the C code that is executed by the DSP TMS320F28335 from Texas Instrument. The C code shown in this appendix is very hardware specific and readers should refer to the technical manual of the DSP in order to fully understand how it functions. Note that the C code also contains the code to implement the function generator to drive the experiment setup shown in Chapter 6.

```
/*************************************************************************
 * DSP: TMS320F28335
 * (c) Jie Ang Zhao, Missouri S&T
 * Last modified 02/09/2012
 * ***********************************************************************/


#include "DSP2833x_Device.h"
#include "DSP2833x_GlobalPrototypes.h"
#include "DSP2833x_EPwm_defines.h"
#include "DSP2833x_Examples.h"

#include <math.h>                          // contains basic math
                                              functions
#include <string.h>
#include <stdio.h>                         // contains the function
                                              sprintf

//#define __FLASH                          // define if loading
                                              program to flash module

//#define __FUNCTION                       // define if the board is
                                              used as a function
                                              generator


#ifdef __FLASH
extern unsigned int  RamfuncsLoadStart;
extern unsigned int  RamfuncsLoadEnd;
extern unsigned int  RamfuncsRunStart;
#endif

// custom functions
void init_ePWMs(void);                     // setup ports for the PWMs
void Init_SCIA(void);                      // initialize SCIA
//void Gpio_select(void);
interrupt void cpu_timer0_isr(void);       // timer0 ISR
void setup_ePWM1(void);                    // initialize PWM module A
void setup_ePWM2(void);                    // initialize PWM module B


void InfoDisplay(void);                    // sent output voltage and
                                              output current through
                                              serial port


void setup_ADC_mode(void);                 // initialize ADC module
interrupt void adc_isr(void);              // ADC end of sequence ISR
interrupt void SCIA_TX_isr(void);          // SCIA transmit ISR
interrupt void SCIA_RX_isr(void);          // SCIA receive interrupt ISR
interrupt void timer0_isr(void);           // timer0 ISR
void PWMSD(unsigned int);                  // PWM shut down, 1= turnn
                                              off, 0= turn on

void gateDriveSD(unsigned int gd1, unsigned int gd2);
// gate drivers shut down, 1= shutdown, 0= release

void IV_generator(void);                   // generator look up table
```

```
void IV_generator_Vd(void);                // generate the IV curve
                                              using the Vd method

// global variables
/*********************************************************************
 * PV panel Parameters
 * ********************************************************************/
float f_temp=300;              // temperature in K
float f_Voc=22;                // open circuit voltage of panel in Volt
float f_Isc=5.4;               // short circuit current of panel in Amp
unsigned int n_cells=72;       // number of cells in series
const float q=1.6E-19;         // elememtary charge in C
const float k=1.38E-23;        // Boltzmann constant in J/K
float f_Rs=.3419;              // series resistance in Ohms
float f_Rsh=1152;              // shunt resistance in Ohms
float f_n=.4728;               // ideality factor of diode in PV cell
                                  model

const unsigned int n_size=1000;     // size of look up table
float f_ary_VI[2][1000]={0};
// IV curve lookup table. Voltage generates the index and the element
   values are the corresponding output current

float f_deltaI=0.1;                 // smallest current step
unsigned int n_index=0;             // index to be used in LUT
float f_Vth=.0169;                  // thermal voltage
float f_Io=7.3417E-8;               // reverse saturation current
float f_deltaVd=0.0221;             // equals (open circuit voltage +
                                       .1) /n_size

float f_Iph=5.4017;                 // photogenerated current
float f_Inso=1;                     // insolation level, 0-1
/***********************************
 * End of PV panel Parameters
 * *********************************/


float dutyCycle1 = 0.5;        // this sets the duty cycle for first
                                  stage boost converter

float dutyCycle2 = 0.45;       // this sets the initial duty cycle for
                                  the second stage buck converter
float r1k[3]={0,0,0};          // voltage loop state array
float r2k[2]={0,0};            // curretn loop state array
float r3k[2]={0,0};            // low pass filter state array
float irefn[2]={0};            // reference current state array
unsigned int IV_switch=0;      // switch of the IV curve

/************************************
 * PRD sets the frequency of the PWM */
unsigned int PRD = 375;        // 16bit, period of the PWM
// PRD = 37500 1KHz
// PRD = 7500  5KHz
// PRD = 3750  10KHz
// PRD = 1875  20KHz
// PRD = 1500  25KHz
// PRD = 750   50KHz
```

```
// PRD = 375   100KHz
//*************************

// global variables to save ADC readings
float Vout;                     // output voltage
float Vin;                      // input voltage
float Iout;                     // output current
float Vout_boost;               // first stage output voltage
float Inso[2]={0};              // insolation knob reading
float Inso_Out[2]={0};          // insolation after filter
float Temp;                     // temperature knob reading
float IL;                       // buck stage inductor current

// test variables
unsigned long count=0;
unsigned int IVCurveDone=0;
unsigned int indexOut=0;        // for SCIA-TX
unsigned int indexIn=0;         // for SCIA-RX
char outBuff[32];
char inBuff[32];
unsigned int sup=0;             // for testing
unsigned int txcount=0;
char tempChar;                  // helper character
unsigned long rxcount=0;
unsigned long tcount=0;

void main(void)
 {
      InitSysCtrl();            // Basic Sys Init, 150MHz SYSCLKOUT

      #ifdef __FLASH
      memcpy(&RamfuncsRunStart, &RamfuncsLoadStart, &RamfuncsLoadEnd -
&RamfuncsLoadStart);
      InitFlash();              // initialize flash memory module,
                                   pipline enabled, minimum waitstate
      #endif

      Gpio_select();
      DINT;                     // Disable all interrupts

      #ifdef __FUNCTION
      InitPieCtrl();
      InitPieVectTable();
      InitAdc();
      setup_ADC_mode();
      InitCpuTimers();
      ConfigCpuTimer(&CpuTimer0,150,100000);
      EALLOW;
      PieVectTable.TINT0=&timer0_isr;
      PieVectTable.ADCINT = &adc_isr;
      EDIS;
      PieCtrlRegs.PIEIER1.bit.INTx7=1;
      PieCtrlRegs.PIEIER1.bit.INTx6 = 1;
      IER |=0x001;
      #endif

      #ifndef __FUNCTION
```

```
InitPieCtrl();          // basic setup of PIE table; from
                           DSP2833x_PieCtrl.c
InitPieVectTable();     // default ISR's in PIE

InitAdc();              // calibrate and start the ADC module

setup_ADC_mode();       // setup ADC module
//InitCpuTimers();      // basic setup for CPU timer0, 1 and 2
//ConfigCpuTimer(&CpuTimer0,150,2000);
// timer0 triggers every 2000us
EALLOW;                 // allow access to protected
                           registers
//PieVectTable.TINT0=&timer0_isr;
// add timer0 ISR to PIE vector table
PieVectTable.ADCINT = &adc_isr;           // add adc ISR to PIE
                                             vector table
PieVectTable.SCITXINTA=&SCIA_TX_isr;
// add scia transmit interrupt to PIE vector table
PieVectTable.SCIRXINTA=&SCIA_RX_isr;
// add scia receive interrrupt to PIE vector table
EDIS;                   // disable access to protected registers
PieCtrlRegs.PIEIER1.bit.INTx6 = 1;        // enable ADC
                                             interrrupt
PieCtrlRegs.PIEIER9.bit.INTx2=1;          // SCIA-A-TX-isr
PieCtrlRegs.PIEIER9.bit.INTx1=1;          // SCIA-A-RX-isr
//PieCtrlRegs.PIEIER1.bit.INTx7 = 1;// Enable CPU Timer 0 INT
IER |=0x101;                              // enable INT1 and INT9



//IV_generator();       // initialize LUT
IV_generator_Vd();      // initialize LUT with diode voltage
                           method
IVCurveDone=1;          // shown that IV curve is done
#endif

gateDriveSD(1,1);       // shut down the two gate drivers
init_ePWMs();           // setup ports for ePWM outputs
PWMSD(1);               // turn off PWM module
setup_ePWM1();          // setup PWM1A and PWM1B
setup_ePWM2();          // setup PWM2A and PWM2B
PWMSD(0);               // turn on PWM module
DELAY_US(1000000);      // 1 second delay
gateDriveSD(0,1);       // turn on first stage gate driver for
                           boost converter

Init_SCIA();            // setup ports for SCIA

CpuTimer0Regs.TCR.bit.TSS = 0;      // start timer0

EINT;                   // enable global interrupt
ERTM;                   // enable debug events
DELAY_US(2000000);      // 2 second delay to allow for first
                           stage output to settle
gateDriveSD(0,0);       // turn on second stage buck converter
                           gate driver
while(1){
```

```
            IV_generator_Vd();      // update the LUT to reflect new
                                       Insolation condition
      };

}

void Gpio_select(void)
{
      EALLOW;
      GpioCtrlRegs.GPAMUX1.all = 0;       // GPIO15 ... GPIO0 = General
                                             Puropse I/O
      GpioCtrlRegs.GPAMUX2.all = 0;       // GPIO31 ... GPIO16 =
                                             General Purpose I/O
      GpioCtrlRegs.GPBMUX1.all = 0;       // GPIO47 ... GPIO32 =
                                             General Purpose I/O
      GpioCtrlRegs.GPBMUX2.all = 0;       // GPIO63 ... GPIO48 =
                                             General Purpose I/O
      GpioCtrlRegs.GPCMUX1.all = 0;       // GPIO79 ... GPIO64 =
                                             General Purpose I/O
      GpioCtrlRegs.GPCMUX2.all = 0;       // GPIO87 ... GPIO80 =
                                             General Purpose I/O
      GpioCtrlRegs.GPADIR.bit.GPIO7 = 1;  // setup shut down on gate
                                             driver 1
      GpioCtrlRegs.GPBDIR.bit.GPIO34 = 1; // setup shut down on gate
                                             driver 2

      EDIS;
}

interrupt void cpu_timer0_isr(void)
{
      CpuTimer0.InterruptCount++;
      EALLOW;
      SysCtrlRegs.WDKEY = 0xAA;     // service WD #2
      EDIS;
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}

void init_ePWMs(void){
      InitEPwm1Gpio();        //setup ePWM1, with pullup enabled,
                                GPIO0(1A), GPIO1(1B)
      InitEPwm2Gpio();        //setup ePWM2, with pullup enabled,
                                GPIO2(2A), GPIO3(2B)
//    InitEPwm3Gpio();        //setup ePWM3, with pullup enabled,
                                GPIO4(3A), GPIO5(3B)
//    InitEPwm4Gpio();        //setup ePWM4, with pullup enabled,
                                GPIO6(4A), GPIO7(4B)
}

void setup_ePWM1(void){

      /*********************************************
       * formula to calculate TBPRD in updown mode
       * TBPRD = 0.5*fSYSCLKOUT/(fPWM*2^CLKDIV*HSPCLKDIV),HSPCLKDIV is
       * always 2 for f28335
       * ********************************************/
      EPwm1Regs.TBCTL.bit.CLKDIV = 0x0;   // TB clock pre-scale, 3bit
```

```
        EPwm1Regs.TBPRD = PRD;
        EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;
        // set the counter to up-down mode

        // set up the duty cycle
        EPwm1Regs.CMPA.half.CMPA = (unsigned
              int)(dutyCycle1*EPwm1Regs.TBPRD);   // set duty cycle
        EPwm1Regs.AQCTLA.all = 0x0090;            //symmetrical PWM

        // deadband implementation
        // Active High Complentary suitable for power switches
           applications
        EPwm1Regs.DBCTL.bit.IN_MODE = DBA_ALL;
        //PWMA is used for both channels A and B
        EPwm1Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
        //Active High Complementary
        EPwm1Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
        //Both rising and falling edge delay

        /*********************************************
         * formula to calculate the dead band delay
         * Rising Edge Delay = T(TBCLK)*DBRED
         * Falling EDge Delay = T(TBCLK)*DBFED
         * f(TBCLK) = f(SYSCLKOUT)/(2^CLKDIV*HSPCLKDIV),HSPCLKDIV is
         * always 2 for f28335
         * *******************************************/
        EPwm1Regs.DBRED = 12;               // 160ns dead band; 10bit
        EPwm1Regs.DBFED = 12;               // 160ns dead band; 10bit
        #ifdef __FUNCTION
        EPwm1Regs.DBRED = 0;                // 0ns dead band; 10bit
        EPwm1Regs.DBFED = 0;                // 0ns dead band; 10bit
        #endif
        // end of deadband implementation

        EPwm1Regs.TBCTL.bit.SYNCOSEL = 1;   //SyncOut if CTR = 0;
}

void setup_ePWM2(void){

        /*********************************************
         * formula to calculate TBPRD in updown mode
         * TBPRD = 0.5*fSYSCLKOUT/(fPWM*2^CLKDIV*HSPCLKDIV), HSPCLKDIV is
         * always 2 for f28335
         * *******************************************/
        EPwm2Regs.TBCTL.bit.CLKDIV = 0x0;   // TB clock pre-scale, 3bit
        EPwm2Regs.TBPRD = PRD;
        EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;
        // set the counter to up-down mode

        // set up the duty cycle
        EPwm2Regs.CMPA.half.CMPA = (unsigned
              int)(dutyCycle2*EPwm1Regs.TBPRD);
        EPwm2Regs.AQCTLA.all = 0x0090;      //symmetrical PWM

        // deadband implementation
        // Active High Complentary suitable for power switches
           applications
```

```
        EPwm2Regs.DBCTL.bit.IN_MODE = DBA_ALL;    //PWMA is used for both
                                                    channels A and B
        EPwm2Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
        //Active High Complementary
        EPwm2Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
        //Both rising and falling edge delay

        /*********************************************
         * formula to calculate the dead band delay
         * Rising Edge Delay = T(TBCLK)*DBRED
         * Falling EDge Delay = T(TBCLK)*DBFED
         * f(TBCLK) = f(SYSCLKOUT)/(2^CLKDIV*HSPCLKDIV),HSPCLKDIV is
         * always 2 for f28335
         * *******************************************/
        EPwm2Regs.DBRED = 12;               // 160ns dead band; 10bit
        EPwm2Regs.DBFED = 12;               // 160ns dead band; 10bit
        #ifdef __FUNCTION
        EPwm2Regs.DBRED = 0;                // 0ns dead band; 10bit
        EPwm2Regs.DBFED = 0;                // 0ns dead band; 10bit
        #endif
        // end of deadband implementation

        EPwm2Regs.ETPS.all = 0x0100;  // Configure ADC start by ePWM2
        /*
         bit 15-14      00:     EPWMxSOCB, read-only
         bit 13-12      00:     SOCBPRD, don't care
         bit 11-10      00:     EPWMxSOCA, read-only
         bit 9-8        01:     SOCAPRD, 01 = generate SOCA on first event
         bit 7-4        0000:   reserved
         bit 3-2        00:     INTCNT, don't care
         bit 1-0        00:     INTPRD, don't care
        */

        EPwm2Regs.ETSEL.all = 0x0900;
        // Enable SOCA to ADC, change SOCASEL to select even trigger
           point
        /*
         bit 15         0:      SOCBEN, 0 = disable SOCB
         bit 14-12      000:    SOCBSEL, don't care
         bit 11         1:      SOCAEN, 1 = enable SOCA
         bit 10-8       010:    SOCASEL, 001 = SOCA on CTR = 0 event
         bit 7-4        0000:   reserved
         bit 3          0:      INTEN, 0 = disable interrupt
         bit 2-0        000:    INTSEL, don't care
        */
        EPwm2Regs.TBCTL.bit.SYNCOSEL = 0;   // SyncOut=syncin;
        EPwm2Regs.TBCTL.bit.PHSEN = 1;      // enable phase shift
        EPwm2Regs.TBPHS.half.TBPHS = PRD;   // phase shift amount
}

void init_SCIA(void){
        InitSciaGpio();         // SCIRXDA (GPIO28), SCITXDA (GPIO29);
                                   internal pullup enabled
}

void setup_ADC_mode(void){
        AdcRegs.ADCTRL1.all = 0;
```

```
        AdcRegs.ADCTRL1.bit.ACQ_PS = 7;     // ACQ_PS + 1 ADCCLK
                                               cycles of sampling window
        AdcRegs.ADCTRL1.bit.SEQ_CASC =1;          // cascaded mode
        AdcRegs.ADCTRL1.bit.CPS = 0;              // ADCCLK = FCLK/1
        AdcRegs.ADCTRL1.bit.CONT_RUN = 0;         // single run mode

        AdcRegs.ADCTRL2.all = 0;
        AdcRegs.ADCTRL2.bit.INT_ENA_SEQ1 = 1;     // enable SEQ1
                                                     interrupt
        AdcRegs.ADCTRL2.bit.EPWM_SOCA_SEQ1 =1;
        // triggered by ePWM_SOCA event
        AdcRegs.ADCTRL2.bit.INT_MOD_SEQ1 = 0;
        // interrupt every end of sequence

        /*************************************************
         * formula to calculate fFCLK = fHSPCLK/(2*ADCCLKPS)
         * fHSPCLK is default to fSYSCLKOUT/2 = 75MHz
         *************************************************/
        AdcRegs.ADCTRL3.bit.ADCCLKPS = 3;         // 12.5MHz ADC clock

        AdcRegs.ADCTRL3.bit.SMODE_SEL = 1;
        // simultaneous sampling mode
        AdcRegs.ADCMAXCONV.all = 0x0003;
        // ADCMAXCONV + 1 conversions from SEQ1

        /*************************************************
         * connect the ADC channels to the specific sequence slot
         * Ex: 1 stand for ADC1A and ADC1B for simultaneous sampling mode
         * Length of sequence limited by ADCMAXCONV
         *************************************************/
        AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0;
        AdcRegs.ADCCHSELSEQ1.bit.CONV01 = 1;
        AdcRegs.ADCCHSELSEQ1.bit.CONV02 = 2;
        AdcRegs.ADCCHSELSEQ1.bit.CONV03 = 3;
        //AdcRegs.ADCCHSELSEQ2.bit.CONV04 = 4;
        //AdcRegs.ADCCHSELSEQ2.bit.CONV05 = 5;
        //AdcRegs.ADCCHSELSEQ2.bit.CONV06 = 6;
        //AdcRegs.ADCCHSELSEQ2.bit.CONV07 = 7;

}

interrupt void adc_isr(void){
        // reads from the ADC mirror registers and convert to real
           voltage values
        Vout = AdcMirror.ADCRESULT0*0.007381;     // output voltage
        Temp = AdcMirror.ADCRESULT1*3/(float)4096;// temperature setting
        IL = AdcMirror.ADCRESULT2*0.001744;       // inductor current
        Inso[0] = AdcMirror.ADCRESULT3*3/(float)4096;// insolation level
        Iout = AdcMirror.ADCRESULT4*0.001744;     // output current
        Vin = AdcMirror.ADCRESULT5*0.004395;      // input voltage
        Vout_boost = AdcMirror.ADCRESULT7*0.008789;
        // output voltage of first stage boost converter

        //pass Inso through a digital low pass filter
        Inso_Out[0]=(Inso[0]+Inso[1]+3182*Inso_Out[1])/3184;
        Inso[1]=Inso[0];
        Inso_Out[1]=Inso_Out[0];
```

```
#ifndef __FUNCTION
/*******************************************
 * Controller implementation
 * *****************************************/
n_index=(unsigned int)floor((Vout+Iout*f_Rs)/f_deltaVd);
//generate the index

if(n_index > (n_size -1))
      n_index = n_size -1;
Iref=f_ary_VI[IV_switch%2][n_index];
// generate current reference

r1k[0]=(Iref-Iout);           // add saturation to states
if(r1k[0]>10)
      r1k[0]=10;
else if(r1k[0]<-10)
      r1k[0]=-10;

irefn[0]=7.61*r1k[0]-7.59*r1k[1]+irefn[1];

if(irefn[0]>100)              // saturate the current loop error
      irefn[0]=100;
else if(irefn[0]<-100)
      irefn[0]=-100;          // end of saturation


r2k[0]=Vout*2/Vout_boost+(irefn[0]-IL)*.46-r2k[1];
// inner current loop

if(r2k[0]>.9)
      r2k[0]=.9;
else if(r2k[0]<.1)
      r2k[0]=0.1;
dutyCycle2=r2k[0];

r1k[2]=r1k[1];                // update states
r1k[1]=r1k[0];
r2k[1]=r2k[0];
irefn[1]=irefn[0];            // end of update states

if(Iout>6)                    // limit output current
      dutyCycle2=0;

/*************************************************
 * End of Controller Implementation
 * ***********************************************/

EPwm2Regs.CMPA.half.CMPA = (unsigned
      int)(dutyCycle2*EPwm2Regs.TBPRD);   // update duty cycle
#endif
AdcRegs.ADCTRL2.bit.RST_SEQ1 = 1;         // reset SEQ1
AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1;       // clear INT
PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;   // acknowledge
                                             interrupt
}
```

```
void PWMSD(unsigned int sd){
	if(sd==0){
		EALLOW;
		SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;
		// start all the TB clocks
		EDIS;
	}
	else if(sd==1){
		EALLOW;
		SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0;
		// stop all the TB clocks
		EDIS;
	}
}

void gateDriveSD(unsigned int sd1, unsigned int sd2){
	if(sd1==0){
		GpioDataRegs.GPBCLEAR.bit.GPIO34 = 1;
		//release shutdown on GD2
	}
	else if(sd1==1){
		GpioDataRegs.GPBSET.bit.GPIO34 = 1;	//shut down gd2
	}
	if(sd2==0){
		GpioDataRegs.GPACLEAR.bit.GPIO7 = 1;
		//release shutdown on GD1
	}
	else if(sd2==1){
		GpioDataRegs.GPASET.bit.GPIO7 = 1;	//shut down gd1
	}
}
void IV_generator(){
	//unsigned int i;
	//unsigned int j;

	f_deltaI=f_Isc*f_Rsh/(n_size*(f_Rs+f_Rsh));	//calculates the
current step in the LUT
	f_Vth=k*f_temp/q;
	//calculates the thermal voltage
	f_Io=f_Isc/(exp(f_Voc/n_cells/f_Vth)-1);  //calculates the
reverse saturation current
	f_Vn=10;
	//initial guess of voltage
	f_Vf=0;

	/*for(i = 0;i < n_size;i++){
	//numerically solve for the IV curve with Newton's method
		for(j=0;j < 2000;j++){
			f_Vf=f_Vn-
(f_Io*exp((f_Vn+i*f_deltaI*f_Rs)/f_Vth)+(f_Vn+i*f_deltaI*f_Rs)/f_Rsh-
f_Io+i*f_deltaI-f_Isc)/

	(f_Io*exp((f_Vn+i*f_deltaI*f_Rs)/f_Vth)/f_Vth+1/f_Rsh);

		f_Vn=f_Vf;
		}
		f_Vn=10;
```

```
            f_ary_IV[i]=n_cells*f_Vf;
      }*/

      /*f_deltaV=f_Voc/n_size;
      for(i = 2;i <= n_size;i++){                              // use
newton's method to find the voltage VS current curve
            for(j=0;j < 2000;j++){
                  f_I2=f_I1-
(f_Io*exp((i*f_deltaV/n_cells+f_I1*f_Rs)/f_Vth)+(i*f_deltaV/n_cells+f_I
1*f_Rs)/f_Rsh-f_Io+f_I1-f_Isc)/

      (f_Io*exp((i*f_deltaV/n_cells+f_I1*f_Rs)/f_Vth)*f_Rs/f_Vth+f_Rs/f
_Rsh+1);

            f_I1=f_I2;
            }
            f_I1=2.5;
            f_ary_VI[i-2]=f_I2;
      }
      f_ary_VI[998]=0;
      f_ary_VI[999]=0;*/

      /*f_deltaR=(f_Rmax-f_Rmin)/1000;
      for(i = 1;i <= n_size;i++){
            for(j=0;j < 2000;j++){
                  f_I2=f_I1-
(f_Io*exp((f_I1*(f_deltaR*i+f_Rmin)/n_cells+f_I1*f_Rs)/f_Vth)+(f_I1*(i*
f_deltaR+f_Rmin)/n_cells+f_I1*f_Rs)/f_Rsh-f_Io+f_I1-f_Isc)/

      (f_Io*exp((f_I1*(i*f_deltaR+f_Rmin)/n_cells+f_I1*f_Rs)/f_Vth)*(i*
f_deltaR+f_Rmin+f_Rs)/f_Vth+(i*f_deltaR+f_Rmin+f_Rs)/f_Rsh+1);

            f_I1=f_I2;
            }
            f_I1=2.5;
            f_ary_RI[i-1]=f_I2;
      }*/
}

/*void init_SCIA(void){
      GpioCtrlRegs.GPAMUX2.bit.GPIO28 = 1;      // SCIRXDA
      GpioCtrlRegs.GPAMUX2.bit.GPIO29 = 1;      // SCITXDA

      SciaRegs.SCICCR.all =0x0027;        // 1 stop bit,  No loopback
                                    // ODD parity,8 char bits,
                                    // async mode, idle-line protocol
      SciaRegs.SCICTL1.all =0x0003;       // enable TX, RX, internal
SCICLK,
                                    // Disable RX ERR, SLEEP, TXWAKE

      // SYSCLOCKOUT = 150MHz; LSPCLK = 1/4 = 37.5 MHz
      // BRR = (LSPCLK / (9600 x 8)) -1
      // BRR = 487  gives 9605 Baud
      SciaRegs.SCIHBAUD    = 487 >> 8;          // Highbyte
      SciaRegs.SCILBAUD    = 487 & 0x00FF;      // Lowbyte
      SciaRegs.SCICTL1.all = 0x0023;      // Relinquish SCI from Reset
}*/
```

```
void Init_SCIA(void){
      EALLOW;
      GpioCtrlRegs.GPAMUX2.bit.GPIO28 = 1;      // SCIRXDA
      GpioCtrlRegs.GPAMUX2.bit.GPIO29 = 1;      // SCITXDA
      EDIS;

      SciaRegs.SCICCR.all = 0x0027; // 1 stop bit,  No loopback
                                    // ODD parity,8 char bits,
                                    // async mode, idle-line protocol
      SciaRegs.SCICTL1.all =0x0003; // enable TX, RX, internal SCICLK,
                                    // Disable RX ERR, SLEEP, TXWAKE

      // SYSCLOCKOUT = 150MHz; LSPCLK = 1/4 = 37.5 MHz
      // BRR = (LSPCLK / (9600 x 8)) -1
      // BRR = 487  gives 9605 Baud
      SciaRegs.SCIHBAUD    = 487 >> 8;          // Highbyte
      SciaRegs.SCILBAUD    = 487 & 0x00FF;      // Lowbyte
      SciaRegs.SCICTL2.bit.TXINTENA = 1;        // enable SCI-A Tx-ISR
      SciaRegs.SCICTL2.bit.RXBKINTENA = 1;      // enable SCI_A Rx-ISR
      SciaRegs.SCIFFTX.all=0xC060;
      // bit 15 = 1 : relinquish from Reset
      // bit 14 = 1 : Enable FIFO
      // bit 6 = 1 :  CLR TXFFINT-Flag
      // bit 5 = 1 :  enable TX FIFO match
      // bit 4-0 :  TX-ISR, if TX FIFO is 0(empty)
      SciaRegs.SCIFFCT.all = 0x0000;
      // Set FIFO transfer delay to 0
      SciaRegs.SCIFFTX.bit.TXFIFOXRESET = 1;
      // re-enable transmit fifo operation


      //SciaRegs.SCIFFRX.all = 0xE065;    // Rx interrupt level = 5

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

}

interrupt void SCIA_RX_isr(void){
      tempChar=SciaRegs.SCIRXBUF.bit.RXDT;
      if(tempChar!='\0'){
            inBuff[indexIn]=tempChar;
            indexIn++;
      }
      else {
            inBuff[indexIn]='\0';
            indexIn=0;
            InfoDisplay();
      }
      rxcount++;
      PieCtrlRegs.PIEACK.all=PIEACK_GROUP9;
}

interrupt void SCIA_TX_isr(void){
      while(indexOut < sizeof(outBuff) && SciaRegs.SCIFFTX.bit.TXFFST <
16 && outBuff[indexOut]!='\0'){
            SciaRegs.SCITXBUF=outBuff[indexOut];
```

```
            indexOut++;
      }
      if(SciaRegs.SCIFFTX.bit.TXFFST !=0){
            SciaRegs.SCIFFTX.bit.TXFFINTCLR = 1;
            // enable TX FIFO operation
            txcount++;
      }
      PieCtrlRegs.PIEACK.all=PIEACK_GROUP9;
}

void InfoDisplay(void){
      sprintf(outBuff,"%06.3f %06.3f %06.3f %06.3f\n\r", Vout, Iout,
            Temp, Inso);
      indexOut=0;
      SciaRegs.SCIFFTX.bit.TXFFINTCLR = 1;
      // enable TX FIFO operation
}
void IV_generator_Vd(void){
      unsigned int i;
            for(i=0;i<n_size;i++){
                  f_ary_VI[(IV_switch+1)%2][i]=f_Iph*f_Inso-
f_Io*(exp(f_deltaVd*i/(n_cells*f_Vth))-1)-f_deltaVd*i/f_Rsh;
            }
      IV_switch++;
}

interrupt void timer0_isr(void){
      if(f_Inso==1)
            f_Inso=.6;
      else f_Inso=1;
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}
```

APPENDIX C.
MATLAB CODE TO EXTRACT PV MODULE MODEL PARAMETERS

This appendix contains the Matlab code to implement the photovoltaic module equivalent circuit model parameter extraction method shown in Section 3.1. The solar panel parameters used is for the PV module SW-S85P from Sunwize, and the extracted parameters are shown in Table 6.2; however, users can modify the solar panel datasheet parameters to extract equivalent model parameters for any type of PV module.

```
%% Extraction of PV module parameters based on datasheet values
% PV Module SW-S85P from SunWire


close all;
clear all;

% Intitalize parameters----------------------------------------------
A=1;            % initial ideality factor
k=1.38e-23;     % Bolzman's Constant
q=1.1602e-19;   % elemental charge
Tstc=300;       % STC temperature
Io=1;           % initial dark saturation current
Iph=0;          % initial photo generated current
Rs=0;           % initial series resistance
Rsh=1000;       % initial shunt resistance
Vt=.026;        % initial thermal voltage

% solar panel datasheet parameters-----------------------------------
n=72;           % number of panels
Voc=22;         % open circuit voltage
Isc=5.4;        % short circuit current
Impp=4.9;       % max power point current
Vmpp=17.4;      % max power point voltage

% Test values of Rs and Rsh------------------------------------------
Rs_t=zeros(1,3000);
Rsh_t=zeros(1,3000);
for i=1:1:length(Rs_t)  % Initialize test values
    Rs_t(i)=.5 * power(.9995,i);
    Rsh_t(i)=5000 * power(.9995,i);
end

% helper values------------------------------------------------------
dpdvtemp=0;
dpdv=0;
didvtemp=0;
didv=0;
count=1;

% Numeric iteration method of extracting the PV panel model parameters
% with the method presented in "PV panel model based on datasheet
values"
% by D. Sera etc.
for i=1:1:length(Rs_t)
    Rs=Rs_t(i);
    for j=1:1:length(Rsh_t)
        Rsh=Rsh_t(j);
                Vt=(Vmpp+Impp*Rs-Voc)/n/log((Isc-(Vmpp+Impp*Rs-
Isc*Rs)...
                    /Rsh-Impp)/(Isc-(Voc-Isc*Rs)/Rsh));
                alpha=(Isc*Rsh-Voc+Isc*Rs)...
                    *exp((Vmpp+Impp*Rs-Voc)/n/Vt)/(n*Vt);
                dpdv=Impp+Vmpp*((-1*alpha/Rsh-
1/Rsh)/(1+alpha/Rsh+Rs/Rsh));
```

```
            if (imag(Vt)~=0 || Vt<=0)
                continue;
            elseif (dpdv==0)
                break;
            elseif (sign(dpdvtemp)*sign(dpdv)==-1)
                break;
            else
                dpdvtemp=dpdv;
            end
    end
    beta=(Isc*Rsh-Voc+Isc*Rs)*exp((Isc*Rs-Voc)/n/Vt)/(n*Vt);
    didv=(-1*beta/Rsh-1/Rsh)/(1+beta/Rsh+Rs/Rsh)+1/Rsh;
    if (didv==0)
        break;
    end
    if (sign(didvtemp)*sign(didv)==-1)
        break;
    else didvtemp=didv;
    end
end

% Outputs the finalized parameters-------------------------------------
Vt
Rs
Rsh
Io=(Isc-(Voc-Isc*Rs)/Rsh)*exp(-1*Voc/n/Vt)
Iph=Io*exp(Voc/n/Vt)+Voc/Rsh
A=Vt*q/k/Tstc
```

BIBLIOGRAPHY

[1] J. W. Kimball and P. T. Krein, "Discrete-time ripple correlation control for maximum power point tracking," *IEEE Trans. Power Electron.*, vol. 23, pp. 2353-2362, Sep. 2008.

[2] T. Esram and P. L. Chapman, "Comparison of photovoltaic array maximum power point tracking techniques," *IEEE Trans. Energy Conversion*, vol. 22, pp. 439-449, June 2007.

[3] H. Nagayoshi, "Characterization of the module/array simulator using I-V magnifier circuit of a pn photo-sensor " in *Proc. 3rd World Conf. Photovoltaic Energy Conversion*, 2003, vol. 2, pp. 2023-2026.

[4] A. Koran, K. Sano, R. Y. Kim, and J. S. Lai, "Design of a photovoltaic simulator with a novel reference signal generator and two-stage LC output filter," *IEEE Trans. Power Electron.*, vol. 25, pp. 1331-1338, May 2010.

[5] G. Vachtsevanos and K. Kalaitzakis, "A hybrid photovoltaic simulator for utility interactive studies," *IEEE Trans. Energy Conversion*, vol. EC-2, pp. 227-231, June 2987.

[6] E. Koutroulis, K. Kalaitzakis, and V. Tzitzilonis, "Development of a FPGA-based system for real-time simulation of photovoltaic modules," in *Proc. IEEE RSP'06*, 2006, pp. 200-206.

[7] H. Lee, M. J. Lee, S. N. Lee, H. C. Lee, H. K. Nam, and S. J. Park, "Development of photovoltaic simulator based on DC-DC converter," in *Proc. 31st INTEL 2009*, 2009, pp. 1-5.

[8] J. P. Lee, B. D. Min, T. J. Kim, *et al.*, "Development of a photovoltaic simulator with novel simulation method of photovoltaic characteristics," in *Proc. 31st INTEL 2009*, 2009, pp. 1-5.

[9] Y. Li, T. Lee, and F. Z. Peng, "A hybrid control strategy for photovoltaic simulator," in *Proc. IEEE APEC 2009*, 2009, pp. 899-903.

[10] M. C. D. Piazza, M. Pucci, A. Ragusa, and G. Vitale, "Analytical versus neural real-time simulation of a photovoltaic generator based on a DC-DC converter," *IEEE Trans. Industry Applications*, vol. 46, pp. 2501-2510, Dec 2010.

[11] S. Chattopadhyay and S. Das, "A digital current-mode control technique for DC-DC converters," *IEEE Trans. Power Electron.*, vol. 21, pp. 1718-1726, Nov. 2006.

[12] J. Chen, A. Prodic, R. W. Erickson, and D. Maksimovic, "Predictive digital current programmed control," *IEEE Trans. on Power Electron.*, vol. 18, pp. 411-419, Jan 2003.

[13] Y. S. Lai and C. A. Yeh, "Predictive digital-controlled converter with peak current-mode control and leading-edge modulation," *IEEE Trans. on Ind. Electron.*, vol. 56, pp. 1854-1863, June 2009.

[14] S. Bibian and H. Jin, "Time delay compensation of digital control for DC switchmode power supplies using prediction techniques," *IEEE Trans. Power Electron.*, vol. 15, pp. 835-842, Sep. 2000.

[15] S. Bibian and H. Jin, "High performance Predictive dead-beat digital controller for DC power supplies," *IEEE Trans. Power Electron.*, vol. 17, May 2002.

[16] P. Athalye, D. Maksimovic, and R. Erickson, "Variable-frequency predictive digital current mode control," *IEEE Power Electron. Letters*, vol. 2, pp. 113-116, Dec. 2004.

[17] E. Lorenzo, G. Araujo, A. Cuevas, M. Egido, J. Minano, and R. Zilles, *Solar electricity: engineering of photovoltaic system*. Seville, Spain: Progensa, 1994.

[18] D. A. Neaman, *Microelectronics Circuit Analysis and Design*, 3 ed. New York: McGraw-Hill, 2007.

[19] D. Sera, R. Teodorescu, and P. Rodriguez, "PV panel model based on datasheet values," in *Proc. IEEE ISIE 2007*, 2007, pp. 2392-2396.

[20] J. Stewart, *Calculus*, 5th ed: Thompson, 2003.

[21] R. W. Erickson and D. Maksimovic, *Fundamentals of Power Electronics*, 2 ed: Kluwer, 2004.

VITA

Jie Ang Zhao was born in Xinhui, Guangdong, China. He came to the United States with his family in 2003. He received the Bachelor of Science degree in Electrical Engineering from Missouri University of Science and Technology in 2010 and the Master of Science degree in Electrical Engineering from Missouri University of Science and Technology in 2012.

Jie Zhao has been a member of the honor society Eta Kappa Nu since 2009 and a student member of IEEE since 2011. He was a Grainger Award recipient in 2011 for his excellence in the field of power engineering.