COMPACT AND EFFICIENT POWER ELECTRONICS WITH APPLICATIONS TO SOLAR PV, AUTOMOTIVE, AND AEROSPACE SYSTEMS

BY

DEREK CHOU

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, 2017

Urbana, Illinois

Adviser:

Assistant Professor Robert C. N. Pilawa-Podgurski

ABSTRACT

Improving the power density of a power converter has many benefits for systems integration. Aspects such as thermal management, weight, conformation to mounting locations, and the footprint of the converter all become critical factors as systems continue to scale down in size. The flying-capacitor multilevel (FCML) converter topology is of interest because it has characteristics which contribute to high power density. This work presents some different applications of the FCML converter which exhibit characteristics of high power density. One such application is a converter built on a flexible polyimide substrate circuit board controlled to achieve quasi-square-wave (QSW) zero-voltage switching (ZVS). ZVS minimizes switching losses and enables high-frequency operation of the converter. The flexible nature of the board enables the converter to be integrated to non-flat surfaces such as motors, pipes, or airfoils. Another such application is the minimization of size and weight of the power stage of a maximum power point tracking system for usage in the solar photovoltaic space. The frequency multiplication effect of the FCML topology enables a $4\times$ reduction in size of this power stage. Both such applications are made possible with the usage of high device switching frequency, fast GaN transistors, and careful thermal management.

To my family, and all those who care for me

ACKNOWLEDGMENTS

I would like to thank the Pilawa Research Group and my advisor, Prof. Robert Pilawa-Podgurski, for sharing their excellence in power electronics with me. Those well-involved in the Google Little Box project, Yutian Lei, Shibin Qin, Christopher Barth, Zitao Liao, Zichao Ye, Joseph Wen-Chuen Liu, Thomas Foulkes, Intae Moon, Andrew Stillwell, and Enver Candan helped shape much of my understanding of the subject matter. The International Future Energy Challenge team of 2016-17, especially Intae Moon, Carl Haken, Erik Saathoff, and Joe Stahl, were extremely helpful in learning the practicalities and pitfalls of research projects. I would also like to thank the UC Berkeley Solar Vehicle Team (CalSol) and the Illini Solar Car Team (ISC) for putting up with the various large amounts of hardware, software, and ideas that I have generated over the years; without them I would not be anywhere near the versatile engineering student I have become. Finally, these projects were made possible in part with the funding of POETS, the center for Power Optimization of Electro-Thermal Systems.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ADC	Analog-Digital Converter
CAN	Controller Area Network
CCM	Continuous Conduction Mode
DC	Direct Current
GaN	Gallium Nitride
FCML	Flying Capacitor Multilevel
I-V	Current-Voltage
MPP	Maximum Power Point
MPPT	Maximum Power Point Tracker/Tracking
PCB	Printed Circuit Board
P&O	Perturb-and-Observe
PSPWM	Phase-shifted Pulse-width Modulation
PWM	Pulse-Width Modulation
PV	Photovoltaic
QSW	Quasi-Square-Wave
ZVS	Zero-Voltage Switching

CHAPTER 1

INTRODUCTION AND MOTIVATION

Increasing systems integration density in fields such as transportation, aerospace, and renewables has made reducing individual component sizes a priority [1, 2, 3, 4]. Processing the same amount of power in a smaller volume or weight increases the power density of a power converter in such an application, but this also creates new challenges in converter design, such as in packaging, thermal management, and other areas. Two metrics of interest, the specific or gravimetric power density (power-to-weight ratio) and the volumetric power density (power-to-volume ratio), will be referenced throughout this work. This work aims to explore applications and methods to improve both the gravimetric and volumetric power density of power converters without sacrificing other desirable qualities such as efficiency.

1.1 The Flying Capacitor Multilevel Converter

To improve the power density of power converters, it is desirable to employ capacitors, as they have superior energy density as compared to that of inductors [5, 6, 7]. Recently, it has been shown that hybrid resonant and soft-charging switched-capacitor converters are capable of achieving very high power densities [8]. A similar capacitor-based converter, the flying-capacitor multilevel (FCML) topology, has characteristics that make it desirable for building high power density converters [9]. An advantage is its effectiveness in allowing large voltage conversion ratios [10], which can also help improve power density without much impact to efficiency. Another desirable feature of the FCML converter is its ability to generate very high effective switching frequencies at the input of the filtering inductor; this reduces the overall size and weight of the passive components. The FCML topology distributes switching stress across a number of series-connected switches. Each switch withstands a fraction of what a conventional converter's switches must withstand. This has a few important implications. Lower-voltage devices may be used, which in practical cases tend to be able to switch faster, which further minimizes passive component size. Since total converter losses are distributed across multiple devices, this can help reduce hot spots and ease thermal management requirements.

1.2 Flexible Converters

The FCML topology, combined with novel substrate choices, such as polyimide, can reduce the overall weight and volume of a power converter. As one example, in motor drive applications, such converters may be integrated into the curved enclosure of the machine rather than requiring a separate housing. Similarly, conformal power converters can be integrated to surfaces such as on pipes, shafts, and wings. Moreover, the flexible nature of the PCB allows the converter to expand and contract more readily than a FR–4-based circuit board when subjected to thermal stresses. This construction method can be useful in applications involving high temperature variations, and/or weight-restricted situations.

At high switching frequencies, loss mechanisms such as overlap loss can dominate the overall power loss in a hard–switched converter. Of particular importance is the large commutation loop found in multilevel converters, as compared to conventional 2-level converters. One method to reduce the commutation loop is to use decoupling capacitors at strategic locations in the circuit [9, 11]. However, the physical size of the high voltage capacitors when creepage and clearance tolerances are considered limits how small the commutation loop can be made, introducing many restrictions on layout. To fully utilize the fast switching capabilities of GaN devices [12, 13], we present a 7-level FCML DC-DC converter operating with quasi-square-wave (QSW) zero-voltage switching (ZVS) [14]. The converter switching frequency is pushed up to 500 kHz (3 MHz effective) to reduce passive component size, and zero-voltage switching keeps the efficiency high. The converter achieves over 98.5% peak efficiency and 250 W output.

1.3 Maximum Power Point Trackers

Another practical usage of the FCML topology is in reducing the mass of the power stage in maximum power point trackers. Such a power stage must be highly efficient over a wide range of operational conditions, as the solar resource is highly variable. In conventional designs, a high efficiency power stage tends to require a relatively slower switching frequency and therefore larger and heavier passive devices. Here, we again leverage the fast switching capabilities of GaN devices [12, 13] and the frequency multiplication effect of the FCML topology to improve the power density figure of the power stage of a maximum power point tracker. Presented in this work is a 3-level FCML DC-DC converter used as the main power stage for a solar PV maximum power point tracker for battery backup applications, which achieves over 98% efficiency for a wide range of operating conditions, and a peak efficiency of 98.6% including all control, logic, and gate drive losses. The power point tracker is small and lightweight, achieving a high power density figure of 128 W/in^3 and with the ability to process 546 W.

CHAPTER 2

BACKGROUND OF FLYING-CAPACITOR MULTILEVEL CONVERTERS

This work aims to improve both the specific and volumetric power density metrics of the FCML converter and to demonstrate practical hardware prototypes in different application spaces. Power density can be improved by increasing the switching frequency of such a converter, which reduces the sizes of the passive components required for the construction of the converter. Power density can also be improved by selecting different, lighter materials used in construction of the converter. In particular, this work explores a flexible power converter which can be integrated to non-flat surfaces, and a high power density maximum power point tracker.

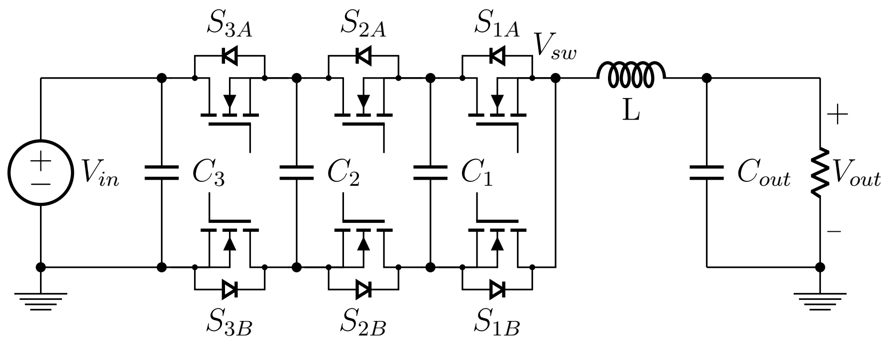

Figure 2.1: 4-level FCML schematic, buck mode.

2.1 Multilevel DC–DC Conversion

The topology implemented in this work is the flying–capacitor multilevel converter, shown in the schematic drawing of Fig. 2.1. Control of this converter is achieved through phase shifted pulse width modulation (PSPWM) and is discussed in [15]. The FCML topology may be extended to include an arbitrary number of voltage levels; however, for practicality and for the purposes of illustration, the 4-level FCML case is examined here. This analysis may be extended to different numbers of voltage levels.

The voltage conversion ratio for this converter is the same as that for a conventional buck converter; the output voltage is directly proportional to the duty ratio, D. Derivations of this relationship and others for conventional converters are covered in detail in [16] and will be presented concisely in this work.

$$V_{out} = D \cdot V_{in} \tag{2.1}$$

Through appropriate control, the FCML topology generates a high-frequency waveform at the inductor switch-node (V_{sw}, Fig. 2.1), which allows converter size and weight to be reduced dramatically through the reduction of passive component size. Furthermore, the voltage of the ith flying capacitor (C_1–C_3) is a fraction of the input voltage [15]:

$$V_{C_i} = \frac{i \cdot V_{in}}{N-1} \tag{2.2}$$

From analysis of the PSPWM technique, the effective switching frequency seen by the inductor, $f_{sw,eff}$, for a switching frequency f_{sw}, is

$$f_{sw,eff} = f_{sw} \cdot (N-1) \tag{2.3}$$

where N denotes the number of levels in the multilevel converter. Varying the number of levels in a multilevel converter is a way to balance switching frequency per switch with other important metrics such as control complexity, efficiency, and size.

Table 2.1 details the possible output voltage states for the 4-level FCML converter as seen in Fig. 2.1. Complementary switches (S_{iB}) are assumed to be in the opposite state as that of the S_{iA} switches. Here, the converter is operating in periodic steady state, so $V_{C_1} = V_{in}/3$ and $V_{C_2} = 2V_{in}/3$. Note the redundant switch combinations for generating the different voltage levels. The PSPWM technique generates waveforms that select between such output states, and balances the voltages on the flying capacitors by alternating charging and discharging paths throughout the converter [15].

Figure 2.2 details the operational waveforms of a 4-level FCML converter when $V_{in}/3 \leq V_{out} \leq 2V_{in}/3$. Because the timing of the switches in the PSPWM method causes each switch on transition to be followed by another switch off transition (but not necessarily of the same switch channel), the

Table 2.1: 4-level FCML Switching States, $V_{C_1} = V_{in}/3$ and $V_{C_2} = 2V_{in}/3$, assuming periodic steady state

S_{1A}	S_{2A}	S_{3A}	V_{sw} (math)	V_{sw}
0	0	0	0	0
1	0	0	$V_{in}/3$	$V_{in}/3$
0	1	0	$2V_{in}/3 - V_{in}/3$	$V_{in}/3$
0	0	1	$V_{in} - 2V_{in}/3$	$V_{in}/3$
1	1	0	$2V_{in}/3$	$2V_{in}/3$
0	1	1	$V_{in} - V_{in}/3$	$2V_{in}/3$
1	0	1	$V_{in} - 2V_{in}/3 + V_{in}/3$	$2V_{in}/3$
1	1	1	V_{in}	V_{in}

switch node voltage swings between two flying capacitor voltages in periodic steady state. Only one switch transition happens at any point in time. In addition to the switch timing case seen in Fig. 2.2, the 4-level FCML converter has two other operational modes that have different switch transition timings. Figures 2.3 and 2.4 show these two additional cases. Note that since the current ripple, i_L, exhibits similar behavior for all cases, the graph is not shown for the latter two figures. In Fig. 2.3, the output voltage swings between 0 and $V_{in}/3$. In Fig. 2.4, the output voltage swings between $2V_{in}/3$ and V_{in}. In all cases, the PSPWM method generates an oscillating voltage signal at the switching node. These principles can be applied to N-level FCML converters.

The conventional buck converter topology exhibits a ripple current characteristic dependent on duty ratio, maximized when D = 0.5:

$$\Delta i_{pp} = \frac{V_{in}(D \cdot (1 - D))}{L \cdot f_{sw}} \tag{2.4}$$

Similarly, the FCML converter topology exhibits a maximum current ripple characteristic at select duty ratios. The PSPWM technique gives rise to effective duty ratios, which are divided across the operating voltage range (Eqn. 2.5).

$$D_{eff} = D \cdot (N - 1) - floor(D \cdot (N - 1)) \tag{2.5}$$

The effective duty ratios are a result of the PSPWM technique; the inductor switch node sees a voltage that swings between multiples of $V_{in}/(N - 1)$

(Eqn. 2.2). The FCML converter exhibits points of minimized current ripple which occur when the output voltage is a multiple of $V_{in}/(N-1)$ and the effective duty ratio, D_{eff}, is 0%. At these points, the converter is switching between the redundant output voltage states shown in Table 2.1. When this effective duty ratio is 50% – the output voltage is exactly in between multiples of $V_{in}/(N-1)$ – the current ripple is maximized. In addition, the voltage across the inductor may only change by at most $V_{in}/(N-1)$ (Eqn. 2.6).

$$V_{eff} = \frac{V_{in}}{N-1} \tag{2.6}$$

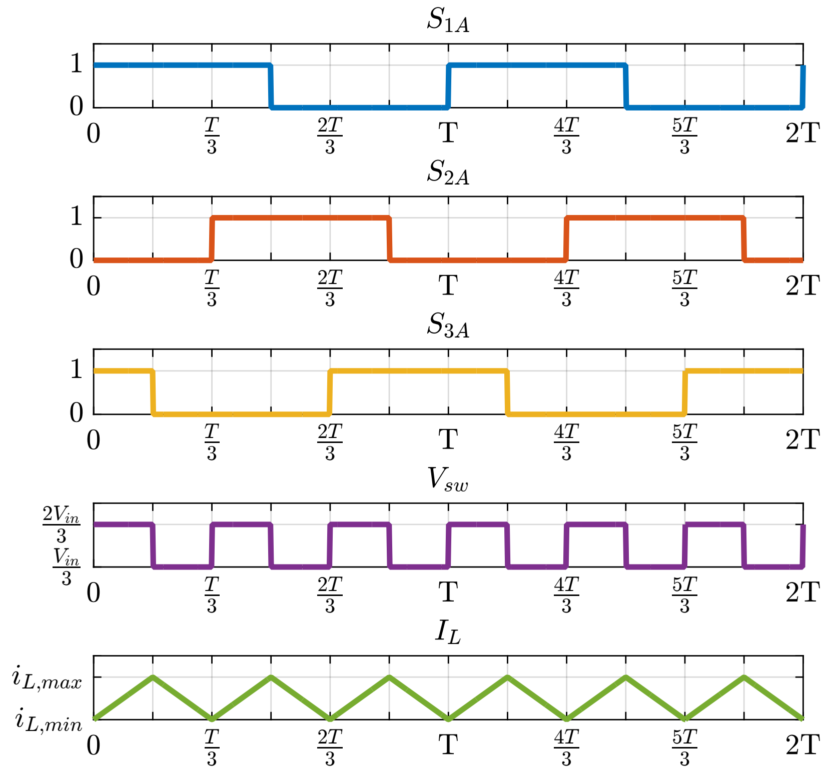

Figure 2.2: 4-level FCML operations, $V_{in}/3 \leq V_{out} \leq 2V_{in}/3$.

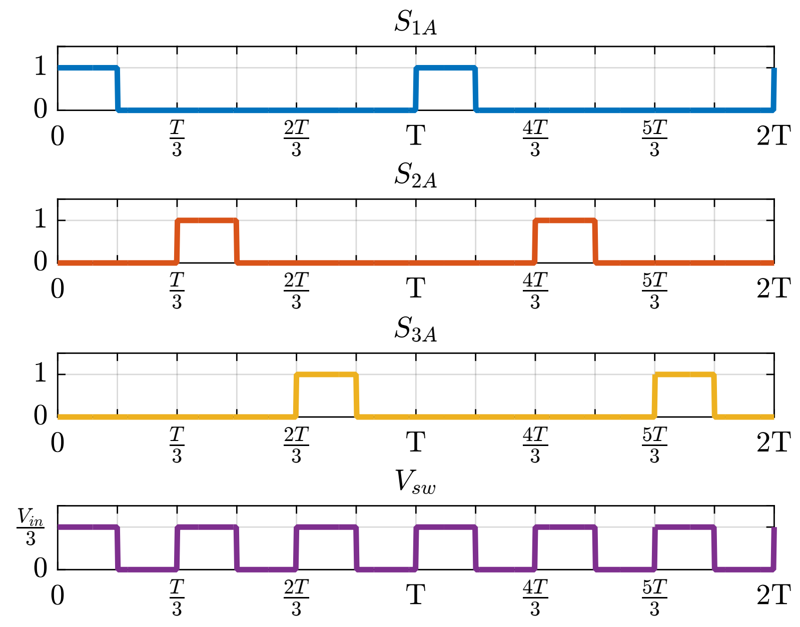

Figure 2.3: 4-level FCML operations, $0 \leq V_{out} \leq V_{in}/3$.

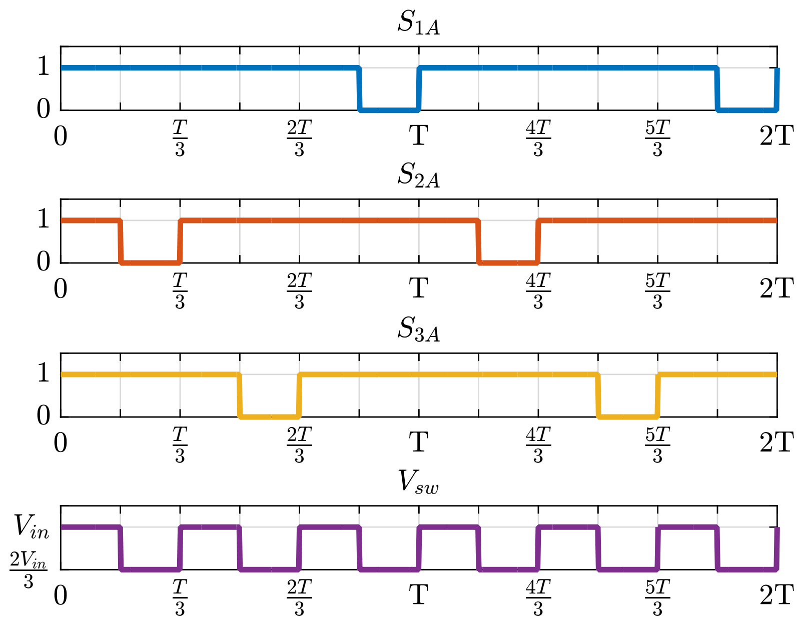

Figure 2.4: 4-level FCML operations, $2V_{in}/3 \leq V_{out} \leq Vin$.

Taking Eqn. 2.4 and replacing V_{in} with V_{eff} (Eqn. 2.6), f_{sw} with $f_{sw,eff}$ (Eqn. 2.3), and D with D_{eff} (Eqn. 2.5), the ripple current characteristic of the FCML converter is derived:

$$\Delta i_{pp,fcml} = \frac{V_{in}(D_{eff} \cdot (1 - D_{eff}))}{L \cdot f_{sw} \cdot (N-1)^2} \tag{2.7}$$

The FCML topology exhibits a lower maximum current ripple than a comparable buck converter running with the same input voltage and switching frequency. This characteristic allows a further degree of freedom when designing the filter inductor of the converter, as the inductor may now be a fraction of the size it would be in a conventional 2-level converter. Note that for $N = 2$, Eqn. 2.7 is the same as Eqn. 2.4.

Figure 2.5 illustrates the ripple current behavior comparison between a conventional buck converter (Eqn. 2.4) and FCML converters (Eqn. 2.7) where $N = 3, 4, 5$. The current ripple calculations have been normalized to the maximum current ripple characteristic of a conventional buck converter, found at $D = 0.5$.

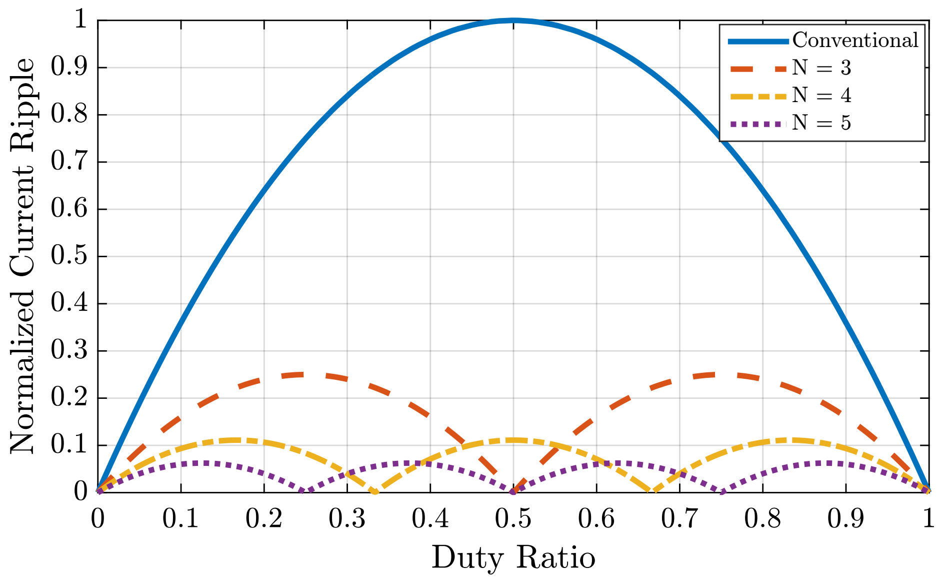

Figure 2.5: Current ripple comparison for conventional buck converter vs FCML.

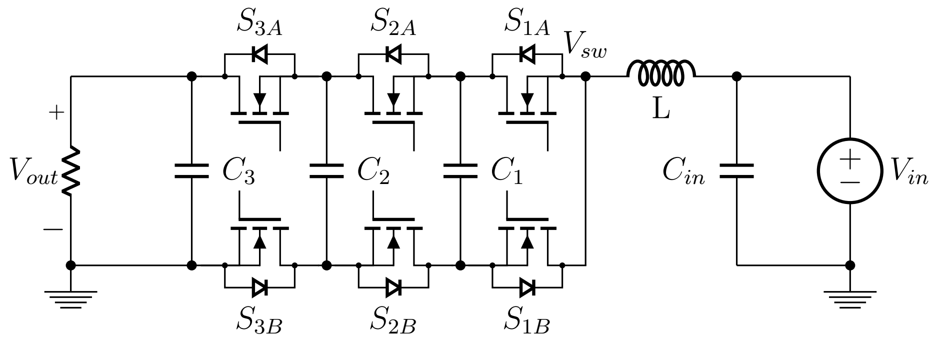

Figure 2.6: 4-level FCML schematic, boost mode.

2.2 FCML Boost Mode

Like the conventional synchronous buck converter topology, running the power stage in reverse, switching the input and the output will cause the converter to run in boost mode (Fig. 2.6), whereby the voltage conversion ratio will also be the same as a conventional boost converter:

$$V_{out} = \frac{1}{1-D} \cdot V_{in} \tag{2.8}$$

In this mode, the flying capacitors are charged to a fraction of the output voltage:

$$V_{C_i} = \frac{i \cdot V_{out}}{N-1} \tag{2.9}$$

The ripple current characteristics and effective duty ratio characteristics are the same as that of the FCML converter running in buck mode.

CHAPTER 3

QUASI-SQUARE-WAVE ZERO-VOLTAGE SWITCHING

In this chapter, the quasi-square-wave (QSW) ZVS technique [14, 17] is applied to the FCML converter, and a control scheme is proposed. ZVS is achieved by designing the inductor to allow a large current ripple, such that the output current goes negative during part of a switching cycle. The negative inductor current charges the parasitic capacitances of the active switches and enables ZVS for all switches. This method is common for a conventional buck converter [17, 18] and is extended here to the FCML topology. Since the FCML topology has a lower current ripple for all cases as compared to a conventional buck converter, either a smaller inductor or a slower switching frequency may be used to achieve the same ripple current characteristics. As QSW ZVS greatly reduces switching losses, a slower switching frequency does not make a large difference, so the size of the filter inductor is reduced in this work.

3.1 Application to the FCML Converter

Here, the QSW ZVS operation of the FCML converter is described using an arbitrary switching cell pair, as shown in Fig. 3.1. In the switching cell pair, the inductor current may flow through two possible paths, depending on the state of the preceding switch pair. These paths are denoted as i_{LA} and i_{LB}. The rightmost switching cell pair, as shown in Fig. 3.2, is a special case of Fig. 3.1 for $i = 1$. We also assume that the value of parasitic capacitor $C_{S_{iB}}$ is much smaller than the value of the flying capacitors C_i, which is true for practical designs. Therefore, during a switching cycle, discharging or charging the flying capacitors with either path, i_{LA} or i_{LB}, will not change its voltage appreciably. The resultant QSW ZVS waveforms are as noted in Fig. 3.3, and the operation of the switch pair is described in Table 3.1. The timing is denoted for the ith switch.

Some differences between the operation of this cell and a conventional buck converter are:

- The parasitic capacitor $C_{S_{iB}}$ has a voltage swing between 0 and $V_{in}/(N-1)$, as opposed to in a buck converter, where this value swings between 0 and V_{in}. This also reduces the losses in charging and discharging this capacitor fully during a full switching cycle.

- As the FCML converter has many switch pairs, there can be many switching events between the commutation of a single switch pair, depending on the overall duty ratio of the converter. Therefore, the turn-on of the A switch in a switch pair may not be directly followed by the turn-on of the B switch. However, in all cases, every switch on transition is followed by a switch off transition, and QSW ZVS is achieved at a high effective ripple frequency.

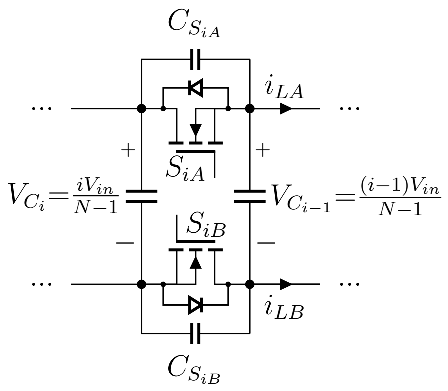
$C_{S_{iA}}$
i_{LA}
S_{iA}
$V_{C_i}=\frac{iV_{in}}{N-1}$
$V_{C_{i-1}}=\frac{(i-1)V_{in}}{N-1}$
S_{iB}
i_{LB}
$C_{S_{iB}}$

Figure 3.1: Arbitrary (ith) FCML switch pair with important elements labeled.

The proper operations of a switch pair in the FCML converter are dependent only on the commutation behavior of each cell loop, QSW ZVS can be achieved for the entire converter when combined with the PSPWM technique of [15], and the switch-node waveform is consistent for all switching events.

There are some challenges associated with the QSW technique. First, the ZVS operation necessitates a peak-to-peak inductor current ripple that is greater than twice the load current. Thus, it is essential to design the inductor such that the conduction and core losses are acceptable. Moreover,

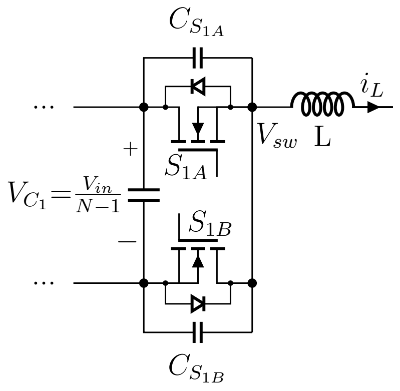

Figure 3.2: Rightmost FCML switch pair with important elements labeled.

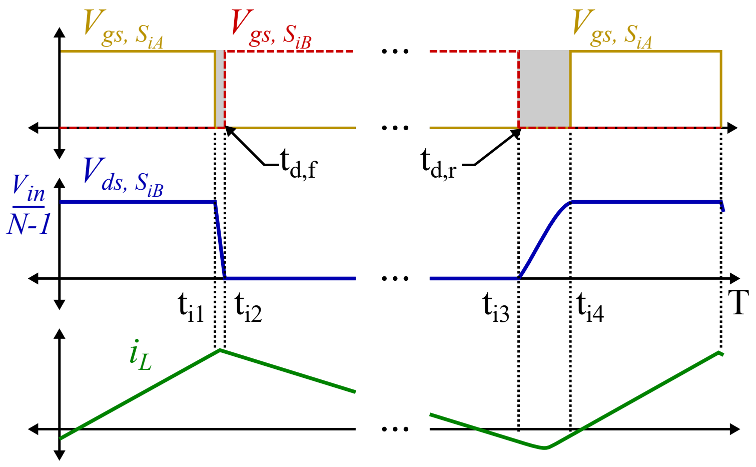

Figure 3.3: Schematic drawing of gate signals and soft-switching waveforms, at the inductor switch–node.

Table 3.1: FCML ZVS Operation for Arbitrary Switch Pair

Time	Notes
t_{i_1}	S_{iA} turns *off* with zero volts across it.
t_{i_1} to t_{i_2}	Inductor current i_L discharges parasitic capacitance $C_{S_{iB}}$.
t_{i_2}	Parasitic capacitance $C_{S_{iB}}$ has a voltage of zero across it; S_{iB} turns *on* with zero volts across it.
t_{i_2} to t_{i_3}	During this period, other switch pairs may also undergo similar ZVS transitions, depending on the output voltage of the FCML converter. A switch on transition of one switch pair is followed by either its own complementary switch off transition or by another switch pair commutation.
t_{i_3}	S_{iB} turns *off* with zero volts across it, while inductor current i_L is negative.
t_{i_3} to t_{i_4}	Inductor current i_L is negative and charges $C_{S_{iB}}$to $\frac{V_{in}}{N-1}$.
t_{i_4}	Parasitic capacitance $C_{S_{iB}}$ has a voltage of $\frac{V_{in}}{N-1}$ across it; S_{iA} turns *on* with zero volts across it.
t_{i_4} to T	Inductor current i_L becomes positive again.
T	S_{iA} turns *off* with zero volts across it, and the cycle continues.

as the inductor current changes with load, the switching frequency must continuously be varied to maintain ZVS operation through the majority of the load range. Another challenge is the duration of the deadtime, i.e., the time when both switches are off. Since the capacitance is charged by the inductor current, the duration that it takes to fully charge or discharge the capacitance depends on the value of the inductor current at the time of switching. A deadtime that is too short will force the commutation to occur before soft-switching is complete. A deadtime that is too long introduces power loss through the reverse conduction of the GaN switch. Furthermore, the falling-edge deadtime value (Fig. 3.3, $t_{d,f}$) is smaller than the rising-edge deadtime value (Fig. 3.3, $t_{d,r}$). This is because the falling edge of the switch node voltage happens at the peak of the inductor current while the rising edge happens at the valley of the inductor current. Therefore, optimal deadtime values exist for the highest efficiency.

In practice, the timing of the switches will also not be perfect, and the switches S_A and S_B will still have a minimal voltage across them when toggled [19]. Depending on the mismatch, imperfect ZVS can occur and the benefits of ZVS will be reduced. However, with proper control, the switching losses associated with switch transition times in ZVS are still reduced greatly as compared to hard-switching.

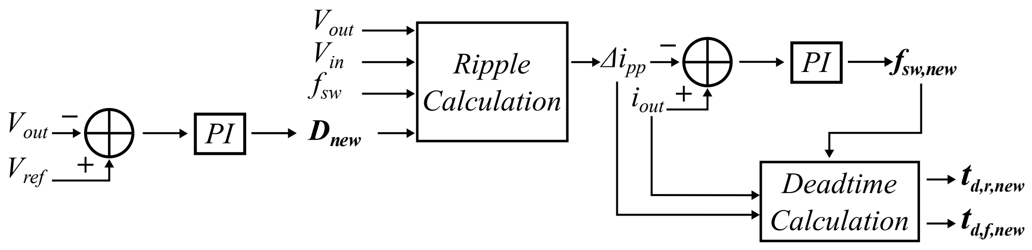

Figure 3.4: ZVS control scheme.

In this work, QSW ZVS and output voltage regulation are achieved by adjusting duty ratio, deadtime, and switching frequency. First, the controller uses the sensed output voltage, V_{out}, and adjusts the duty ratio to compensate for any error. This adjusted duty ratio, D_{new}, is then used to compute the resulting inductor current ripple, Δi_{pp}. Finally, the current ripple is compared to the sensed output current, whereby the new switching frequency, $f_{sw,new}$, and deadtime values, $t_{d,r,new}$ and $t_{d,f,new}$, are computed. Figure 3.4 provides a high-level view of the control scheme.

CHAPTER 4

EXPERIMENTAL RESULTS OF THE FLEXIBLE CONVERTER

4.1 Hardware of the Flexible Converter

The flexible converter utilizes a 7-level FCML (Fig. 4.1) as the core power conversion stage; the prototype has been constructed and tested. Fig. 4.2 shows the placement of the major components on the converter, and Table 4.1 details key components used in the construction of this prototype. The schematics and the layout of the "switching cell" for the GaN transistors can be found in Appendix A. The schematics and the layout of the converter can be found in Appendix B.

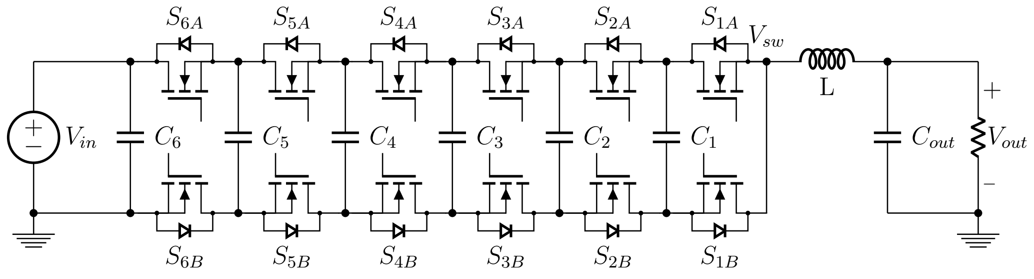

Figure 4.1: 7-level FCML schematic

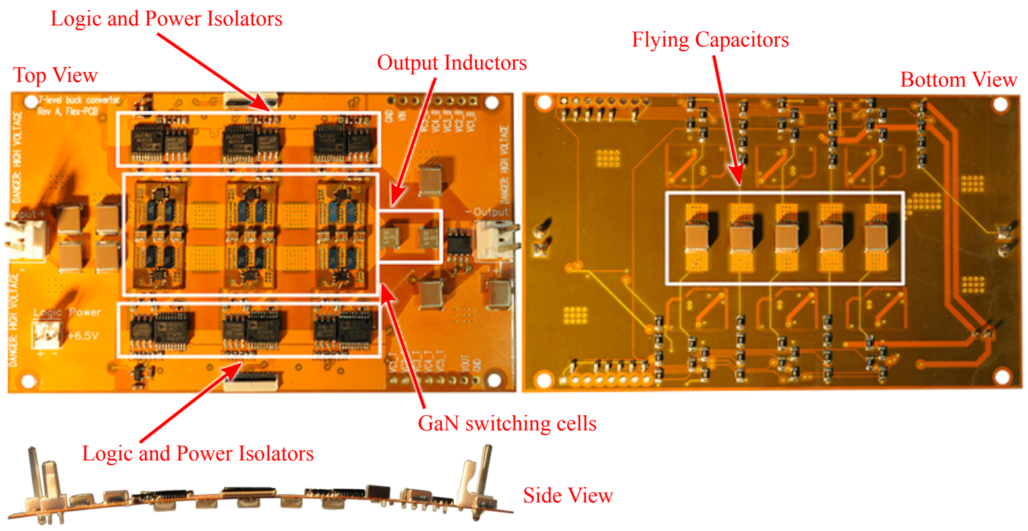

Figure 4.2: Photographs of the flexible converter presented in this work.

Table 4.1: Component Listing

Component	Part No.	Parameters
GaN FETs	EPC 2001C [20]	100 V, 7 $m\Omega$
Gate Drivers	TI LM5113 [21]	100 V, half-bridge
Flying Capacitors	TDK C5750X6S225K250KA [22]	2.2 μF
Inductor (hard switched)	Coilcraft XAL5030-332 [23]	3.3 μH $\times$ 2
Inductor (soft switched)	Coilcraft XEL4020-331 [24]	0.33 μH $\times$ 2
Digital Isolators	Silicon Labs Si8423BB-D-IS [25]	
Power Isolators	Analog Devices ADuM5010 [26]	

4.2 Results

Tests comparing conventional hard switching control against soft switching operations were run to evaluate the merit of QSW ZVS. Efficiency measurements with 500 kHz switching frequency using different inductor values (Fig. 4.3) show the effects of fixed-frequency ZVS operation versus fully hard-switched operations. A 10$\times$ smaller inductor value allows the converter to enter shallow forced CCM, making QSW ZVS operations possible. As the output load increases and the converter enters full CCM, efficiency drops slightly as the switches begin operating in a hard-switched regime again. With soft–switching control techniques [27], the switching losses in the GaN devices are greatly reduced. Figure 4.4 shows a comparison between hard-switched converter operations and variable-frequency QSW ZVS operation.

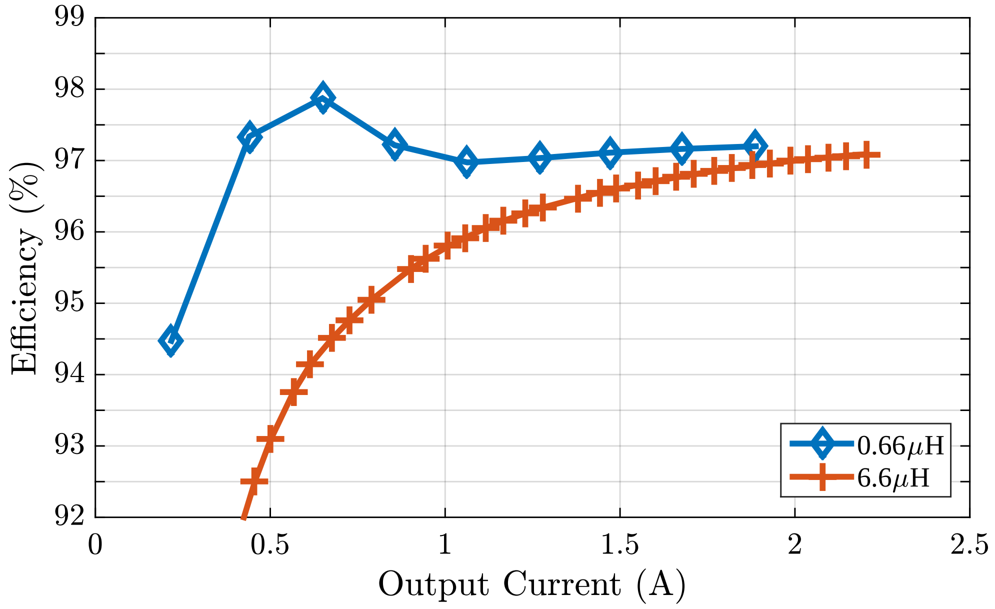

Figure 4.3: Measured efficiency, 500 kHz, $V_{in} = 200$ V, $V_{out} = 116$ V.

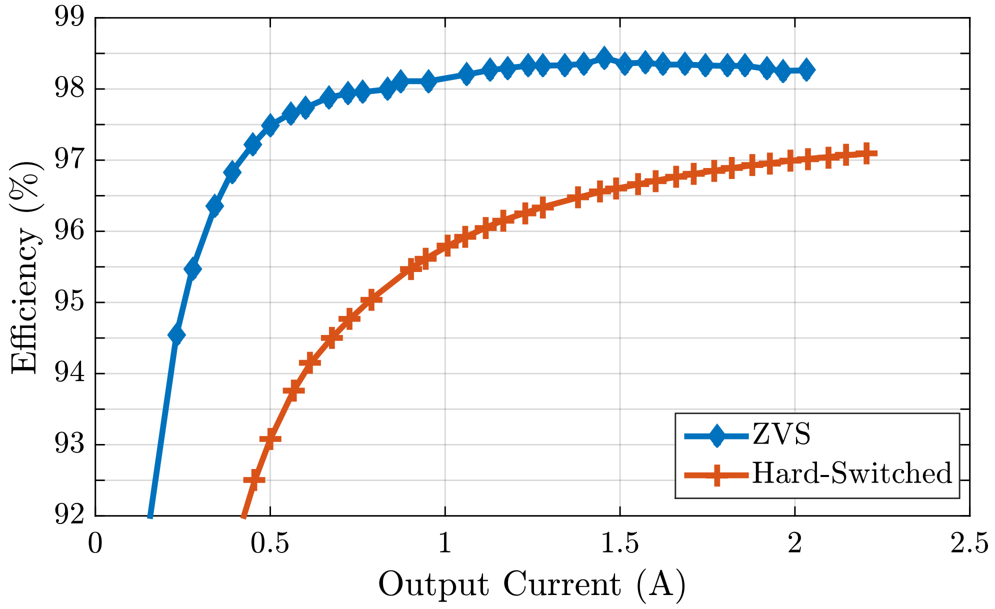

Figure 4.4: ZVS versus hard switching, measured efficiency, $V_{in} = 200$ V, $V_{out} = 116$ V.

The control signals for each switch in the 7-level FCML are shown in Fig. 4.5 for $D = 0.58$. Figure 4.6 shows representative oscilloscope traces of zero–voltage switching operations for this case at an input voltage of 200 V. These waveforms correlate with the explanation in Table 3.1. The terms i, j, k, l correspond to the "A" switch designations – for a 7-level FCML converter, these are S_{1A}–S_{6A}. In Fig. 4.6, assuming that S_{1A} turns on at $t = 0$, $i = 4$, $j = 2$, $k = 5$, and $l = 3$. The full *on–off* switch sequencing can be seen in Fig. 4.5.

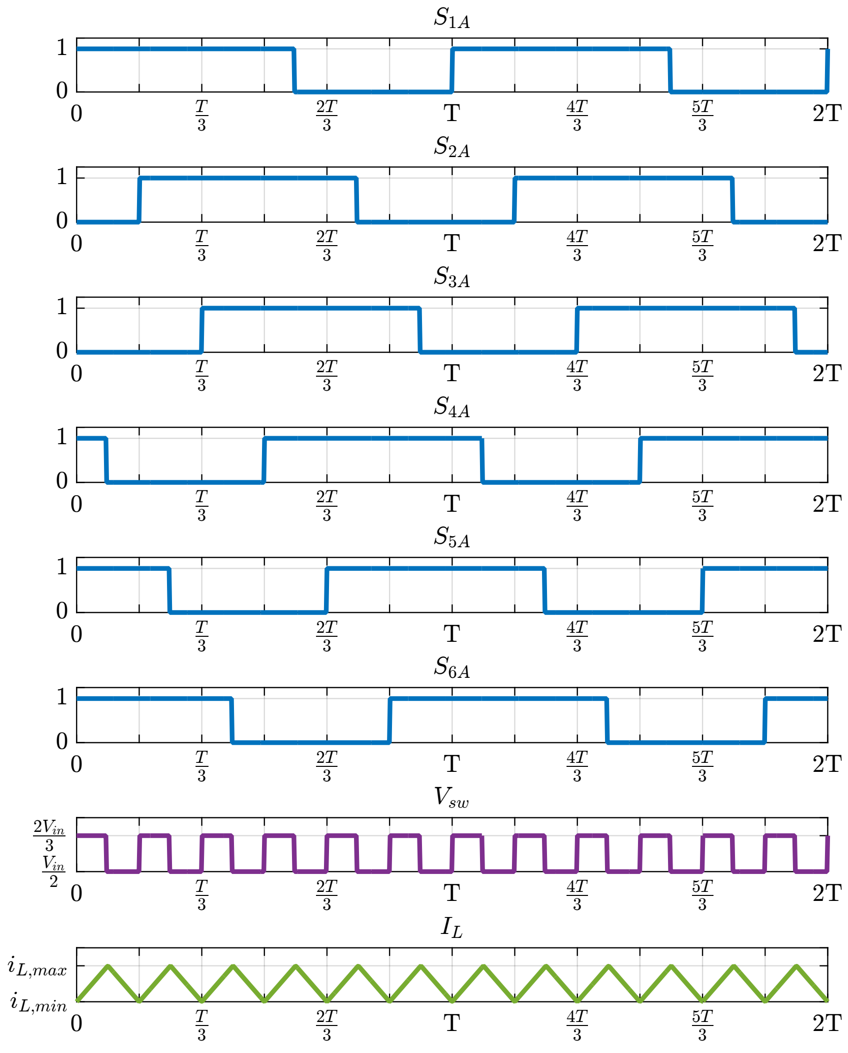

Figure 4.5: Switch control signals for 7-level FCML, $D = 0.58$

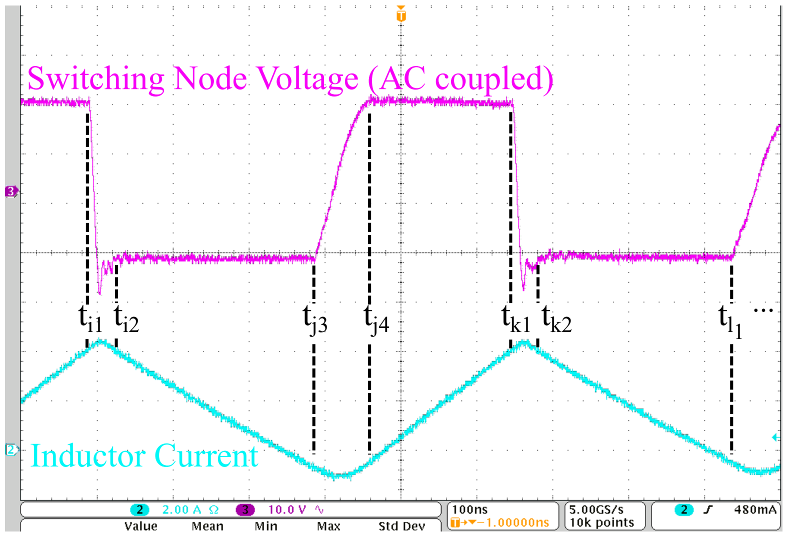

Figure 4.6: Oscilloscope traces of measured zero–voltage switching operations.

The effect of online tuning of ZVS is illustrated in Fig. 4.7. For a fixed switching frequency and deadtime, the optimal output load range is limited: efficiency is high at a single operating point but lower elsewhere. The controller adjusts the switching frequency using the control scheme of Fig. 3.4 in response to variations in the output voltage and current to maximize the efficiency over a wide load range. In this preliminary implementation, the deadtime values are fixed. Figure 4.8 is a plot of the measured efficiency of the converter operating under soft-switching conditions over a wide range of loads. This plot accounts for gate drive losses, which are significant because of the relatively high switching frequencies employed [28]. However, logic and control losses are not included in this plot, as a microcontroller evaluation board was used to generate the switching signals. ADuM5010 power isolators are used to generate the floating voltage supplies required for driving the gates of the GaN switches, which exacerbates gate drive losses due to their relatively low efficiency; this is particularly pronounced at light load. Methods such as a bootstrap gate drive [29, 30] can reduce these losses but are not explored in this work. The microcontroller code implementation for this control can be found in Appendix B.

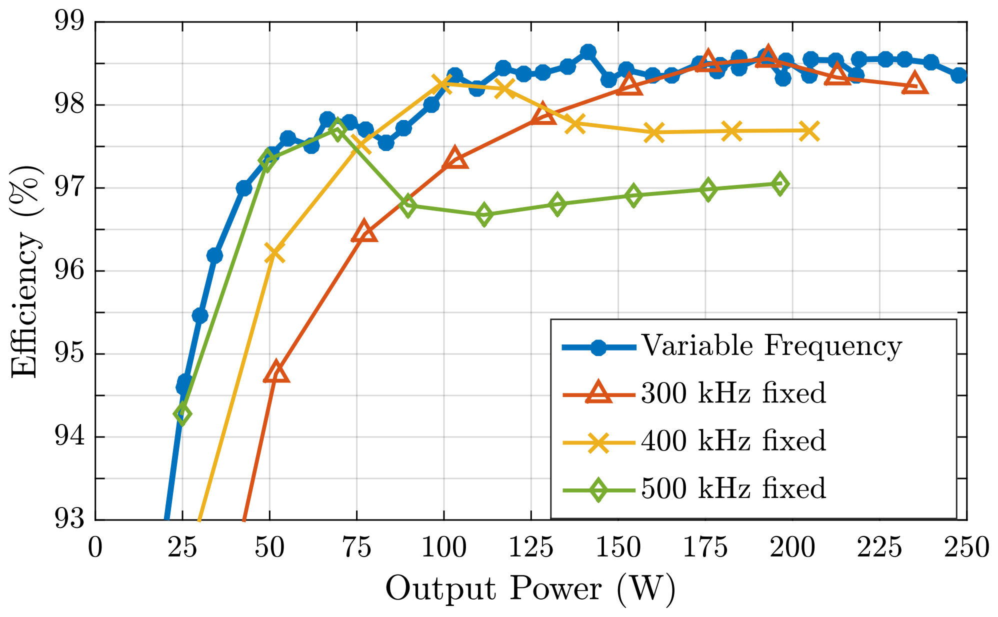

Figure 4.7: Variable frequency versus fixed frequency operation.

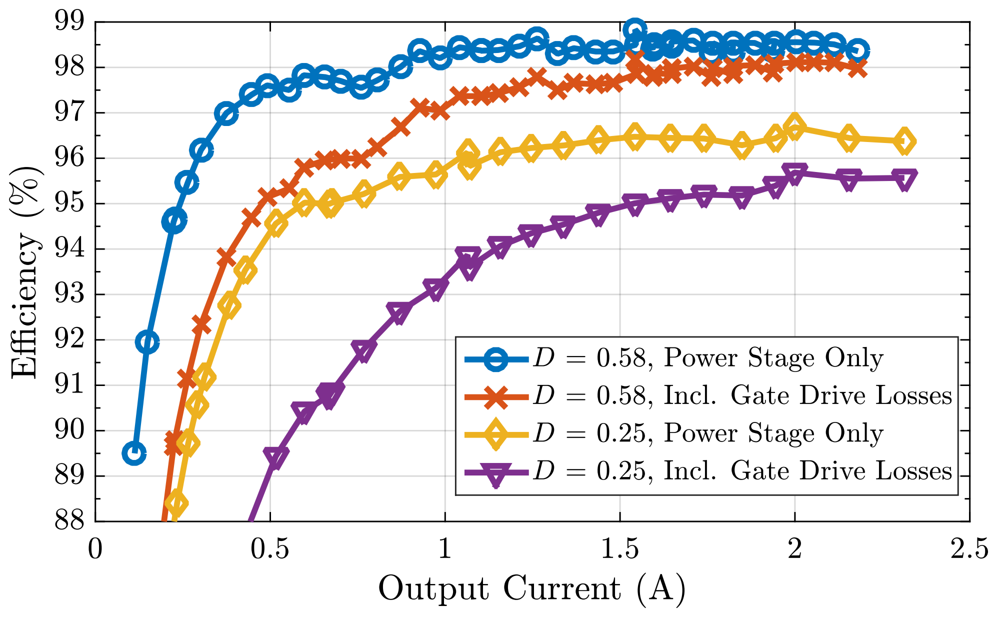

Figure 4.8: Measured efficiency, $D = 0.58$ and $D = 0.25$, $V_{in} = 200$ V; comparison with gate drive losses.

For hard-switched operations, the switching and overlap losses in the GaN FETs cause the most significant hot spots in the overall assembly; Fig. 4.9 shows the heat distribution for hard switching at $D = 0.58$ and 150 W output. As a comparison, Fig. 4.10 shows the heat distribution for soft switching at $D = 0.58$ and 150 W output. For the QSW ZVS control scheme, the AC losses in the inductors dominate.

The EPC2001C GaN FETs [20] for the prototype built in this work are switched at frequencies of up to 500 kHz, which generates a ripple frequency of up to 3 MHz at the inductor. The inductor (L) size is therefore reduced greatly as compared to a conventional converter; two 0.33 μH inductors placed in series are sufficient for filtering. Capacitors C_1 through C_6 are similarly reduced in size, and are only 2.2 μF each. Measured performance metrics are listed in Table 4.2. Two volumetric power density values are reported: one calculated using the product of the maximum dimensions in each direction of length, width, and height, and the other using the total volume of the individual components. The converter achieves both a high gravimetric power density of over 14 kW/kg and a high volumetric power density of over 109 W/in^3 (902 W/in^3, when empty space is excluded).

Table 4.2: Key Performance Specifications

Parameter	Notes	Value
Input Voltage	Tested	200 V
Output Current	Tested	2.5 A
Output Power	Tested	250 W
Switching Frequency	Per switch	200–500 kHz
Effective Frequency	At inductor	1.2–3.0 MHz
Weight	Excl. controller	17.5 g
Dimensions	Excl. connectors	4.3 in × 2.55 in × 0.209 in
		(10.9 cm × 6.48 cm × 0.53 cm)
Volume	Excl. connectors	2.29 in^3 (37.6 cm^3)
Component Volume	Excl. connectors	0.277 in^3 (4.54 cm^3)
Volumetric Power Density	Bounded by prism	109 W/in^3 (6.65 W/cm^3)
Volumetric Power Density	Excl. empty space	902 W/in^3 (55.0 W/cm^3)
Gravimetric Power Density	Excl. controller	14 kW/kg

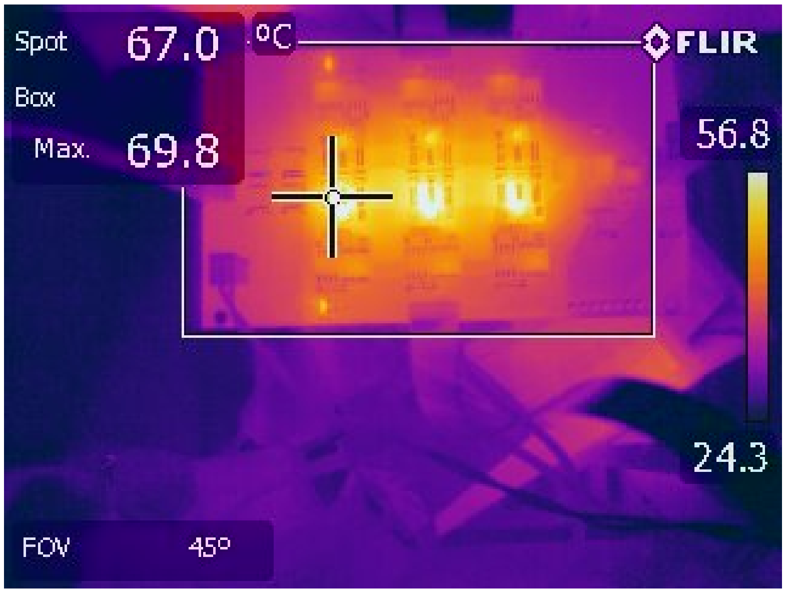

Figure 4.9: Heat distribution in converter: hard switching. $P_{out} = 150\,W$.

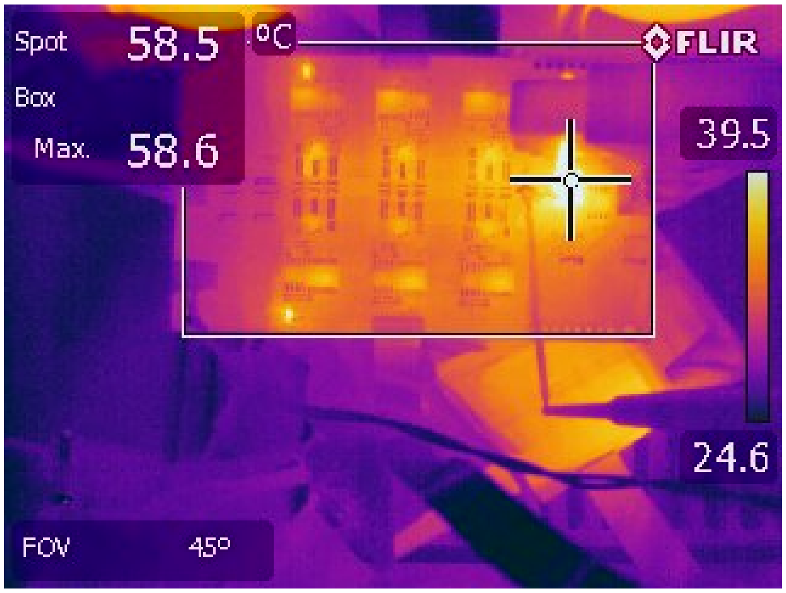

Figure 4.10: Heat distribution in converter: QSW ZVS. $P_{out} = 150\,W$.

CHAPTER 5

MAXIMUM POWER POINT TRACKING SYSTEMS

A possible application of the FCML topology is as the main power conversion stage of a maximum power point tracking system. In this case we will examine a solar-to-battery DC-DC conversion application. As seen before, the FCML topology has potential benefits in reducing the overall size and weight of the MPPT system. This work aims to produce a hardware prototype MPPT that will operate on an experimental solar vehicle.

5.1 Maximum Power Point Tracking

Solar PV cells have a typical I-V characteristic as shown in Fig. 5.1. The product of these two characteristics is the resultant power of the array. At the extremes of the I-V characteristic are the short-circuit current, I_{SC}, and the open-circuit voltage, V_{OC}. Operating at these points nets zero power. Given a particular I-V curve characteristic of a panel, there exists a point at which the power output of the PV panel is optimized, where the maximum power point is denoted as the MPP [31]. At this operational point, the product of the voltage, V_{MPP}, and the current, I_{MPP}, is maximized. Since the open-circuit voltage and short-circuit current characteristic of a PV panel are dependent on the solar resource, which is highly variable, it is advantageous to be able to track the changing maximum power point of the panel. To accomplish this, a DC-DC converter can be inserted between the solar panel and the load to change the operational characteristic of the panel. For a solar panel with multiple sub-arrays, each sub-array will have a DC-DC converter connected to it. The outputs of these converters are connected in parallel to a DC bus, which is connected to a battery. Solar sub-arrays may contain submodules of strings of solar cells; further methods of MPPT for submodule-level granularity and differential power processing between sub-arrays or submodules [32, 33, 34] can be applied to improve efficiency further, but are not explored in this work.

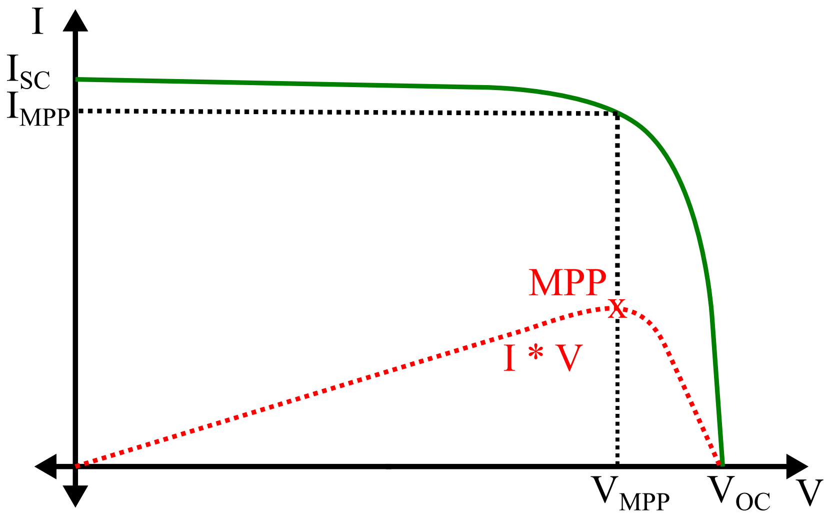

Figure 5.1: Typical solar PV IV characteristic

5.2 Flying Capacitor Multilevel Topology Application to Maximum Power Point Trackers

This work applies the FCML topology to maximum power point trackers, in order to improve the overall system power density. The FCML topology promises improvements in both volumetric and gravimetric power density figures. In a mobile application, such as in automotive uses, the weight of the converter becomes a large factor in the overall efficiency of the system, where the system considered is the vehicle [35, 36]. Reference [36] provides extensive background on the application of such a converter to a solar vehicle system, and the implications of efficiency versus weight considerations. In summary, only optimizing the efficiency of a power converter may result in an extremely efficient converter, but at the cost of increased system weight. This hurts the overall system efficiency because the extra weight of the converter must now be carried with the rest of the system, which could increase static and dynamic power draw depending on the application. There exists an optimal point at which efficiency is traded for light weight. Increasing the power density of the system by using the FCML topology should improve the overall system efficiency as the weight of the system will be reduced.

In a solar PV application, the input voltage to the MPPT can vary wildly. It is advantageous to pick a converter topology that can generate a large range of conversion ratios to account for this variation. A fast response rate to transients is also desirable. However, in many cases, large DC conversion ratios will result in large losses within the converter [37]. An aspect of the FCML converter is that it is able to produce high step-up ratios with relatively high efficiency and high power density [10]. Another aspect of the FCML converter is that it is capable of handling fast transient loading conditions [38]. The application of the FCML converter topology to an MPPT system should allow the system to accept and to respond quickly to a large range of input voltages while keeping efficiency and power density high.

5.3 Overview of System Topology

The system topology of the MPPT is detailed in Fig. 5.2. It is a DC-DC converter (in this case, the power stage is an FCML converter) surrounded by current and voltage sensors which allow a microcontroller to sense and maximize the system output power through a changing PWM signal to the DC-DC converter. The microcontroller is able to provide all of the logic and control functionality to perform MPPT and to generate the PWM signals to control the DC-DC converter [39, 40, 41]. The solar array and the battery are included in the illustration for clarity; they are not part of the MPPT.

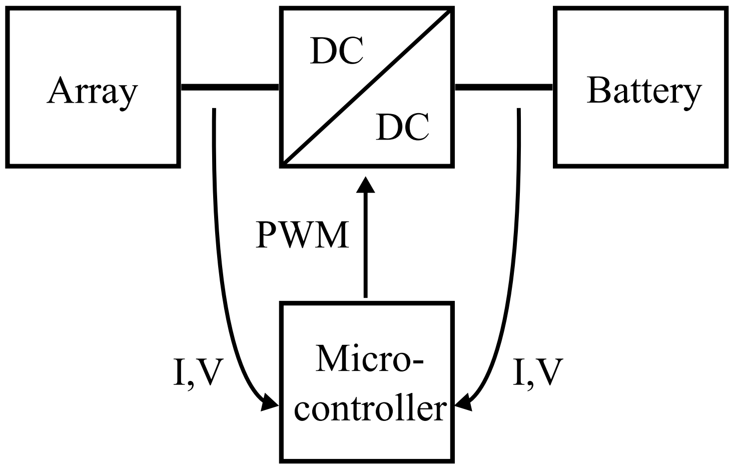

Figure 5.2: System block diagram of MPPT.

In this particular system, both the input and the output voltages are sensed, and the microcontroller uses both power values to determine the optimal duty ratio at which to run the DC-DC converter. There is a degree of redundancy in this schema, and therefore in a future revision, a set of sensors may be eliminated to reduce logic and control losses as well as to reduce code complexity.

Voltage and current sensors are implemented using simple op-amp circuits which buffer resistive elements. The voltage sensor is a simple resistive divider connected to a voltage follower. The current sensor is a differential amplifier with an offset voltage applied to allow it to sense bidirectional current. These circuits are then connected to a multiplexer, and then to an ADC, which the microcontroller queries regularly. Detailed schematics of the voltage and current sensors can be found in Appendix C.

5.4 Overview of Design Parameters

The MPPT is designed to operate as the main power input power conditioning device of a solar vehicle designed to compete in the World Solar Challenge (WSC), which is a journey of over 3000 km (over 1864 mi) through the Australian Outback in the course of about a week. The vehicle is also designed to compete in the American Solar Challenge (ASC), a journey of over 1600 mi (over 2575 km), typically across the American Midwest. Table 5.1 shows the multitude of design constraints that have been generated to assure proper operation of the electrical power systems in the vehicle.

Table 5.1: Key MPPT Design Specifications

Parameter	Notes	Value
Input Voltage Range	Solar sub-array	6–84 V
Input Current Range	Solar sub-array	0–6 A
Output Voltage Range	DC bus voltage	70–120 V
Maximum Power Processed	Solar sub-array	500 W
Waterproofing	Rain resistance	IP63+
Operating Temperature	Heat resistance	0–70°C ambient
Logic Supply Voltage		12 V
Communications Bus		CAN

The solar array will generate a maximum of 1.5 kW of total power. There will be five sub-arrays connected to five MPPTs on the vehicle in order to improve robustness against variable insolation conditions [42]. The sub-arrays which each MPPT will be attached to are not equivalent in size, so the MPPT power processing capability has been rated at 500 W to account for the mismatches between sub-arrays. The FCML converter is run in boost mode, as the solar sub-array voltages will be lower than the battery bus voltage.

Reliability is a key concern in the design of this converter system, as the serviceability of the MPPT is limited when in the field. Rain resistance and heat resistance have been quoted as design parameters because of the multitude of climate conditions that the vehicle will be traveling through. These two distinct requirements make the thermal management of the power devices on the MPPT an important consideration, as waterproofing measures can interfere with thermal management. One way to alleviate thermal management concerns is to keep the efficiency of the DC-DC power stage high. The FCML topology can be made to be extremely power dense without sacrificing power conversion efficiency. This can further improve the power density of the system as the heatsinking requirements of the power devices are reduced or even eliminated as compared to those in a conventional topology.

CHAPTER 6

EXPERIMENTAL RESULTS OF MAXIMUM POWER POINT TRACKING SYSTEMS

6.1 Hardware of the Maximum Power Point Tracker

The main power stage of the MPPT is a 3-level FCML converter, as shown in the schematic of Fig. 6.1. In normal operational conditions, both a battery source, V_{bat}, and a solar array source, V_{sol}, are present. The converter runs in boost mode from the solar array to the battery load. Table 6.1 details the relevant components used to construct the MPPT system. The system includes an on-board NXP LPC1549 [43, 44] microcontroller and all supporting circuitry for gate drive [21], current and voltage sensing, etc. Figure 6.2 shows the placement of the major components on the converter. The EPC 2032 GaN FETs [45] are switched at 120 kHz to balance switching losses and passive component size. The schematics and the layout of the "switching cell" used in the FCML DC-DC converter can be found in Appendix A. The schematics and the layout of the MPPT implementation can be found in Appendix C.

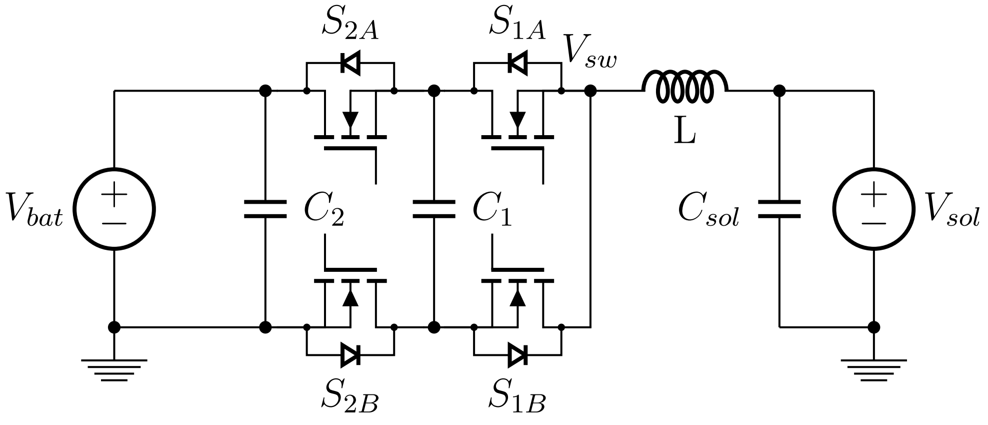

Figure 6.1: 3-level FCML schematic, connected between a solar array source and a battery load.

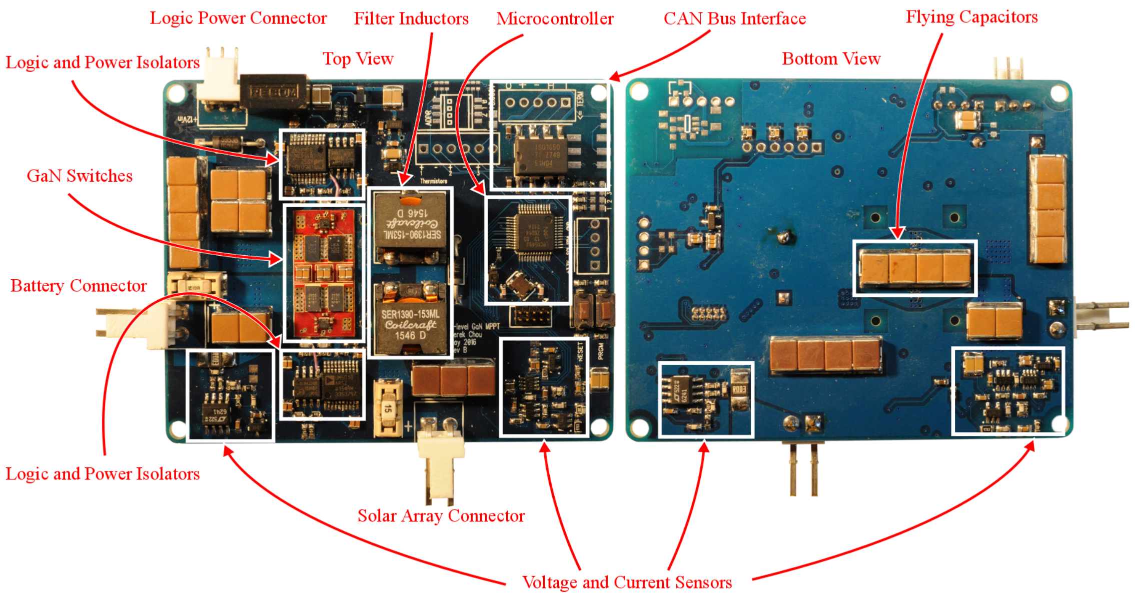

Figure 6.2: Photographs of the MPPT presented in this work.

Table 6.1: Component Listing

Component	Part No.	Parameters
GaN FETs	EPC 2032 [45]	100 V, 4 $m\Omega$
Gate Drivers	TI LM5113 [21]	100 V, half-bridge
Flying Capacitors	TDK C5750X6S225K250KA [22]	2.2 μF $\times$ 4
Inductor	Coilcraft SER1390-153 [46]	15 μH $\times$ 2
Digital Isolators	Silicon Labs Si8423BB-D-IS [25]	
Power Isolators	Analog Devices ADuM5010 [26]	
Microcontroller	NXP LPC1549 [43, 44]	72 MHz ARM Cortex-M3

6.1.1 Start-up Behavior

In a field application, this converter does not have the benefit of a soft-start circuit on either the solar end or the battery end. In addition, the FCML topology makes use of flying capacitors, which are assumed to be charged to certain specific voltages during periodic steady state. However, during startup, the flying capacitors may not be charged at all. Therefore, a precharge operation must be applied before operating the converter, to prevent overvoltage failure modes.

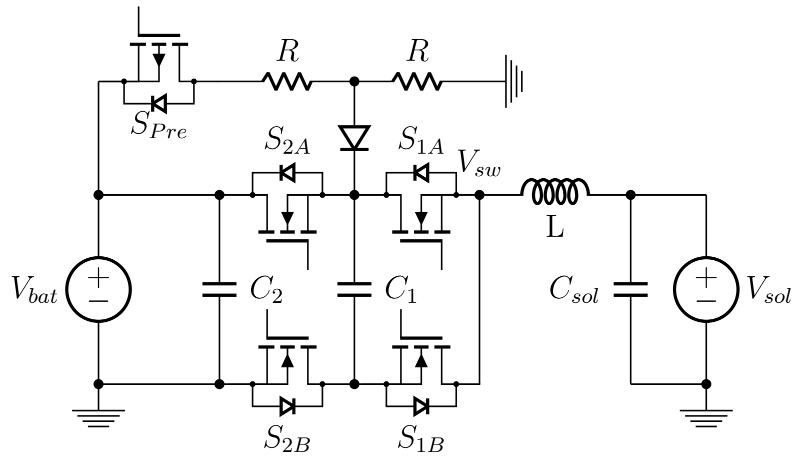

Figure 6.3: 3-level FCML schematic, connected between a solar array source and a battery load, including precharge circuitry.

To accomplish the precharge task, the circuit of Fig. 6.3 is proposed and implemented. A resistor value of 3.9 $k\Omega$ in the resistive divider allows for an RC time constant of 250 ms, as the flying capacitor C_1 in this implementation is 4 $\times$ 2.2 μ F ceramic capacitors in parallel, with a derating to about 70% of their nominal value at 65 V as shown by TDK Corporation's voltage derating graph for the ceramic capacitors [22]. In the core 3-level FCML converter, the lowest-side switch, S_{2B}, is turned *on* to allow a charging current to pass into the flying capacitor, C_1. Since the converter is designed to work with a battery pack output, this method takes advantage of the existing voltage at the output terminals of the converter. During precharge, the resistor ladder is activated, and the flying capacitor is charged to $V_{out}/2$. Then, the resistor ladder is deactivated, and the converter power stage may begin operating as normal. This method of precharging the flying capacitor requires only an extra P-MOSFET, two small power resistors rated for pulse currents of tens of milliamps at 120 V, and an N-MOSFET and proper bias resistors to drive the P-MOSFET. The circuit was successfully tested with a hot-plugged 120 V DC voltage source.

6.2 Software of the Maximum Power Point Tracker

To account for full autonomous operations in the field, a simple state machine is implemented to ensure proper start-up and tracking behavior. Figure 6.4 gives the high level state transition diagram that the converter uses to start up and track the maximum power point of a solar PV panel charging a battery load.

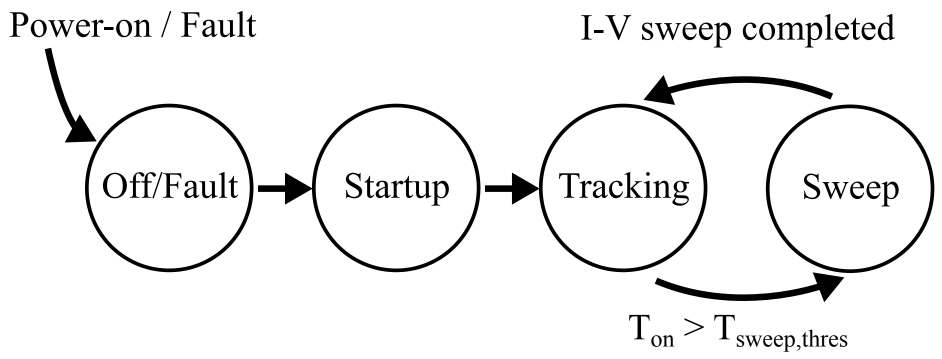

Figure 6.4: High level state transition diagram of MPPT

A description of each level state is given below:

- **Off/Fault**: The controller is either starting up or has undergone a fault. The controller attempts to determine whether the fault condition has cleared, in order to move to the startup state.

- **Startup**: A battery voltage is present and above the required threshold to precharge. The controller activates the proper circuitry to charge the flying capacitor to $V_{out}/2$.

- **Tracking**: The MPPT state. The classic perturb-and-observe (P&O) algorithm [47] is implemented here. Sensors are read on both the solar and the battery side to determine the optimal duty ratio to run the DC-DC power stage at the solar PV panel's MPP.
- **Sweep**: A modification to the P&O algorithm; after a certain threshold of time, the controller commands a full sweep of the I-V curve of the solar panel, so as to find the global maximum power point of the panel. This prevents the classic P&O algorithm from tracking only a local maximum which may not be the global maximum of power.

The microcontroller code implementation for this control can be found in Appendix C.

6.3 Results

The MPPT power stage was tested under a wide range of operating input voltage conditions to capture much of the real-world operating range of a time-varying solar array source and battery load. Further testing will be done with an emulated solar array input [48] prior to field testing with an actual solar array. Table 6.2 gives some key performance specifications of this power stage. As with the flexible converter, two power density metrics are reported: one calculated using the product of the maximum dimensions in each direction of length, width, and height, and the other using the total volume of the individual components. The converter achieves both a high gravimetric power density of over 9.75 kW/kg and a high volumetric power density of over 128 W/in^3 (574 W/in^3, when empty space is excluded).

Table 6.2: Key Performance Specifications

Parameter	Notes	Value
Input Voltage Range	Tested	6–84 V
Input Current Range	Tested	0–6.5 A
Output Voltage Range	Tested	6–120 V
Maximum Output Power	Tested	546 W
Control Losses	Tested	708 mW
Switching Frequency	Per switch	120 kHz
Effective Frequency	At inductor	240 kHz
System Weight	Excl. enclosure	56 g
Dimensions		3.25 in × 2.625 in × 0.500 in
		(8.26 cm × 6.67 cm × 1.27 cm)
Volume	Excl. enclosure	4.26 in^3 (70.0 cm^3)
Component Volume	Excl. enclosure	0.951 in^3 (15.58 cm^3)
Volumetric Power Density	Bounded by prism	128 W/in^3 (7.8 W/cm^3)
Volumetric Power Density	Excl. empty space	574 W/in^3 (35 W/cm^3)
Gravimetric Power Density	Excl. enclosure	9.75 kW/kg

Figures 6.5 and 6.6 show the operational efficiency of the MPPT power stage, including all logic, control, and gate driving losses. The converter system achieves a peak efficiency of 98.6%, and when the input voltage is higher than 60 V (boost ratio ≤ 2 for 120 V in) the efficiency of the system exceeds 98% for much of the input current range. The output voltage was fixed at 120 V to keep a relative comparison point for different input voltages. Note that for the same input current, the efficiency is higher for a higher input voltage.

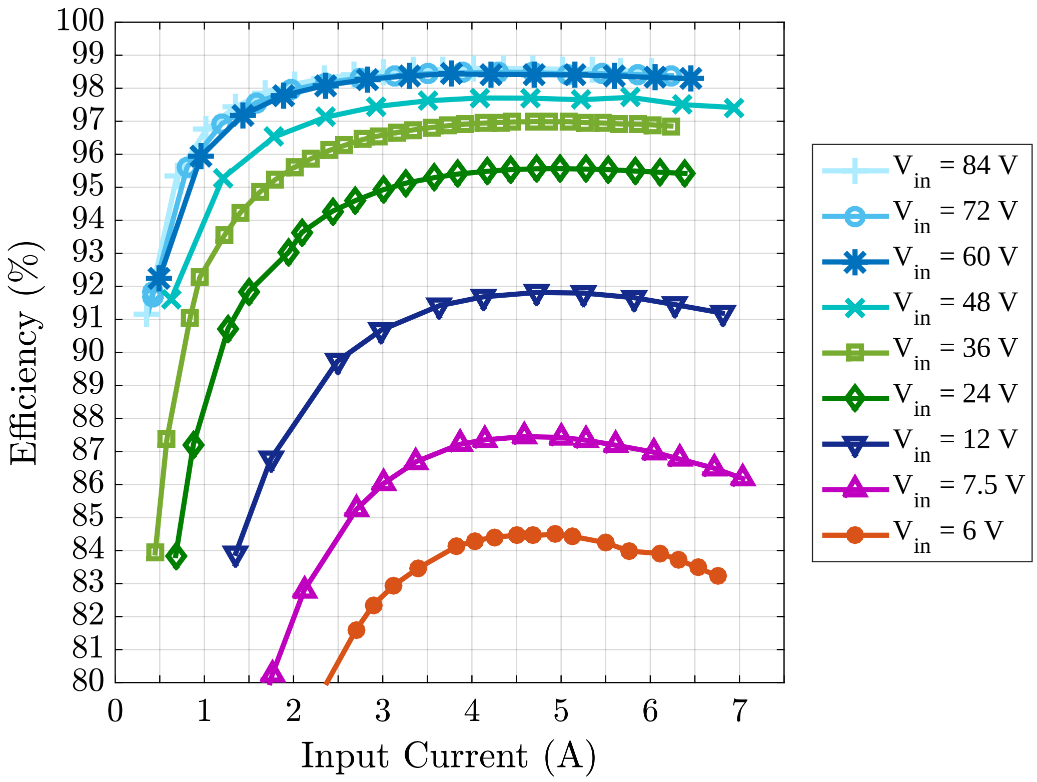

Figure 6.5: Overall system efficiency, $V_{out} = 120$ V, including all control and logic losses.

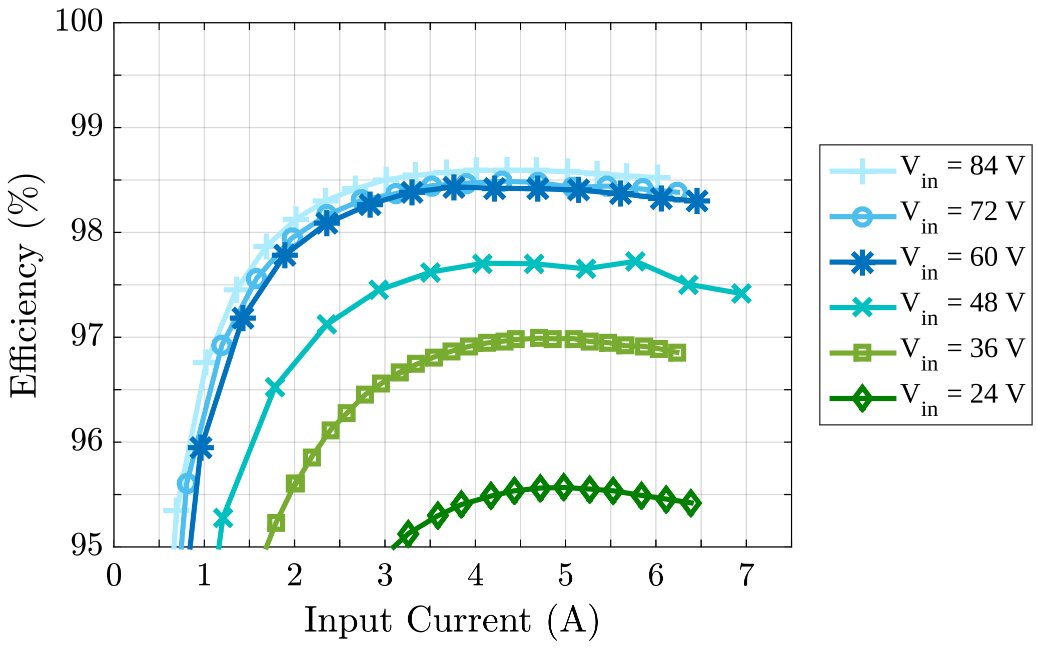

Figure 6.6: Detail of overall system efficiency, $V_{out} = 120$ V, including all control and logic losses.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

A physically flexible converter with simultaneous high power density and efficiency was developed. Its operations are enabled by a polyimide PCB substrate and a 7–level FCML converter topology. The QSW ZVS technique is applied to the FCML converter, which minimizes the switching losses in the converter. A hardware prototype has been implemented, which has demonstrated a high gravimetric power density of 14 kW/kg and a high volumetric power density of 109 W/in^3. The prototype implements automatic frequency scaling (between 200 kHz and 500 kHz per switch, 1.2 MHz and 3 MHz effective) to attain ZVS operation. A peak efficiency of 98.5% is achieved, and an efficiency higher than 98% is maintained over a wide load range.

Future work on the flexible converter involves further tuning of the control loop to achieve more ideal QSW ZVS operation. In addition, the GaN FETs could be switched at even higher frequencies, allowing still smaller inductors to be used on the converter. This may improve the efficiency of the system. A planar inductor could be implemented on this converter, further reducing its size.

A high power density power stage for a maximum power point tracker with high efficiency was developed and tested. The 3-level FCML topology and high-speed GaN switches switching at 120 kHz enable the converter to process up to 546 W, with a wide input voltage range of 6–84 V and a wide output voltage range of 6–120 V. The system achieves a peak efficiency of 98.6% and maintains an efficiency of greater than 98% over a wide load range. The system achieves a gravimetric power density of 9.75 kW/kg, and a volumetric power density of 128 W/in^3.

Future work on the maximum power point tracker involves further testing with variable input and output loads, as well as the system response to step load changes, such as a sudden disconnection of output or input sources. System efficiency may be improved by modifying converter behavior to reduce switching losses at light loads [49]. Tracking efficiency and speed can be improved by applying more advanced control methodologies than P&O [50]. Further protection circuitry may be implemented to improve startup conditions and reduce device stresses. The start-up circuitry should be fully integrated into the system in a further revision. Logic and control losses can be reduced by further optimizing the low-voltage distribution power path.

REFERENCES

[1] A. Lidow, D. Kinzer, G. Sheridan, and D. Tam, “The semiconductor roadmap for power management in the new millennium,” *Proceedings of the IEEE*, vol. 89, no. 6, pp. 803–812, Jun 2001. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=931468

[2] J. A. Rosero and J. A. Ortega and E. Aldabas and L. Romeral, “Moving towards a more electric aircraft,” *IEEE Aerospace and Electronic Systems Magazine*, vol. 22, no. 3, pp. 3–9, March 2007. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4145070

[3] K. S. Haran and K. Rajashekara, “Challenges of high power machine and drives for turbo-electric aircraft and a case study,” *IEEE Transportation Electrification Newsletter*, September 2016. [Online]. Available: http://tec.ieee.org/newsletter/september-2016/high-specific-power-machines-and-drives-for-turbo-electric-aircraft

[4] W. Su and H. Eichi and W. Zeng and M. Y. Chow, “A survey on the electrification of transportation in a smart grid environment,” *IEEE Transactions on Industrial Informatics*, vol. 8, no. 1, pp. 1–10, Feb 2012. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6051485

[5] C. B. Barth, T. Foulkes, W. H. Chung, T. Modeer, P. Assem, Y. Lei, and R. C. N. Pilawa-Podgurski, “Design and control of a GaN-based, 13-level, flying capacitor multilevel inverter,” in *2016 IEEE 17th Workshop on Control and Modeling for Power Electronics (COMPEL)*, June 2016. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7556770 pp. 1–6.

[6] S. Qin, Y. Lei, C. Barth, W.-C. Liu, and R. Pilawa-Podgurski, “Architecture and control of a high energy density buffer for power pulsation decoupling in grid-interfaced applications,” in *Control and Modeling for Power Electronics (COMPEL), 2015 IEEE 16th Workshop on*, July 2015. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7236439 pp. 1–8.

[7] S. Qin, Y. Lei, C. Barth, W.-C. Liu, and R. C. Pilawa-Podgurski, “A high-efficiency high energy density buffer architecture for power pulsation decoupling in grid-interfaced converters,” in *Energy Conversion Congress and Exposition (ECCE), 2015 IEEE*, Sept 2015. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7309682 pp. 149–157.

[8] Y. Lei, R. May, and R. Pilawa-Podgurski, “Split-phase control: Achieving complete soft-charging operation of a dickson switched-capacitor converter,” *IEEE Transactions on Power Electronics*, vol. 31, no. 1, pp. 770–782, Jan 2016. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7041205

[9] Y. Lei, C. Barth, S. Qin, W. C. Liu, I. Moon, A. Stillwell, D. Chou, T. Foulkes, Z. Ye, Z. Liao, and R. Pilawa-Podgurski, “A 2 kW, single-phase, 7-level flying capacitor multilevel inverter with an active energy buffer,” *IEEE Transactions on Power Electronics*, in press, 2017. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7811253

[10] Z. Liao, Y. Lei, and R. C. N. Pilawa-Podgurski, “A GaN-based flying-capacitor multilevel boost converter for high step-up conversion,” in *2016 IEEE Energy Conversion Congress and Exposition (ECCE)*, Sept 2016. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7854684 pp. 1–7.

[11] T. Modeer, C. Barth, N. Pallo, W. H. Chung, T. Foulkes, and R. Pilawa-Podgurski, “Design of a GaN-based, 9-level flying capacitor multilevel inverter with low inductance layout,” in *IEEE Applied Power Electronics Conference and Exposition (APEC)*, 2017.

[12] A. Lidow, J. Strydom, R. Strittmatter, and C. Zhou, “GaN: A Reliable Future in Power Conversion: Dramatic performance improvements at a lower cost,” *IEEE Power Electronics Magazine*, vol. 2, no. 1, pp. 20–26, March 2015. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7054053

[13] D. Reusch, J. Strydom, and A. Lidow, “A new family of gan transistors for highly efficient high frequency dc-dc converters,” in *2015 IEEE Applied Power Electronics Conference and Exposition (APEC)*, March 2015. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7104619 pp. 1979–1985.

[14] V. Vorperian, "Quasi-Square-Wave converters: topologies and analysis," *IEEE Transactions on Power Electronics*, vol. 3, no. 2, pp. 183–191, 1988. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4348

[15] T. Meynard and H. Foch, "Multi-level conversion: high voltage choppers and voltage-source inverters," in *Power Electronics Specialists Conference, 1992. PESC '92 Record., 23rd Annual IEEE*, Jun 1992. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=254717 pp. 397–403 vol.1.

[16] R. Erickson and D. Maksimovic, *Fundamentals of Power Electronics.* Kluwer Academics, 2000.

[17] W. A. Tabisz and F. C. Lee, "Principles of quasi- and multi-resonant power conversion techniques," in *1991., IEEE International Sympoisum on Circuits and Systems*, Jun 1991. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=176546 pp. 1053–1056 vol.2.

[18] D. M. Sable, F. C. Lee, and B. H. Cho, "A zero-voltage-switching bidirectional battery charger/discharger for the NASA EOS satellite," in *Applied Power Electronics Conference and Exposition, 1992. APEC '92. Conference Proceedings 1992., Seventh Annual*, Feb 1992. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=228354 pp. 614–621.

[19] M. Kasper, R. M. Burkart, and J. W. Kolar, "ZVS of power MOSFETs Revisited," *IEEE Transactions on Power Electronics*, vol. 31, no. 12, pp. 8063–8067, Dec. 2016. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7482851

[20] Efficient Power Conversion Corporation, "EPC2001C - Enhancement Mode Power Transistor Datasheet." [Online]. Available: http://epc-co.com/epc/Portals/0/epc/documents/datasheets/EPC2001C_datasheet.pdf

[21] Texas Instruments, "LM5113 100 V 1.2-A / 5-A, Half-Bridge Gate Driver for Enhancement Mode GaN FETs." [Online]. Available: http://www.ti.com/lit/ds/symlink/lm5113.pdf

[22] TDK, "C5750X6S2W225K250KA." [Online]. Available: https://product.tdk.com/en/search/capacitor/ceramic/mlcc/info?part_no=C5750X6S2W225K250KA

[23] Coilcraft, "Shielded Power Inductors–XAL50xx." [Online]. Available: http://www.coilcraft.com/pdfs/xal50xx.pdf

[24] Coilcraft, “Shielded Power Inductor XEL4020.” [Online]. Available: http://www.coilcraft.com/pdfs/xel4020.pdf

[25] Silicon Labs, “Low-Power, Single and Dual-Channel Digital Isolators.” [Online]. Available: https://www.silabs.com/documents/public/data-sheets/si841x-2x-datasheet.pdf

[26] Analog Devices, “Integrated DC-to-DC Converter.” [Online]. Available: http://www.analog.com/media/en/technical-documentation/data-sheets/ADuM5010.pdf

[27] C. P. Henze, H. C. Martin, and D. W. Parsley, “Zero-voltage switching in high frequency power converters using pulse width modulation,” in *Proceedings 1988. Third Annual IEEE Applied Power Electronics Conf and Exposition APEC ’88*, 1988. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10548 pp. 33–40.

[28] A. F. Goldberg and J. G. Kassakian, “The application of power MOSFETs at 10MHz,” in *1985 IEEE Power Electronics Specialists Conference*, June 1985. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7070933 pp. 91–100.

[29] Z. Ye and R. C. N. Pilawa-Podgurski, “A power supply circuit for gate driver of GaN-based flying capacitor multi-level converters,” in *2016 IEEE 4th Workshop on Wide Bandgap Power Devices and Applications (WiPDA)*, Nov 2016. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7799909 pp. 53–58.

[30] Z. Ye, W. C. Liu, P. Shenoy, and R. C. N. Pilawa-Podgurski, “Design and implementation of a low-cost and compact floating gate drive power circuit for GaN-based flying capacitor multi-level converters,” in *IEEE Applied Power Electronics Conference and Exposition (APEC)*, 2017.

[31] T. Esram and P. Chapman, “Comparison of photovoltaic array maximum power point tracking techniques,” *IEEE Transaction on Energy Conversion*, vol. 22, no. 2, pp. 439–449, 2007. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4207429

[32] R. C. N. Pilawa-Podgurski and D. J. Perreault, “Submodule integrated distributed maximum power point tracking for solar photovoltaic applications,” *IEEE Transactions on Power Electronics*, vol. 28, no. 6, pp. 2957–2967, June 2013. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6339082

[33] S. Qin, S. T. Cady, A. D. Domnguez-Garca, and R. C. N. Pilawa-Podgurski, “A distributed approach to maximum power point tracking for photovoltaic submodule differential power processing,” *IEEE Transactions on Power Electronics*, vol. 30, no. 4, pp. 2024–2040, April 2015. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6832650

[34] M. Schuck and R. C. N. Pilawa-Podgurski, “Ripple minimization through harmonic elimination in asymmetric interleaved multiphase DC–DC converters,” *IEEE Transactions on Power Electronics*, vol. 30, no. 12, pp. 7202–7214, Dec 2015. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7012116

[35] D. J. Patterson, “Electrical system design for a solar powered vehicle,” in *21st Annual IEEE Conference on Power Electronics Specialists*, 1990. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=471984 pp. 618–622.

[36] C. R. Sullivan and M. J. Powers, “A high-efficiency maximum power point tracker for photovoltaic arrays in a solar-powered race vehicle,” in *Power Electronics Specialists Conference, 1993. PESC ’93 Record., 24th Annual IEEE*, 1993. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=471984 pp. 574–580.

[37] D. Maksimovic and S. Cuk, “Switching converters with wide DC conversion range,” *IEEE Transactions on Power Electronics*, vol. 6, no. 1, pp. 151–157, Jan 1991. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=65013

[38] V. Yousefzadeh, E. Alarcon, and D. Maksimovic, “Three-level buck converter for envelope tracking applications,” *IEEE Transactions on Power Electronics*, vol. 21, no. 2, pp. 549–552, Mar. 2006. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1603688

[39] A. Prodic, D. Maksimovic, and R. W. Erickson, “Design and implementation of a digital pwm controller for a high-frequency switching dc-dc power converter,” in *Industrial Electronics Society, 2001. IECON ’01. The 27th Annual Conference of the IEEE*, vol. 2, November 2001. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=975878 pp. 893–898 vol.2.

[40] E. Koutroulis, K. Kalaitzakis, and N. C. Voulgaris, “Development of a microcontroller-based, photovoltaic maximum power point tracking control system,” *IEEE Transactions on Power Electronics*, vol. 16, no. 1, pp. 46–54, Jan 2001. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=903988

[41] D. Maksimovic, R. Zane, and R. Erickson, “Impact of digital control in power electronics,” in *2004 Proceedings of the 16th International Symposium on Power Semiconductor Devices and ICs*, May 2004. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1332844 pp. 13–22.

[42] J. Traube, F. Lu, D. Maksimovic, J. Mossoba, M. Kromer, P. Faill, S. Katz, B. Borowy, S. Nichols, and L. Casey, “Mitigation of solar irradiance intermittency in photovoltaic power systems with integrated electric-vehicle charging functionality,” *IEEE Transactions on Power Electronics*, vol. 28, no. 6, pp. 3058–3067, June 2013. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6303922

[43] NXP Semiconductors, “LPC15xx.” [Online]. Available: http://www.nxp.com/documents/data_sheet/LPC15XX.pdf

[44] NXP Semiconductors, “LPC15xx User manual.” [Online]. Available: http://www.nxp.com/documents/user_manual/UM10736.pdf

[45] Efficient Power Conversion Corporation, “EPC2032 - Enhancement Mode Power Transistor Datasheet.” [Online]. Available: http://epc-co.com/epc/Portals/0/epc/documents/datasheets/EPC2032_datasheet.pdf

[46] Coilcraft, “Shielded Power Inductors – SER1390.” [Online]. Available: http://www.coilcraft.com/pdfs/ser1390.pdf

[47] O. Wasynczuk, “Dynamic behavior of a class of photovoltaic power systems,” *IEEE Power Engineering Review*, vol. PER-3, no. 9, pp. 36–37, Sept 1983. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5519293

[48] S. Qin, K. A. Kim, and R. C. N. Pilawa-Podgurski, “Laboratory emulation of a photovoltaic module for controllable insolation and realistic dynamic performance,” in *2013 IEEE Power and Energy Conference at Illinois (PECI)*, Feb 2013. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6506029 pp. 23–29.

[49] B. Arbetter, R. Erickson, and D. Maksimovic, "DC-DC converter design for battery-operated systems," in *Power Electronics Specialists Conference, 1995. PESC '95 Record., 26th Annual IEEE*, vol. 1, Jun 1995. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=474799 pp. 103–109 vol.1.

[50] C. Barth and R. C. N. Pilawa-Podgurski, "Dithering digital ripple correlation control for photovoltaic maximum power point tracking," *IEEE Transactions on Power Electronics*, vol. 30, no. 8, pp. 4548–4559, Aug 2015. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6898827

APPENDIX A

GAN SWITCHING CELL

Both the flexible power converter and the MPPT make use of a “switching cell” which contains GaN devices and the supporting gate drive circuitry required to properly drive them. The schematics and layout files are included here for reference. The EPC 2001C device used in the flexible converter has a different footprint than the EPC 2032 device used in the MPPT, and therefore there are two different layouts used for the switching cell. However, the electrical schematics are the same, with slight adjustments to gate resistor $R1$ and $R2$ values. For the EPC 2001C device, $R1 = R2 = 22\ \Omega$, and for the EPC 2032 device, $R1 = R2 = 10\ \Omega$. For other devices, such as the EPC 2033, the gate resistor value is adjusted as needed.

A.1 Schematics

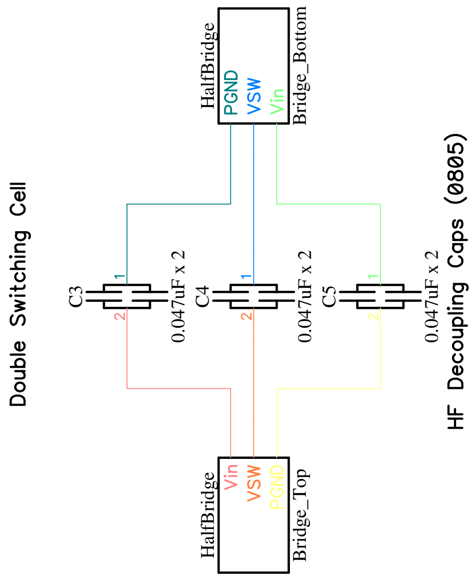

Figure A.1: Toplevel schematics for GaN switching cell.

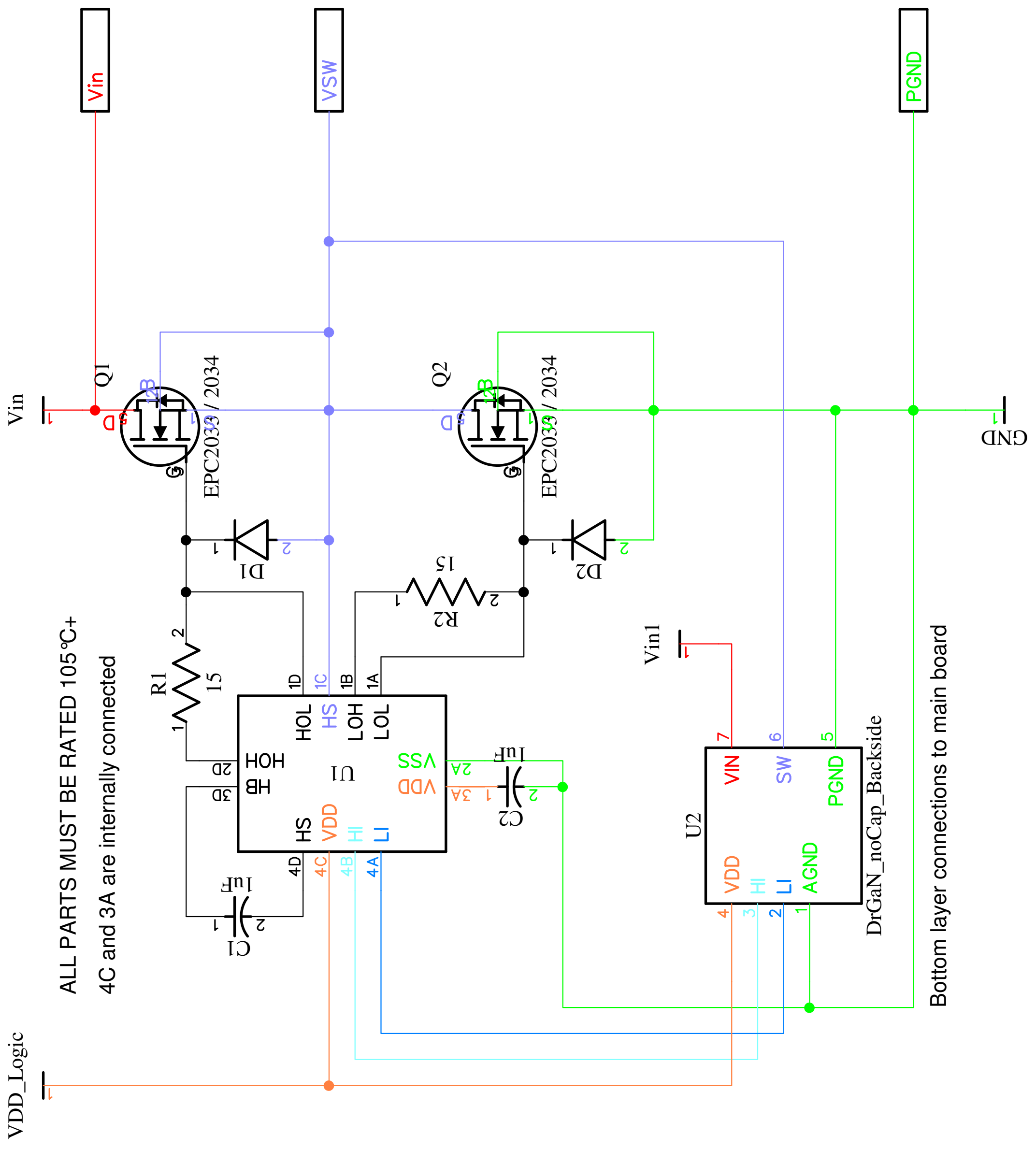

Figure A.2: Half-bridge driver and GaN device schematics for GaN switching cell. The layout is slightly different for the EPC 2001C device versus the EPC 2032 device, but the schematic connections of the GaN devices are the same.

A.2 PCB Layout – EPC 2001C

Note: The EPC 2001C switching cell does not have a bottom silkscreen layer.

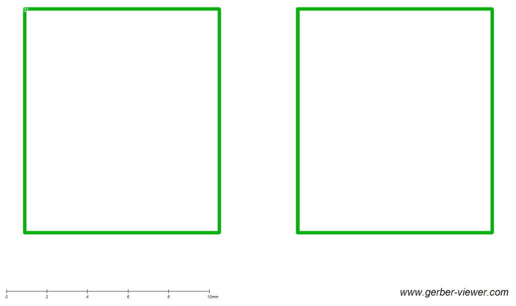

Figure A.3: Top silkscreen layer of the EPC 2001C switching cell.

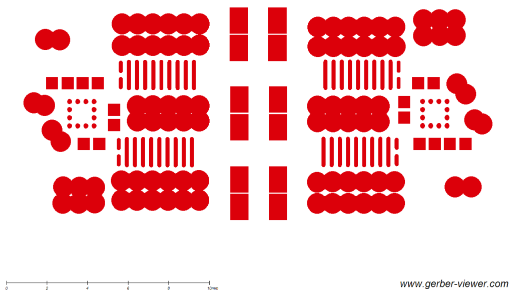

Figure A.4: Top soldermask layer of the EPC 2001C switching cell.

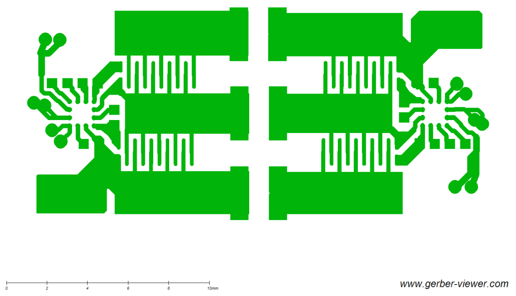

Figure A.5: Top copper layer of the EPC 2001C switching cell.

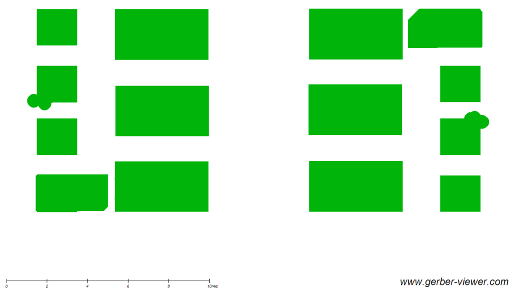

Figure A.6: Bottom copper layer of the EPC 2001C switching cell.

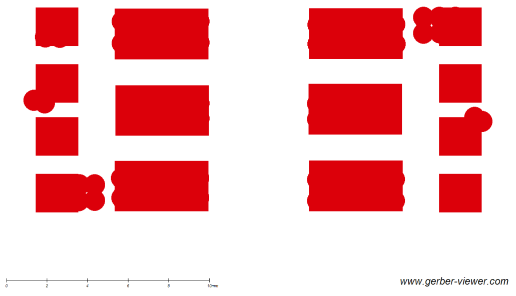

Figure A.7: Bottom soldermask layer of the EPC 2001C switching cell.

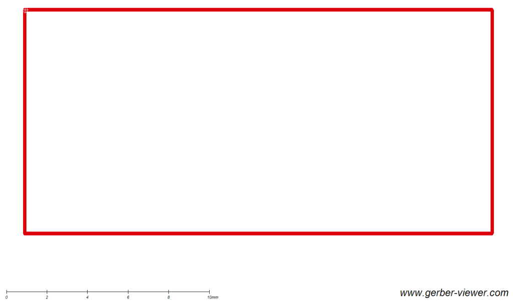

Figure A.8: Board outline of the EPC 2001C switching cell.

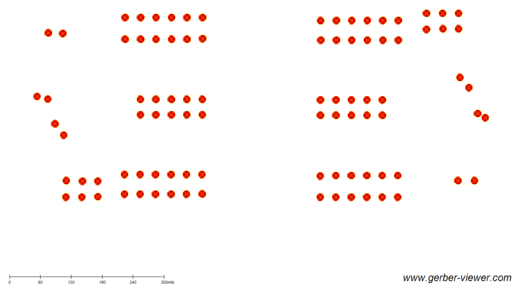

Figure A.9: Drill layer of the EPC 2001C switching cell.

A.3 PCB Layout – EPC 2029-34

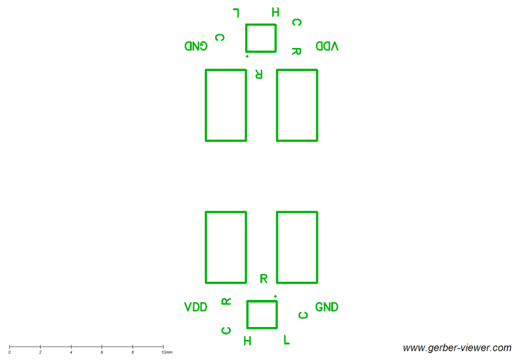

Figure A.10: Top silkcreen layer of the EPC 2029-34 switching cell.

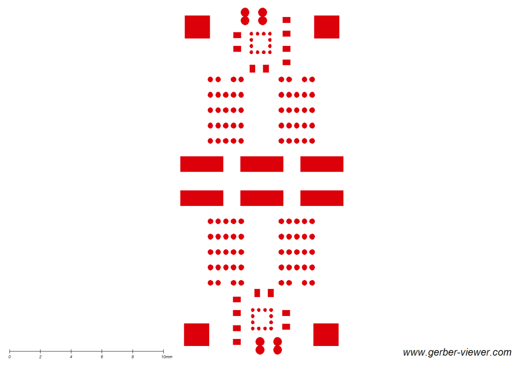

Figure A.11: Top soldermask layer of the EPC 2029-34 switching cell.

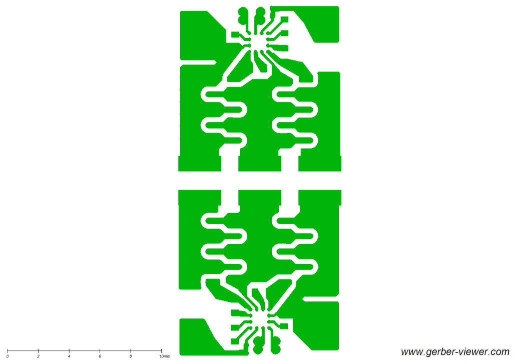

Figure A.12: Top copper layer of the EPC 2029-34 switching cell.

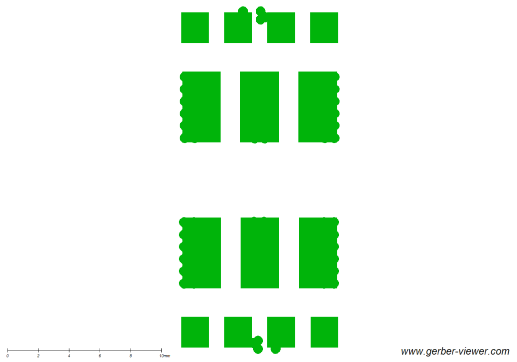

Figure A.13: Bottom copper layer of the EPC 2029-34 switching cell.

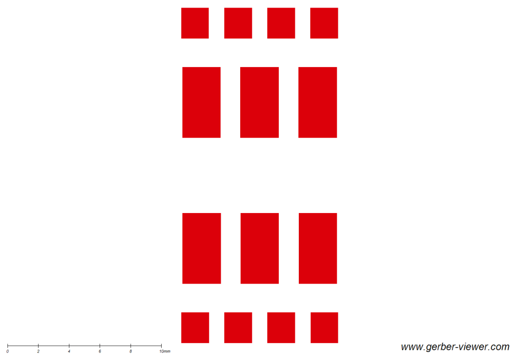

Figure A.14: Bottom soldermask layer of the EPC 2029-34 switching cell.

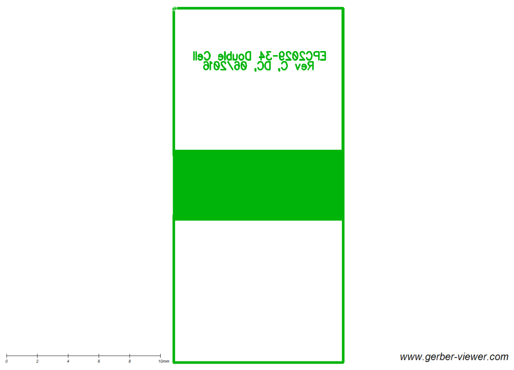

Figure A.15: Bottom silkscreen layer of the EPC 2029-34 switching cell.

Figure A.16: Board outline of the EPC 2029-34 switching cell.

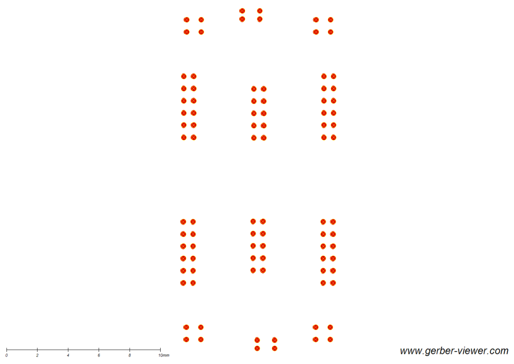

Figure A.17: Drill layer of the EPC 2029-34 switching cell.

APPENDIX B

FLEXIBLE CONVERTER

The flexible converter schematics, layout files, and basic microcontroller code are included here for reference.

B.1 Schematics

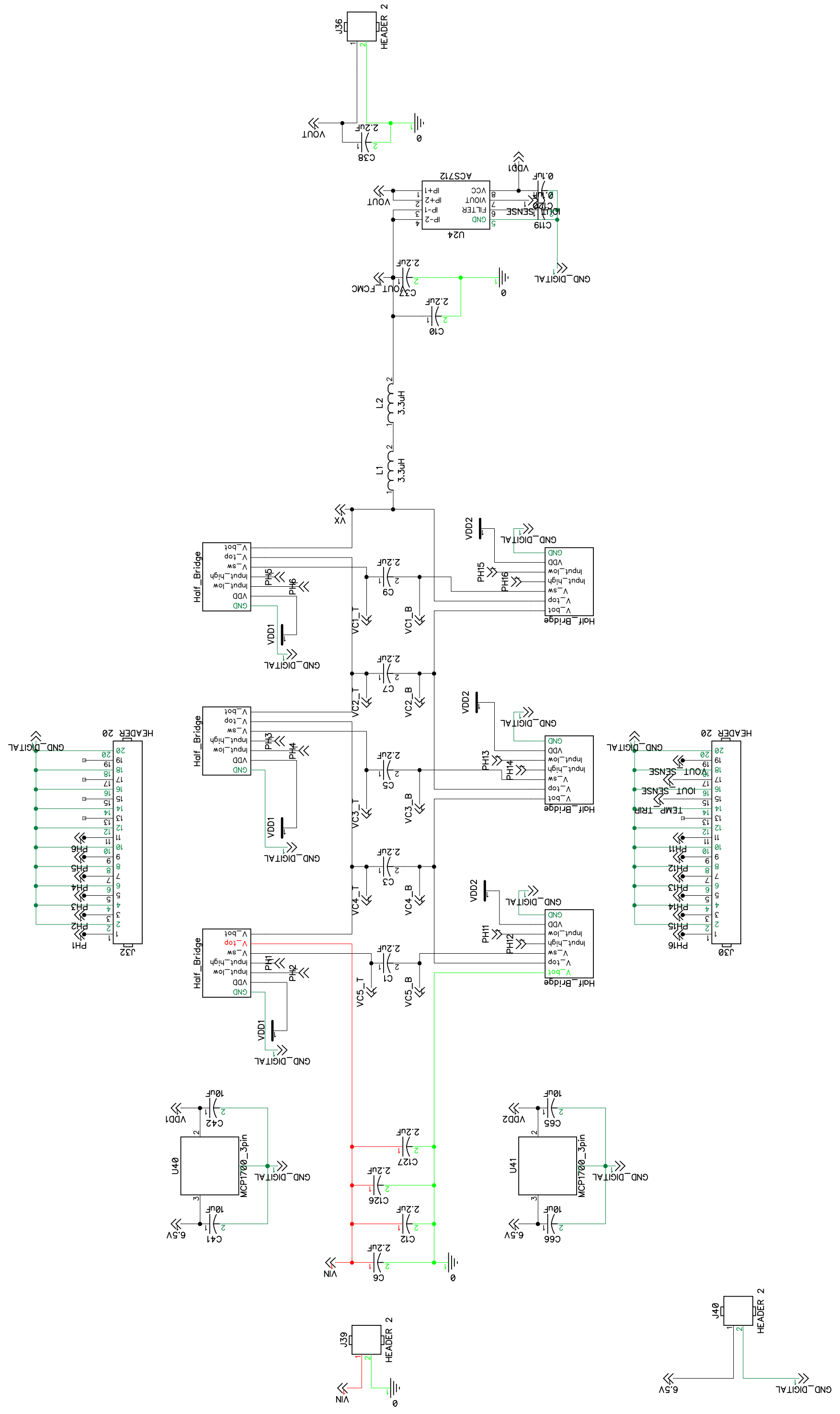

Figure B.1: Main 7-level FCML schematics for flexible converter.

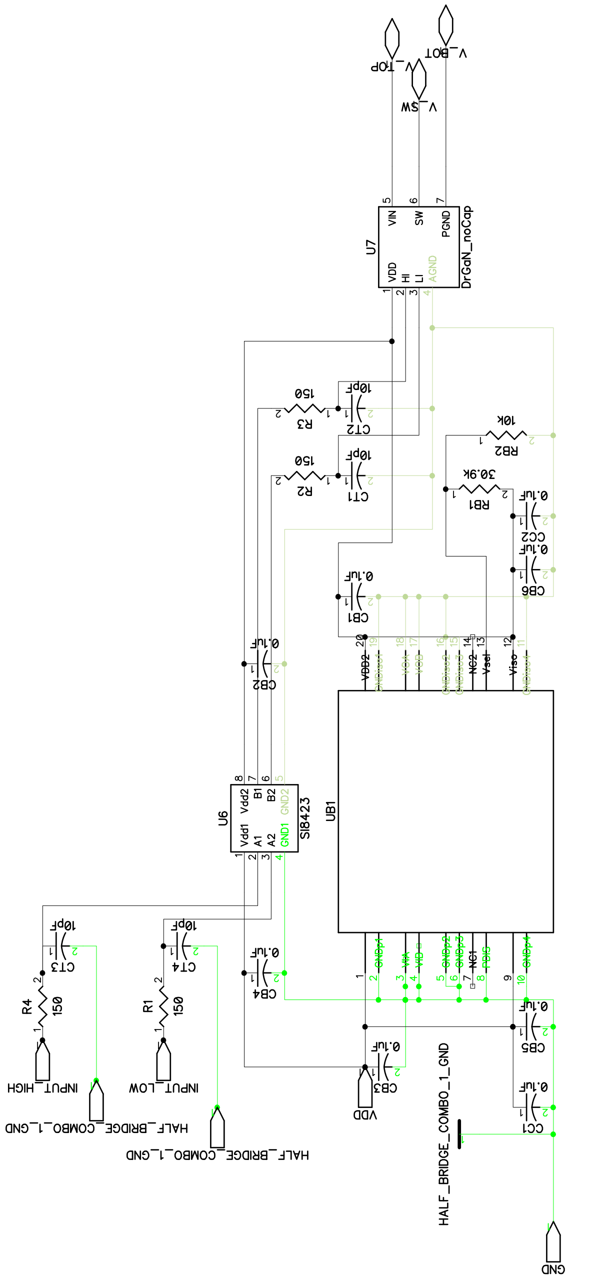

Figure B.2: Half-bridge isolation and gate drive interface schematics for MPPT.

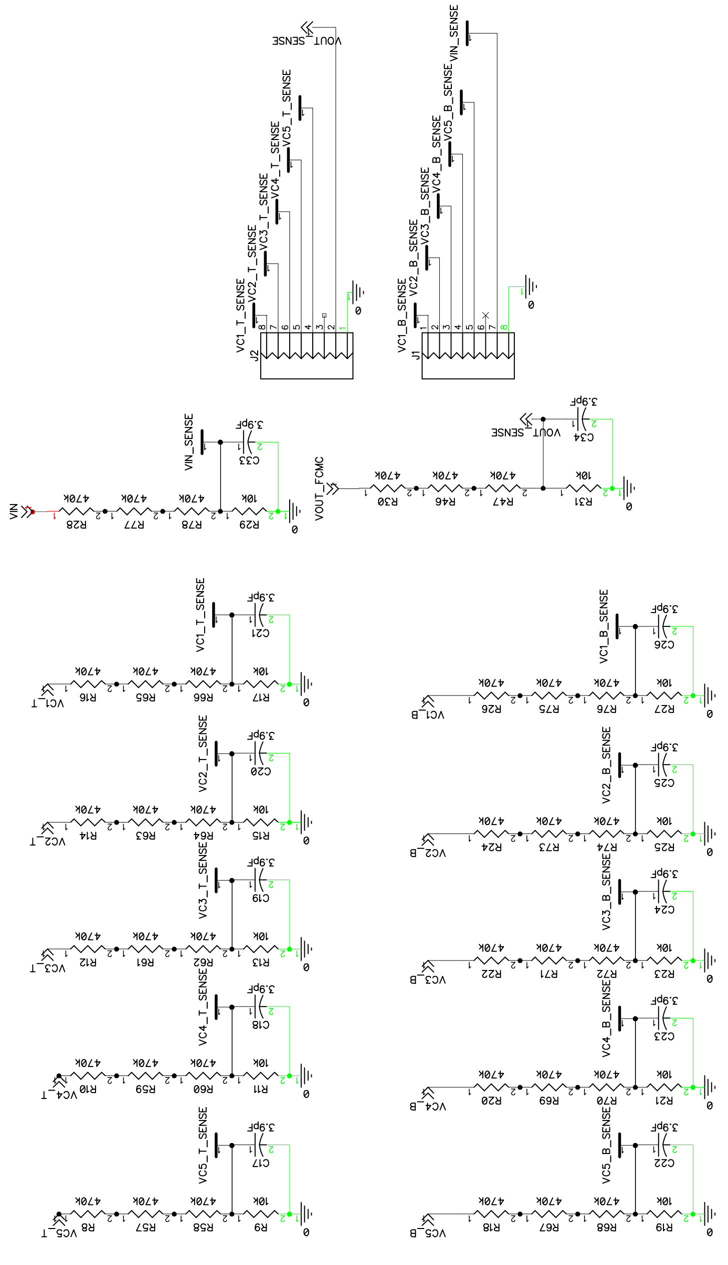

Figure B.3: Voltage divider and sensing schematics for flexible converter.

B.2 PCB Layout

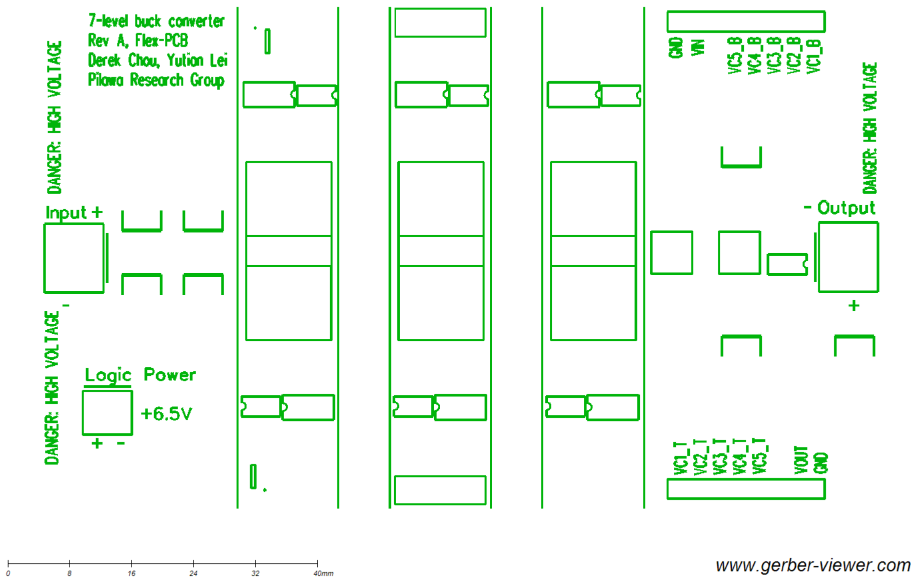

Figure B.4: Top silkscreen layer of the flexible converter.

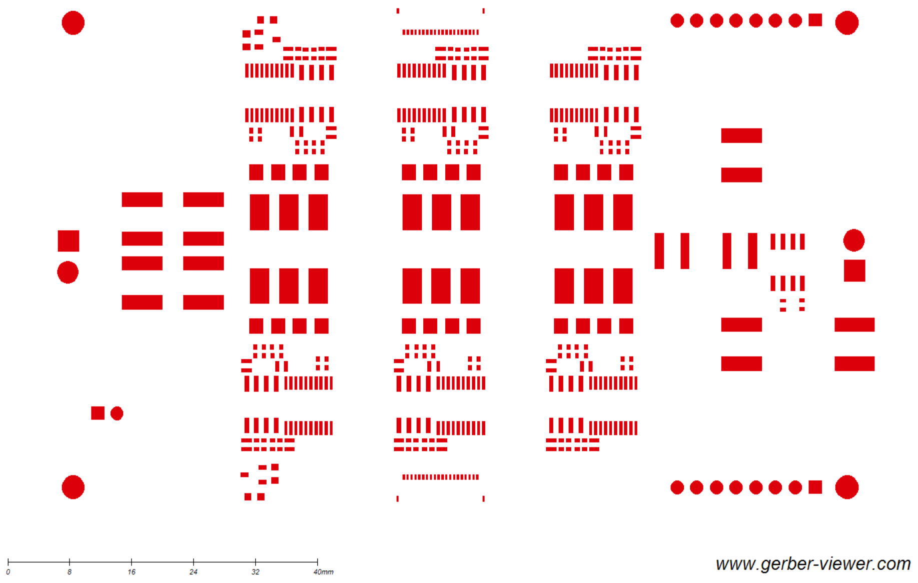

Figure B.5: Top soldermask layer of the flexible converter.

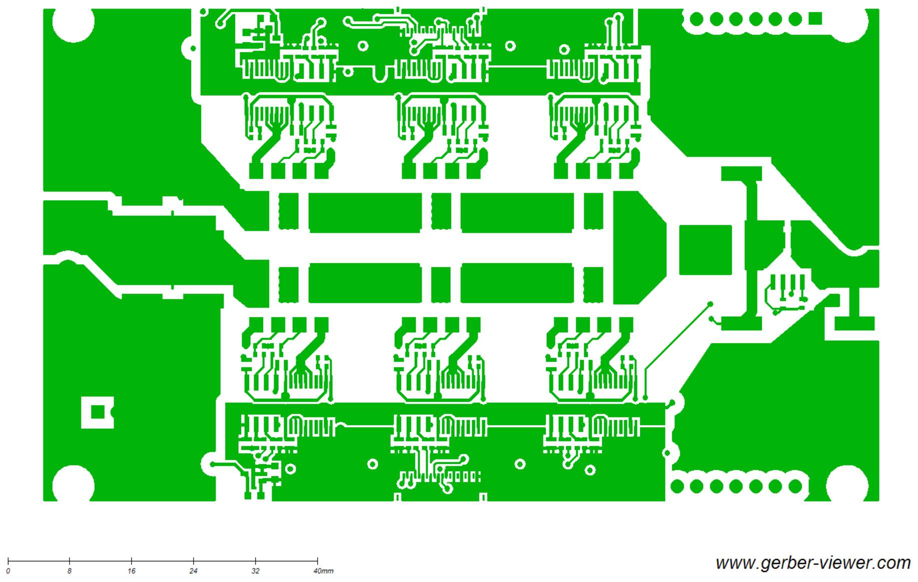

Figure B.6: Top copper layer of the flexible converter.

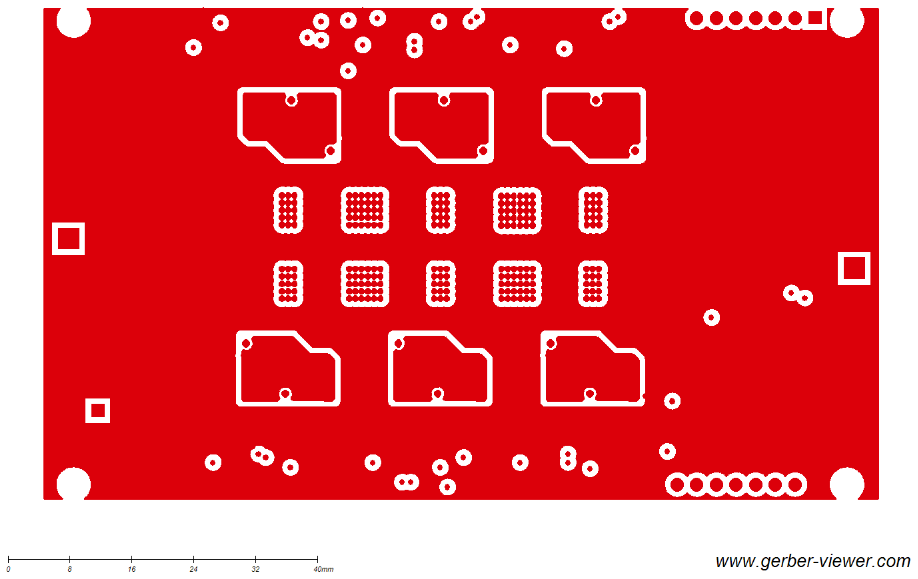

Figure B.7: Inner top copper layer of the flexible converter.

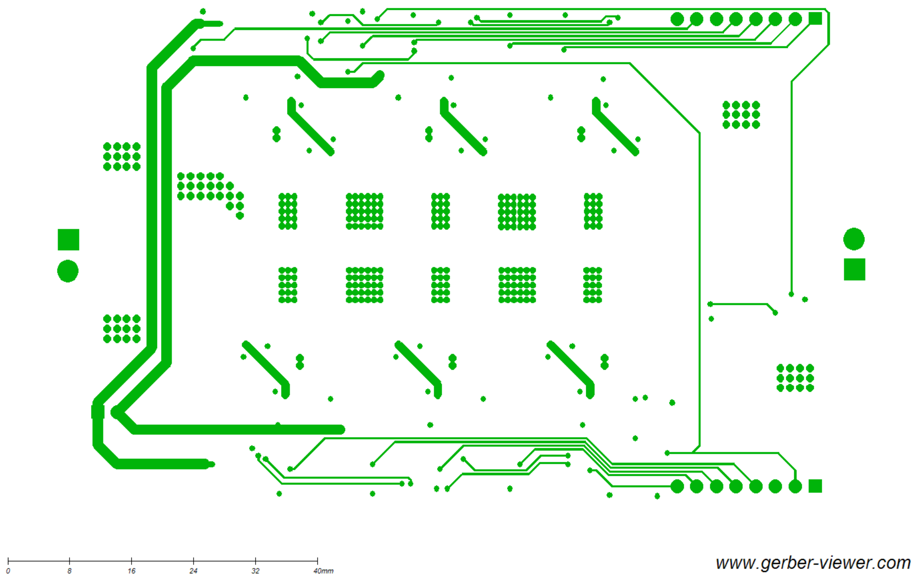

Figure B.8: Inner bottom copper layer of the flexible converter.

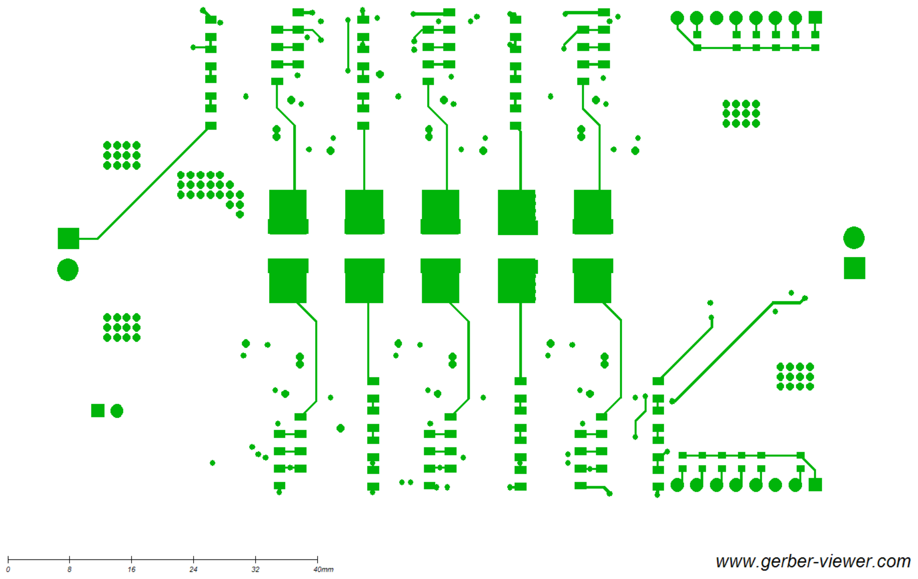

Figure B.9: Bottom copper layer of the flexible converter.

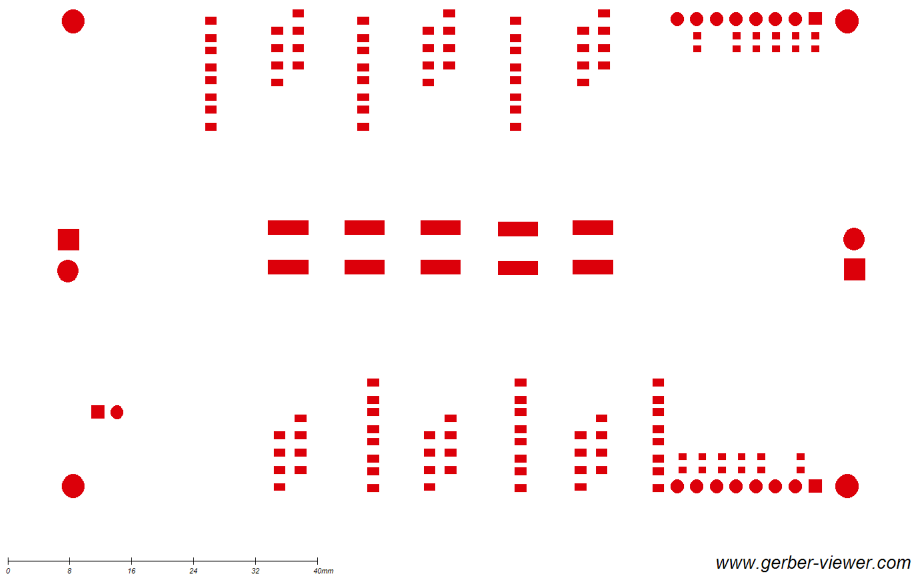

Figure B.10: Bottom soldermask layer of the flexible converter.

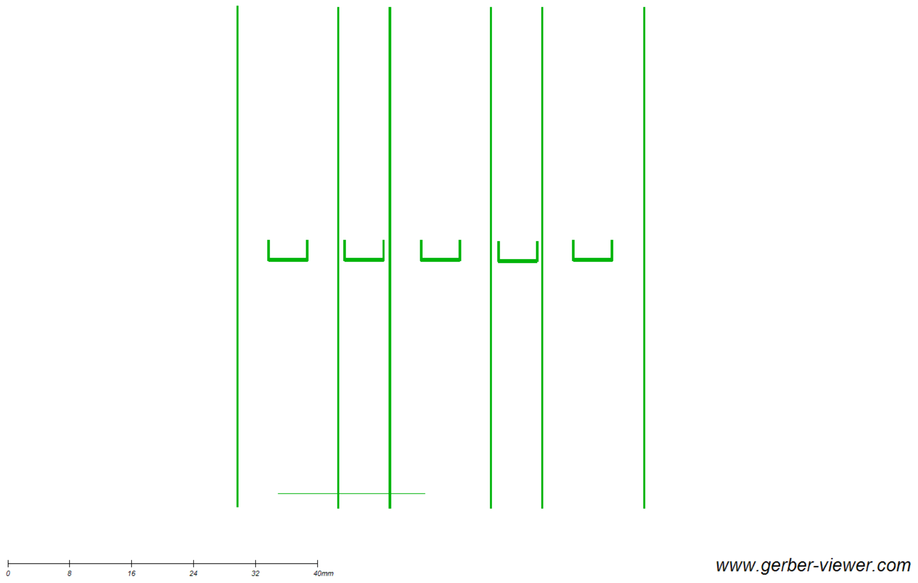

Figure B.11: Bottom silkscreen layer of the flexible converter.

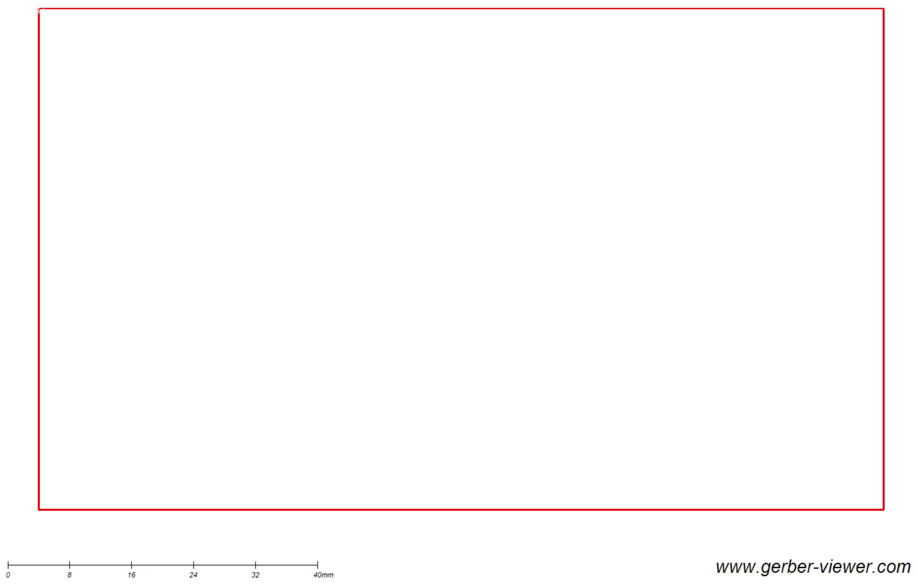

Figure B.12: Board outline of the flexible converter.

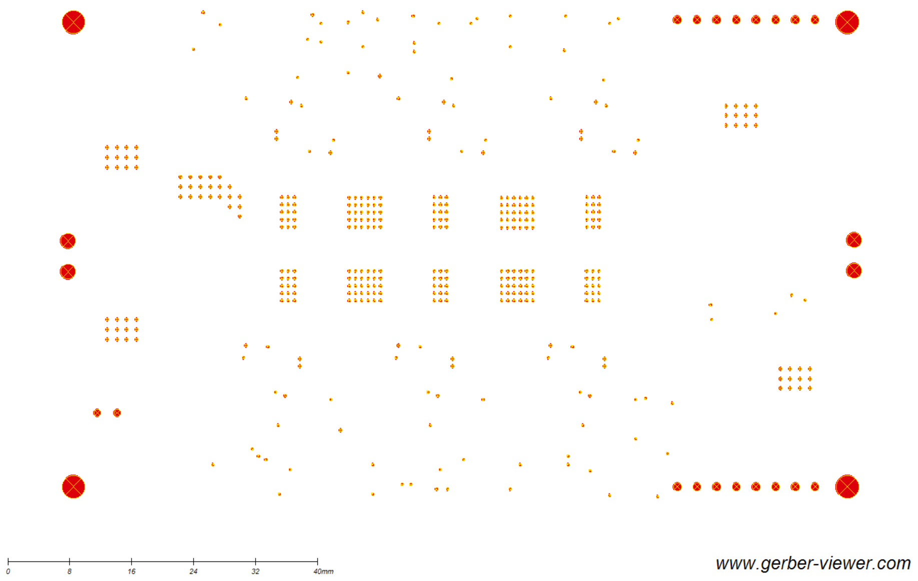

Figure B.13: Drill layer of the flexible converter.

B.3 Microcontroller Code

B.3.1 Main

```
//
    ###########################################################################

// FILE:     main.c
// TITLE:    Check PWM Dead-Band
//
//! \addtogroup cpu01_example_list
//! <h1> EPWM dead band control (epwm_deadband)</h1>
//!
//! This program is adapted from
//! on a scope.
//!
//! - ePWM1A is on GPIO0
//! - ePWM1B is on GPIO1
//! - ePWM2A is on GPIO2
//! - ePWM2B is on GPIO3
//! - ePWM3A is on GPIO4
//! - ePWM3B is on GPIO5
//! - ePWM4A is on GPIO6
//! - ePWM4B is on GPIO7
//! - ePWM5A is on GPIO8
//! - ePWM5B is on GPIO9
//! - ePWM6A is on GPIO10
//! - ePWM6B is on GPIO11
//!
//!
//! 3 Examples are included:
//! - ePWM1: Active low PWMs
//! - ePWM2: Active low complementary PWMs
//! - ePWM3: Active high complementary PWMs
//!
//! Each ePWM is configured to interrupt on the 3rd zero event.
//! When this happens the deadband is modified such that
//! 0 <= DB <= DB_MAX.  That is, the deadband will move up and
//! down between 0 and the maximum value.
//!
//! View the EPWM1A/B, EPWM2A/B and EPWM3A/B waveforms
//! via an oscilloscope
//
//
//
    ###########################################################################

// $TI Release: F2837xD Support Library v120 $
// $Release Date: Fri Aug 22 15:22:27 CDT 2014 $
//
    ###########################################################################
```

```
#include "F28x_Project.h"     // Device Headerfile and Examples Include File

#include "FCMLPhaseShift.h"
#include "ZVSADC.h"

const uint32_t maxfsw = 500;
const uint32_t midfsw = 400;
const uint32_t minfsw = 200;

void main(void) {

// Step 1. Initialize System Control:
// PLL, WatchDog, enable Peripheral Clocks
// This example function is found in the F2837xD_SysCtrl.c file.

        InitSysCtrl();

// Step 2. Initialize GPIO:
// This example function is found in the F2837xD_Gpio.c file and
// illustrates how to set the GPIO to its default state.
        InitGpio();

// Step 3. Clear all interrupts and initialize PIE vector table:
// Disable CPU interrupts
        DINT;

// Initialize the PIE control registers to their default state.
// The default state is all PIE interrupts disabled and flags
// are cleared.
// This function is found in the F2837xD_PieCtrl.c file.
        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 F2837xD_DefaultIsr.c.
// This function is found in F2837xD_PieVect.c.
        InitPieVectTable();

// Interrupts that are used in this example are re-mapped to
// ISR functions found within this file.
        EALLOW;
        // This is needed to write to EALLOW protected registers
        PieVectTable.TIMER0_INT = &cpu_timer0_isr;

        EDIS;
        // This is needed to disable write to EALLOW protected registers
// Step 4. Initialize all the Device Peripherals:
// This function is found in F28M36x_InitPeripherals.c

// Initialize the ePWM

        EALLOW;
        CpuSysRegs.PCLKCR0.bit.TBCLKSYNC = 0;   // disable PWM timer
        ClkCfgRegs.PERCLKDIVSEL.bit.EPWMCLKDIV = 0;

        EDIS;

        EALLOW;
        init_ADCs(); // initialize all ADCs (a,b,c,d)
        EDIS;

        DELAY_US(100);

        /* For this part to work properly, must initialize the isolators on
            the
           converter board first, and then turn on the microcontroller. */
        uint16_t NUM_SAMPLES = 32;
        uint16_t NUM_DUMMYREAD = 4;
        uint32_t vout = 0;
        uint32_t iout = 0;
        uint16_t resa;
        int i;
        for (i = 0; i < NUM_DUMMYREAD; i++) {
                dummyRead1();
        }
        for (i = 0; i < NUM_SAMPLES; i++) {
                dummyRead1();
                readADC1(&resa);
                iout += resa;
                DELAY_US(500);
        }
        iout /= NUM_SAMPLES;
        const float vcurrcorr = iout / 4096.0 * 3.3;


        Init_phase_shifted_pwm();       // Initial PWM for phase shifted
            operation
        Init_cputimer_sin_TMU();        // Initialize cputimer 1 for
            interrupt

        EALLOW;
        CpuSysRegs.PCLKCR0.bit.TBCLKSYNC = 1;   // start PWM timer

        EDIS;

// Step 5. User specific code, enable interrupts:

        // Enable CPU int1 which is connected to CPU-Timer 0,
        IER |= M_INT1;
        // Enable TINT0 in the PIE: Group 1 interrupt 7
        PieCtrlRegs.PIEIER1.bit.INTx7 = 1;

// Enable EPWM INTn in the PIE: Group 3 interrupt 1-3
        PieCtrlRegs.PIEIER3.bit.INTx1 = 1;

// Enable global Interrupts and higher priority real-time debug events:
        EINT;
        // Enable Global interrupt INTM
        ERTM;
        // Enable Global realtime interrupt DBGM

        NUM_SAMPLES = 8;
        uint16_t NUM_LOOPS = 8;
        /* Hack: offset by -0.1A because of lower efficiency. */
        const float curr[] = {0.646-0.1, 0.862-0.1, 1.508-0.1};
        // const float curr[] = {0.323, 0.431, 0.754};
        const float slope[] = {-462963, -154799};

        while(1) {
                uint32_t vout_accum = 0;
                uint32_t iout_accum = 0;
                int j;
                for (j = 0; j < NUM_LOOPS; j++) {
                        uint16_t res;
                        vout = 0;
                        iout = 0;
                        int i;
                        for (i = 0; i < NUM_DUMMYREAD; i++) {
                                dummyRead0();
                        }
                        for (i = 0; i < NUM_SAMPLES; i++) {
                                readADC0(&res);
                                DELAY_US(2);
                                vout += res;
                        }
                        for (i = 0; i < NUM_DUMMYREAD; i++) {
                                dummyRead1();
                        }
                        for (i = 0; i < NUM_SAMPLES; i++) {
                                readADC1(&res);
                                DELAY_US(2);
                                iout += res;
                        }
                        vout /= NUM_SAMPLES;
                        iout /= NUM_SAMPLES;
                        vout_accum += vout;
                        iout_accum += iout;
                        DELAY_US(500);
                }
                vout_accum /= NUM_LOOPS;
                iout_accum /= NUM_LOOPS;
                float vcurr = iout_accum / 4096.0 * 3.3;
                float icurr = (vcurr - vcurrcorr) / 0.066;
```

```
icurr = -icurr;
// float dipp = 12606060.0 / (fsw * 1000 * (num_levels - 1))
    ;
float newfsw;
uint16_t dr = 7;
uint16_t df = 3;

/* Linear fit to find best control point for ZVS. */
if (icurr < curr[0]) {
        newfsw = maxfsw;
        dr = 8;
        df = 4;
} else if (icurr >= curr[0] && icurr < curr[1]) {
        newfsw = ((maxfsw*1000.0)
                        + slope[0]*(icurr - curr[0])) /
                            1000;
        dr = 7;
        df = 3;
} else if (icurr >= curr[1]) {
        newfsw = ((midfsw*1000.0)
                        + slope[1]*(icurr - curr[1])) /
                            1000;
        dr = 6;
        df = 2;
}

/* Guard to prevent bad switching frequency calculations. */
if (newfsw > maxfsw) {
        newfsw = maxfsw;
        dr = 8;
        df = 4;
}
if (newfsw < minfsw) {
        newfsw = minfsw;
        dr = 6;
        df = 2;
}
float kp = 0.1;
newfsw = (int32_t)(fsw + (kp*(newfsw - fsw)));

/* Guards to prevent bad deadtimes or switching frequencies.
     */
if (newfsw > maxfsw) {
        newfsw = maxfsw;
        dr = 8;
        df = 4;
}
if (newfsw < minfsw) {
        newfsw = minfsw;
        dr = 6;
        df = 2;
}
if (dr > 8) {
        dr = 8;
```

```
                }
                if (dr < 6) {
                        dr = 6;
                }
                if (df > 4) {
                        df = 4;
                }
                if (df < 2) {
                        df = 2;
                }

                fsw = newfsw;
                deadtime_f = df;
                deadtime_r = dr;
        }

}
```

B.3.2 PWM generation

```
/*
 * 6to1_FCMC.c
 *
 *  Created on: May 8, 2015
 *      Authors: lei10, foulkes2, dchou5
 */

#include "F28x_Project.h"     // Device Headerfile and Examples Include File
#include "FCMLPhaseShift.h"

// Global variable definitions
float main_duty = DUTY;

// Local variable definitions
uint16_t deadtime_r = 7;     // Rising edge deadtime
uint16_t deadtime_f = 3;     // Falling edge deadtime
int32 phase = 360 / (num_levels - 1);     // phase shift of each ePWM, in
    degrees
int32 sysclk = 120000;  // system clock, in kHz
uint32_t PERIOD;       // period of the ePWM counter
int32_t fsw = 500;

float ps2_float;
float ps3_float;
float ps4_float;
float ps5_float;
float ps6_float;
float ps7_float;

int32 ps2;      // phase shift for ePWM2
int32 ps3;
int32 ps4;
int32 ps5;
```

```
int32 ps6;
int32 ps7;

int32 num_points;   // number of points in a complete sine wave
float step;
int32 index = 1;            // current position in sine wave

//dither variables

// Function definitions
void Init_cputimer_sin_TMU() {

        // Initialize GPIO for unfolder PWM
        GPIO_SetupPinMux(0, GPIO_MUX_CPU1, 0);
        GPIO_SetupPinOptions(0, GPIO_OUTPUT, GPIO_PUSHPULL);
        GPIO_SetupPinMux(1, GPIO_MUX_CPU1, 0);
        GPIO_SetupPinOptions(1, GPIO_OUTPUT, GPIO_PUSHPULL);
        GPIO_WritePin(0, 1);
        GPIO_WritePin(1, 0);
        // setup sine wave number of points and step size
        num_points = PERIOD * 2;
        //num_points = sysclk/120/10*2;                  // Duty ratio update
             frequency is 10 kHz
        step = 1.0 / num_points;

        // CPU Timer 0

        // Make sure timer is stopped:
        CpuTimer0Regs.TCR.bit.TSS = 1;

        // Initialize timer period to maximum:
        int32 timer0_period = sysclk * 1000 / fundamental_frequency /
            num_points;
        CpuTimer0Regs.PRD.all = timer0_period;

        // Initialize pre-scale counter to divide by 1 (SYSCLKOUT):
        CpuTimer0Regs.TPR.all = 0;
        CpuTimer0Regs.TPRH.all = 0;

        // Reload all counter register with period value:
        CpuTimer0Regs.TCR.bit.TRB = 1;

        CpuTimer0Regs.TCR.bit.TIE = 1;          // Enable timer 0 interrupt

        // Start the timer
        CpuTimer0Regs.TCR.bit.TSS = 0;
}

void Init_phase_shifted_pwm() {

        // enable PWM1, PWM2, PWM3, PWM4, PWM5, PWM6
        CpuSysRegs.PCLKCR2.bit.EPWM1 = 1;
        CpuSysRegs.PCLKCR2.bit.EPWM2 = 1;
        CpuSysRegs.PCLKCR2.bit.EPWM3 = 1;
        CpuSysRegs.PCLKCR2.bit.EPWM4 = 1;
        CpuSysRegs.PCLKCR2.bit.EPWM5 = 1;
        CpuSysRegs.PCLKCR2.bit.EPWM6 = 1;
        CpuSysRegs.PCLKCR2.bit.EPWM7 = 1;

        // Initialize GPIO pins for ePWM1, ePWM2, ePWM3, ePWM4, ePWM5, ePWM6
        // These functions are in the F28M36x_EPwm.c file
        InitEPwm2Gpio();
        InitEPwm3Gpio();
        InitEPwm4Gpio();
        InitEPwm5Gpio();
        InitEPwm6Gpio();
        InitEPwm7Gpio();

        PERIOD = sysclk / fsw;    // ePWM timer period

        // Phase shift for each ePWM
        ps2_float = 0;
        ps3_float = (phase * 1.0 / 360.0);
        ps4_float = (phase * 2.0 / 360.0);
        ps5_float = (phase * 3.0 / 360.0);
        ps6_float = (phase * 4.0 / 360.0);
        ps7_float = (phase * 5.0 / 360.0);

        ps2 = ((int) (PERIOD * ps2_float)) % PERIOD;
        ps3 = ((int) (PERIOD * ps3_float)) % PERIOD;
        ps4 = ((int) (PERIOD * ps4_float)) % PERIOD;
        ps5 = ((int) (PERIOD * ps5_float)) % PERIOD;
        ps6 = ((int) (PERIOD * ps6_float)) % PERIOD;
        ps7 = ((int) (PERIOD * ps7_float)) % PERIOD;

        InitEPwm_1();                           // Initialize each ePWM
        InitEPwm_2();
        InitEPwm_3();
        InitEPwm_4();
        InitEPwm_5();
        InitEPwm_6();
        InitEPwm_7();

}

void InitEPwm_1() {

        EPwm1Regs.TBPRD = PERIOD;                          // Set timer period
        EPwm1Regs.TBCTR = 0x0000;                        // Clear counter

        // Setup TBCLK
        EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Count up
        EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE; // Disable phase loading for
             the first ePWM, this becomes the master ePWM
        EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;      // Clock ratio to
            SYSCLKOUT
        EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;         // Same frequency as
            main clock
        EPwm1Regs.TBCTL.bit.SYNCOSEL = TB_CTR_ZERO; // send sync output
            signal when counter is zero

        // Setup compare
        EPwm1Regs.CMPA.bit.CMPA = PERIOD * main_duty;      // initial 50%
            duty ratio

        // Set actions
        EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR;             // Set PWM3A on
            Zero
        EPwm1Regs.AQCTLA.bit.ZRO = AQ_SET;

        // Active high complementary PWMs and Setup the deadband
        EPwm1Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
        EPwm1Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
        EPwm1Regs.DBCTL.bit.IN_MODE = DBA_ALL;
        EPwm1Regs.DBRED.bit.DBRED = deadtime_r;
        EPwm1Regs.DBFED.bit.DBFED = deadtime_f;

}

void InitEPwm_2() {

        EPwm2Regs.TBPRD = PERIOD;                        // Set timer period
        EPwm2Regs.TBPHS.bit.TBPHS = ps2;           // Phase is 0
        EPwm2Regs.TBCTR = 0x0000;                      // Clear counter

        // Setup TBCLK
        EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Count up
        EPwm2Regs.TBCTL.bit.PHSEN = TB_ENABLE;       // Enable phase loading
        EPwm2Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;      // Clock ratio to
            SYSCLKOUT
        EPwm2Regs.TBCTL.bit.CLKDIV = TB_DIV1;        // Same frequency as
            main clock
        EPwm2Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;      // pass sync in to
            sync out

        // Setup compare
        EPwm2Regs.CMPA.bit.CMPA = PERIOD * main_duty;      // initial 50%
            duty ratio

        // Set actions
        EPwm2Regs.AQCTLA.bit.CAU = AQ_CLEAR;             // Set PWM3A on
            Zero
        EPwm2Regs.AQCTLA.bit.ZRO = AQ_SET;

        // Active high complementary PWMs - Setup the deadband
        EPwm2Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
        EPwm2Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
        EPwm2Regs.DBCTL.bit.IN_MODE = DBA_ALL;
        EPwm2Regs.DBRED.bit.DBRED = deadtime_r;
        EPwm2Regs.DBFED.bit.DBFED = deadtime_f;

}
```

```
void InitEPwm_3() {

        EPwm3Regs.TBPRD = PERIOD;                        // Set timer period
        EPwm3Regs.TBPHS.bit.TBPHS = ps3;            //
        EPwm3Regs.TBCTR = 0x0000;                      // Clear counter

        // Setup TBCLK
        EPwm3Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Count up
        EPwm3Regs.TBCTL.bit.PHSEN = TB_ENABLE;        // Enable phase loading
        EPwm3Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Clock ratio to
            SYSCLKOUT
        EPwm3Regs.TBCTL.bit.CLKDIV = TB_DIV1;         // Same frequency as
            main clock
        EPwm3Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;       // pass sync in to
            sync out
        // Setup compare
        EPwm3Regs.CMPA.bit.CMPA = PERIOD * main_duty;       // initial 50%
            duty ratio

        // Set actions
        EPwm3Regs.AQCTLA.bit.CAU = AQ_CLEAR;              // Set PWM3A on
            Zero
        EPwm3Regs.AQCTLA.bit.ZRO = AQ_SET;

        // Active high complementary PWMs - Setup the deadband
        EPwm3Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
        EPwm3Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
        EPwm3Regs.DBCTL.bit.IN_MODE = DBA_ALL;
        EPwm3Regs.DBRED.bit.DBRED = deadtime_r;
        EPwm3Regs.DBFED.bit.DBFED = deadtime_f;

}

void InitEPwm_4() {

        EPwm4Regs.TBPRD = PERIOD;                        // Set timer period
        EPwm4Regs.TBPHS.bit.TBPHS = ps4;            //
        EPwm4Regs.TBCTR = 0x0000;                      // Clear counter

        // Setup TBCLK
        EPwm4Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Count up
        EPwm4Regs.TBCTL.bit.PHSEN = TB_ENABLE;        // Enable phase loading
        EPwm4Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Clock ratio to
            SYSCLKOUT
        EPwm4Regs.TBCTL.bit.CLKDIV = TB_DIV1;         // Same frequency as
            main clock
        EPwm4Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;       // pass sync in to
            sync out
        // Setup compare
        EPwm4Regs.CMPA.bit.CMPA = PERIOD * main_duty;       // initial 50%
            duty ratio

        // Set actions
        EPwm4Regs.AQCTLA.bit.CAU = AQ_CLEAR;             // Set PWM3A on
            Zero
        EPwm4Regs.AQCTLA.bit.ZRO = AQ_SET;

        // Active high complementary PWMs - Setup the deadband
        EPwm4Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
        EPwm4Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
        EPwm4Regs.DBCTL.bit.IN_MODE = DBA_ALL;
        EPwm4Regs.DBRED.bit.DBRED = deadtime_r;
        EPwm4Regs.DBFED.bit.DBFED = deadtime_f;

}
void InitEPwm_5() {

        EPwm5Regs.TBPRD = PERIOD;                        // Set timer period
        EPwm5Regs.TBPHS.bit.TBPHS = ps5;           //
        EPwm5Regs.TBCTR = 0x0000;                      // Clear counter

        // Setup TBCLK
        EPwm5Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Count up
        EPwm5Regs.TBCTL.bit.PHSEN = TB_ENABLE;       // Enable phase loading
        EPwm5Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;      // Clock ratio to
            SYSCLKOUT
        EPwm5Regs.TBCTL.bit.CLKDIV = TB_DIV1;        // Same frequency as
            main clock
        EPwm5Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;      // pass sync in to
            sync out
        // Setup compare
        EPwm5Regs.CMPA.bit.CMPA = PERIOD * main_duty;      // initial 50%
            duty ratio

        // Set actions
        EPwm5Regs.AQCTLA.bit.CAU = AQ_CLEAR;             // Set PWM3A on
            Zero
        EPwm5Regs.AQCTLA.bit.ZRO = AQ_SET;

        // Active high complementary PWMs - Setup the deadband
        EPwm5Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
        EPwm5Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
        EPwm5Regs.DBCTL.bit.IN_MODE = DBA_ALL;
        EPwm5Regs.DBRED.bit.DBRED = deadtime_r;
        EPwm5Regs.DBFED.bit.DBFED = deadtime_f;

}
void InitEPwm_6() {

        EPwm6Regs.TBPRD = PERIOD;                        // Set timer period
        EPwm6Regs.TBPHS.bit.TBPHS = ps6;           //
        EPwm6Regs.TBCTR = 0x0000;                      // Clear counter

        // Setup TBCLK
        EPwm6Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Count up
        EPwm6Regs.TBCTL.bit.PHSEN = TB_ENABLE;       // Enable phase loading
        EPwm6Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;      // Clock ratio to
            SYSCLKOUT
        EPwm6Regs.TBCTL.bit.CLKDIV = TB_DIV1;        // Same frequency as
            main clock
        EPwm6Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;      // pass sync in to
            sync out
        // Setup compare
        EPwm6Regs.CMPA.bit.CMPA = PERIOD * main_duty;      // initial 50%
            duty ratio

        // Set actions
        EPwm6Regs.AQCTLA.bit.CAU = AQ_CLEAR;             // Set PWM3A on
            Zero
        EPwm6Regs.AQCTLA.bit.ZRO = AQ_SET;

        // Active high complementary PWMs - Setup the deadband
        EPwm6Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
        EPwm6Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
        EPwm6Regs.DBCTL.bit.IN_MODE = DBA_ALL;
        EPwm6Regs.DBRED.bit.DBRED = deadtime_r;
        EPwm6Regs.DBFED.bit.DBFED = deadtime_f;

}

void InitEPwm_7() {

        EPwm7Regs.TBPRD = PERIOD;                        // Set timer period
        EPwm7Regs.TBPHS.bit.TBPHS = ps7;           //
        EPwm7Regs.TBCTR = 0x0000;                      // Clear counter

        // Setup TBCLK
        EPwm7Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Count up
        EPwm7Regs.TBCTL.bit.PHSEN = TB_ENABLE;       // Enable phase loading
        EPwm7Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;      // Clock ratio to
            SYSCLKOUT
        EPwm7Regs.TBCTL.bit.CLKDIV = TB_DIV1;        // Same frequency as
            main clock
        EPwm7Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;      // pass sync in to
            sync out
        // Setup compare
        EPwm7Regs.CMPA.bit.CMPA = PERIOD * main_duty;      // initial 50%
            duty ratio

        // Set actions
        EPwm7Regs.AQCTLA.bit.CAU = AQ_CLEAR;             // Set PWM3A on
            Zero
        EPwm7Regs.AQCTLA.bit.ZRO = AQ_SET;

        // Active high complementary PWMs - Setup the deadband
        EPwm7Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
        EPwm7Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
        EPwm7Regs.DBCTL.bit.IN_MODE = DBA_ALL;
        EPwm7Regs.DBRED.bit.DBRED = deadtime_r;
        EPwm7Regs.DBFED.bit.DBFED = deadtime_f;
```

```
}

__interrupt void cpu_timer0_isr(void) {
        index++;

#ifndef CONST_DUTY
//  if (index<=num_points/2){
//      main_duty = 1 -  __sinpuf32(argument);
//  }
//  else {
//      main_duty = 1 + __sinpuf32(argument);
//  }

//  main_duty = 0.5*(1+__sinpuf32(argument));
        float argument=step*index;
        main_duty = 0.5+(0.45*(__sinpuf32(argument)));
        if (main_duty < .002) {
                main_duty = .002;
        }
#endif

        int32 duty = PERIOD * main_duty;
        EPwm7Regs.CMPA.bit.CMPA = duty;
        EPwm6Regs.CMPA.bit.CMPA = duty;                 // update duty ratio
            in ePWMs
        EPwm5Regs.CMPA.bit.CMPA = duty;
        EPwm4Regs.CMPA.bit.CMPA = duty;
        EPwm3Regs.CMPA.bit.CMPA = duty;
        EPwm2Regs.CMPA.bit.CMPA = duty;
        EPwm1Regs.CMPA.bit.CMPA = duty;


        // Update frequency
        PERIOD = sysclk / fsw;    // ePWM timer period

        // Phase shift for each ePWM
        ps2_float = 0;
        ps3_float = (phase * 1.0 / 360.0);
        ps4_float = (phase * 2.0 / 360.0);
        ps5_float = (phase * 3.0 / 360.0);
        ps6_float = (phase * 4.0 / 360.0);
        ps7_float = (phase * 5.0 / 360.0);

        ps2 = ((int) (PERIOD * ps2_float)) % PERIOD;
        ps3 = ((int) (PERIOD * ps3_float)) % PERIOD;
        ps4 = ((int) (PERIOD * ps4_float)) % PERIOD;
        ps5 = ((int) (PERIOD * ps5_float)) % PERIOD;
        ps6 = ((int) (PERIOD * ps6_float)) % PERIOD;
        ps7 = ((int) (PERIOD * ps7_float)) % PERIOD;

        EPwm1Regs.TBPRD = PERIOD;
        EPwm2Regs.TBPRD = PERIOD;
        EPwm2Regs.TBPHS.bit.TBPHS = ps2;
        EPwm3Regs.TBPRD = PERIOD;
        EPwm3Regs.TBPHS.bit.TBPHS = ps3;
        EPwm4Regs.TBPRD = PERIOD;
        EPwm4Regs.TBPHS.bit.TBPHS = ps4;
        EPwm5Regs.TBPRD = PERIOD;
        EPwm5Regs.TBPHS.bit.TBPHS = ps5;
        EPwm6Regs.TBPRD = PERIOD;
        EPwm6Regs.TBPHS.bit.TBPHS = ps6;
        EPwm7Regs.TBPRD = PERIOD;
        EPwm7Regs.TBPHS.bit.TBPHS = ps7;

        // Update deadtime
        EPwm1Regs.DBRED.bit.DBRED = deadtime_r;
        EPwm1Regs.DBFED.bit.DBFED = deadtime_f;
        EPwm2Regs.DBRED.bit.DBRED = deadtime_r;
        EPwm2Regs.DBFED.bit.DBFED = deadtime_f;
        EPwm3Regs.DBRED.bit.DBRED = deadtime_r;
        EPwm3Regs.DBFED.bit.DBFED = deadtime_f;
        EPwm4Regs.DBRED.bit.DBRED = deadtime_r;
        EPwm4Regs.DBFED.bit.DBFED = deadtime_f;
        EPwm5Regs.DBRED.bit.DBRED = deadtime_r;
        EPwm5Regs.DBFED.bit.DBFED = deadtime_f;
        EPwm6Regs.DBRED.bit.DBRED = deadtime_r;
        EPwm6Regs.DBFED.bit.DBFED = deadtime_f;
        EPwm7Regs.DBRED.bit.DBRED = deadtime_r;
        EPwm7Regs.DBFED.bit.DBFED = deadtime_f;


        if (index == 1) {
                GpioDataRegs.GPACLEAR.bit.GPIO0 = 1; // Unfolder Vout_gnd
                    connects to ground
                GpioDataRegs.GPASET.bit.GPIO1 = 1; // Unfolder Vout_gnd
                    connects to ground
        }

#ifndef CONST_DUTY
        if (index==num_points/2) {
                GpioDataRegs.GPASET.bit.GPIO0 = 1; // Unfolder Vout_gnd
                    connectes to Vin
                GpioDataRegs.GPACLEAR.bit.GPIO1 = 1;// Unfolder Vout_gnd
                    connectes to Vin
        }
#endif

        if (index == num_points) {
                index = 0;
        }

#ifdef ENHANCED_BALANCING

#endif
        // Clear interrupt flag
        PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}
```

B.3.3 ADC reading

```
#include "F28x_Project.h"     // Device Headerfile and Examples Include File
#include "ZVSADC.h"

volatile int16 dummy_read = 0;

void init_ADCs() {
        //stop PWM clock, so no ADC will be triggerred and we can setup ADC
        EALLOW;
        ClkCfgRegs.PERCLKDIVSEL.bit.EPWMCLKDIV = 0x0; // make PWM clock the
            same as SYSCLK
        CpuSysRegs.PCLKCR0.bit.TBCLKSYNC = 0; //disable synchronization of
            all ePWMs to the TBCLK
        EDIS;

        // Initialize ADC sampling
        InitADCd(); // init ADCd
}

void InitADCa() {
        EALLOW;
        //write configurations
        AdcaRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4
        AdcSetMode(ADC_ADCA, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE);
        //Set pulse positions to late (at the end of conversion)
        AdcaRegs.ADCCTL1.bit.INTPULSEPOS = 1;
        //power up the ADC
        AdcaRegs.ADCCTL1.bit.ADCPWDNZ = 1;

        //SOC0 measure Vout on A2
        AdcaRegs.ADCSOC0CTL.bit.CHSEL = 4;  //SOC0 will convert pin A2
        AdcaRegs.ADCSOC0CTL.bit.ACQPS = 100; //sample window (# of SYSCLK,
            needs to corresponds to at least 75ns)
        AdcaRegs.ADCSOC0CTL.bit.TRIGSEL = 5; //trigger on ePWM1 SOCA/C
        AdcaRegs.ADCINTSEL1N2.bit.INT1SEL = 0; //end of SOC0 will set INT1
            flag
        AdcaRegs.ADCINTSEL1N2.bit.INT1E = 1;   //enable INT1 flag
        AdcaRegs.ADCINTSEL1N2.bit.INT1CONT = 0; //No further ADCINT1 pulses
            are generated until ADCINT1 flag is cleared by user
        AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //make sure INT1 flag is
            cleared
        // note that enabling flag is different from enabling interrupt

        //SOC1 measure AC_pos on pin A4
        AdcaRegs.ADCSOC1CTL.bit.CHSEL = 5;  //SOC1 will convert pin A4
        AdcaRegs.ADCSOC1CTL.bit.ACQPS = 100; //sample window (# of SYSCLK,
            needs to corresponds to at least 75ns)
        AdcaRegs.ADCSOC1CTL.bit.TRIGSEL = 6; //trigger on ePWM1 SOCB/D
        AdcaRegs.ADCINTSEL1N2.bit.INT2SEL = 1; //end of SOC1 will set INT2
            flag
        AdcaRegs.ADCINTSEL1N2.bit.INT2E = 1;   //enable INT2 flag
        AdcaRegs.ADCINTSEL1N2.bit.INT2CONT = 0; //No further ADCINT2 pulses
            are generated until ADCINT2 flag is cleared by user
        AdcaRegs.ADCINTFLGCLR.bit.ADCINT2 = 1; //make sure INT2 flag is
            cleared

        EDIS;
}

void InitADCb(void) {
        EALLOW;
        //write configurations
        AdcbRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4
        AdcSetMode(ADC_ADCB, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE);
        //Set pulse positions to late (at the end of conversion)
        AdcbRegs.ADCCTL1.bit.INTPULSEPOS = 1;
        //power up the ADC
        AdcbRegs.ADCCTL1.bit.ADCPWDNZ = 1;

        //SOC0 measure Iout on pin B2
        AdcbRegs.ADCSOC0CTL.bit.CHSEL = 4;  //SOC0 will convert pin B2
        AdcbRegs.ADCSOC0CTL.bit.ACQPS = 100; //sample window (# of SYSCLK,
            needs to corresponds to at least 75ns)
        AdcbRegs.ADCSOC0CTL.bit.TRIGSEL = 0; //trigger on ePWM1 SOCA/C
        AdcbRegs.ADCINTSEL1N2.bit.INT1SEL = 0; //end of SOC0 will set INT1
            flag
        AdcbRegs.ADCINTSEL1N2.bit.INT1E = 1;   //enable INT1 flag
        AdcbRegs.ADCINTSEL1N2.bit.INT1CONT = 0; //No further ADCINT1 pulses
            are generated until ADCINT1 flag is cleared by user
        AdcbRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //make sure INT1 flag is
            cleared

        //SOC1 measure Vrec on pin B3
        AdcbRegs.ADCSOC1CTL.bit.CHSEL = 5;  //SOC1 will convert pin B3
        AdcbRegs.ADCSOC1CTL.bit.ACQPS = 100; //sample window (# of SYSCLK,
            needs to corresponds to at least 75ns)
        AdcbRegs.ADCSOC1CTL.bit.TRIGSEL = 0; //trigger on ePWM1 SOCB/D
        AdcbRegs.ADCINTSEL1N2.bit.INT2SEL = 1; //end of SOC1 will set INT2
            flag
        AdcbRegs.ADCINTSEL1N2.bit.INT2E = 1;   //enable INT2 flag
        AdcbRegs.ADCINTSEL1N2.bit.INT2CONT = 0; //No further ADCINT2 pulses
            are generated until ADCINT2 flag is cleared by user
        AdcbRegs.ADCINTFLGCLR.bit.ADCINT2 = 1; //make sure INT2 flag is
            cleared

        EDIS;
}

void InitADCc(void) {
        EALLOW;
        //write configurations
        AdccRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4
        AdcSetMode(ADC_ADCC, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE);
        //Set pulse positions to late (at the end of conversion)
        AdccRegs.ADCCTL1.bit.INTPULSEPOS = 1;
        //power up the ADC
        AdccRegs.ADCCTL1.bit.ADCPWDNZ = 1;
        //SOC1 measure AC_neg on pin C4
        AdccRegs.ADCSOC1CTL.bit.CHSEL = 0;  //SOC1 will convert pin C2
        AdccRegs.ADCSOC1CTL.bit.ACQPS = 100; //sample window (# of SYSCLK,
            needs to corresponds to at least 75ns)
        AdccRegs.ADCSOC1CTL.bit.TRIGSEL = 0; //trigger on ePWM1 SOCB/D
        AdccRegs.ADCINTSEL1N2.bit.INT2SEL = 1; //end of SOC1 will set INT2
            flag
        AdccRegs.ADCINTSEL1N2.bit.INT2E = 1;   //enable INT2 flag
        AdccRegs.ADCINTSEL1N2.bit.INT2CONT = 0; //No further ADCINT2 pulses
            are generated until ADCINT2 flag is cleared by user
        AdccRegs.ADCINTFLGCLR.bit.ADCINT2 = 1; //make sure INT2 flag is
            cleared

        EDIS;
}

void InitADCd(void) {
        EALLOW;
        //write configurations
        AdcdRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4
        AdcSetMode(ADC_ADCD, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE);
        //Set pulse positions to late (at the end of conversion)
        AdcdRegs.ADCCTL1.bit.INTPULSEPOS = 1;
        //power up the ADC
        AdcdRegs.ADCCTL1.bit.ADCPWDNZ = 1;

        AdcdRegs.ADCSOC0CTL.bit.CHSEL = 2;
        AdcdRegs.ADCSOC0CTL.bit.ACQPS = 50; //sample window (# of SYSCLK,
            needs to corresponds to at least 75ns)
        AdcdRegs.ADCSOC0CTL.bit.TRIGSEL = 0;
        AdcdRegs.ADCINTSEL1N2.bit.INT1SEL = 0; //end of SOC0 will set INT1
            flag
        AdcdRegs.ADCINTSEL1N2.bit.INT1E = 1;   //enable INT1 flag
        AdcdRegs.ADCINTSEL1N2.bit.INT1CONT = 0; //No further ADCINT1 pulses
            are generated until ADCINT1 flag is cleared by user
        AdcdRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //make sure INT1 flag is
            cleared

        AdcdRegs.ADCSOC1CTL.bit.CHSEL = 3;
        AdcdRegs.ADCSOC1CTL.bit.ACQPS = 50; //sample window (# of SYSCLK,
            needs to corresponds to at least 75ns)
        AdcdRegs.ADCSOC1CTL.bit.TRIGSEL = 0;
        AdcdRegs.ADCINTSEL1N2.bit.INT2SEL = 1; //end of SOC1 will set INT2
            flag
        AdcdRegs.ADCINTSEL1N2.bit.INT2E = 1;   //enable INT2 flag
        AdcdRegs.ADCINTSEL1N2.bit.INT2CONT = 0; //No further ADCINT1 pulses
            are generated until ADCINT1 flag is cleared by user
        AdcdRegs.ADCINTFLGCLR.bit.ADCINT2 = 1; //make sure INT1 flag is
            cleared

        EDIS;
}
```

```
/**
 * Do a dummy read of the ADC. Follow directly with an actual read.
 * This is to clear the inaccuracies when reading the ADC the first time in
     a while.
 */
void dummyRead() {
        AdcdRegs.ADCSOCFRC1.all = 1<<0;
        while (AdcdRegs.ADCINTFLG.bit.ADCINT1 != 1);
        AdcdRegs.ADCSOCFRC1.all = 1<<1;
        while (AdcdRegs.ADCINTFLG.bit.ADCINT2 != 1);
        AdcdRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //clear INT1 flag
        AdcdRegs.ADCINTFLGCLR.bit.ADCINT2 = 1; //clear INT2 flag
}


void dummyRead0() {
        AdcdRegs.ADCSOCFRC1.all = 1<<0;
        while (AdcdRegs.ADCINTFLG.bit.ADCINT1 != 1);
        AdcdRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //clear INT1 flag
}


void dummyRead1() {
        AdcdRegs.ADCSOCFRC1.all = 1<<1;
        while (AdcdRegs.ADCINTFLG.bit.ADCINT2 != 1);
        AdcdRegs.ADCINTFLGCLR.bit.ADCINT2 = 1; //clear INT2 flag
}

/**
 * Read all of the ADC results. Must be preceded with a dummy read.
 */
void readAllADC(uint16_t* res) {
        AdcdRegs.ADCSOCFRC1.all = 1<<0;
        while (AdcdRegs.ADCINTFLG.bit.ADCINT1 != 1);
        AdcdRegs.ADCSOCFRC1.all = 1<<1;
        while (AdcdRegs.ADCINTFLG.bit.ADCINT2 != 1);
        res[0] = AdcdResultRegs.ADCRESULT0;
        res[1] = AdcdResultRegs.ADCRESULT1;
        AdcdRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //clear INT1 flag
        AdcdRegs.ADCINTFLGCLR.bit.ADCINT2 = 1; //clear INT2 flag
}


/**
 * Read all of the ADC results. Must be preceded with a dummy read.
 */
void readADC0(uint16_t* res) {
        AdcdRegs.ADCSOCFRC1.all = 1<<0;
        while (AdcdRegs.ADCINTFLG.bit.ADCINT1 != 1);
        *res = AdcdResultRegs.ADCRESULT0;
        AdcdRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //clear INT1 flag
}
```

```
void readADC1(uint16_t* res) {
        AdcdRegs.ADCSOCFRC1.all = 1<<1;
        while (AdcdRegs.ADCINTFLG.bit.ADCINT2 != 1);
        *res = AdcdResultRegs.ADCRESULT1;
        AdcdRegs.ADCINTFLGCLR.bit.ADCINT2 = 1; //clear INT2 flag

}
```

APPENDIX C

MAXIMUM POWER POINT TRACKER

The MPPT schematics, layout files, and basic microcontroller code are included here for reference.

C.1 Schematics

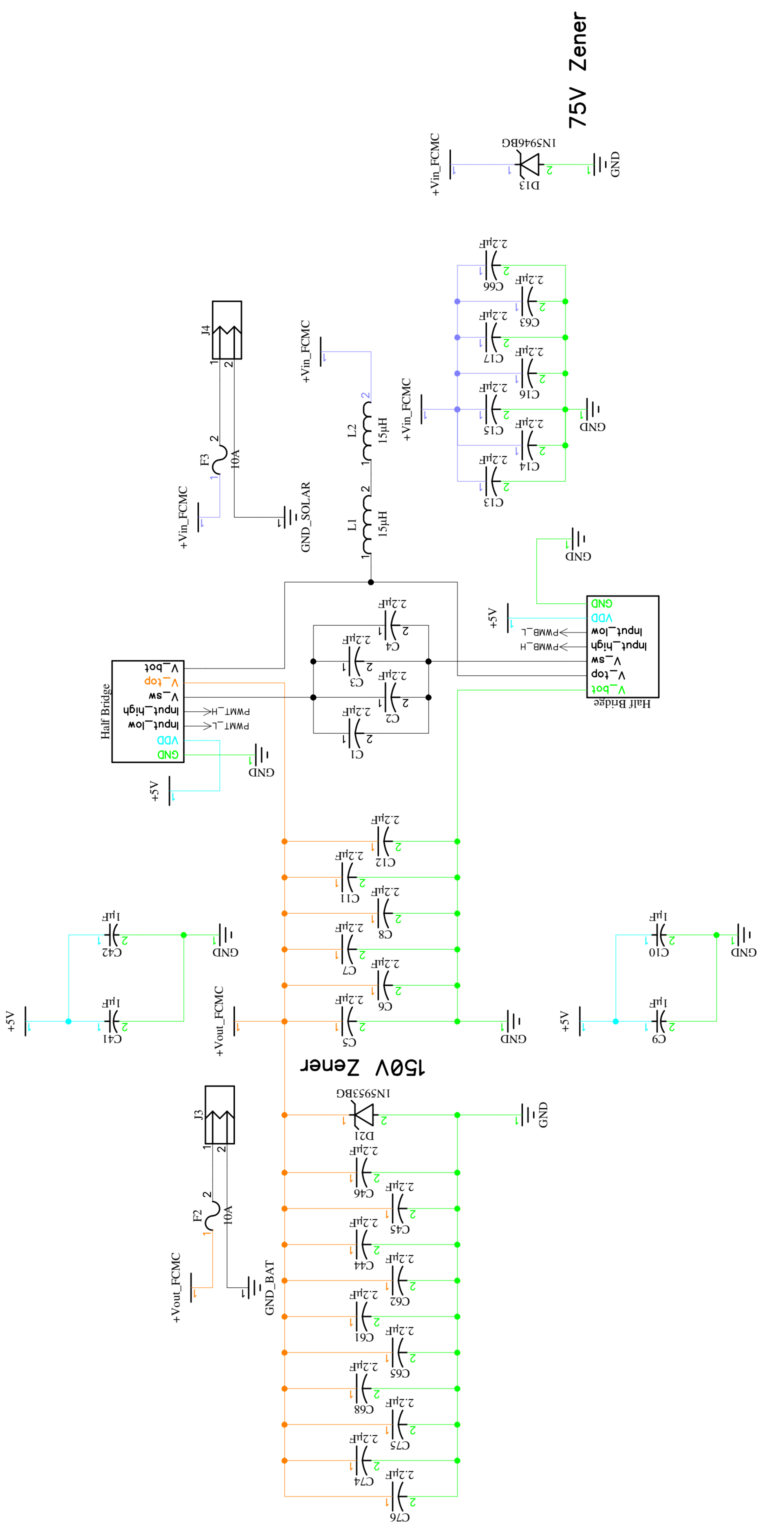

Figure C.1: Main power schematics for MPPT.

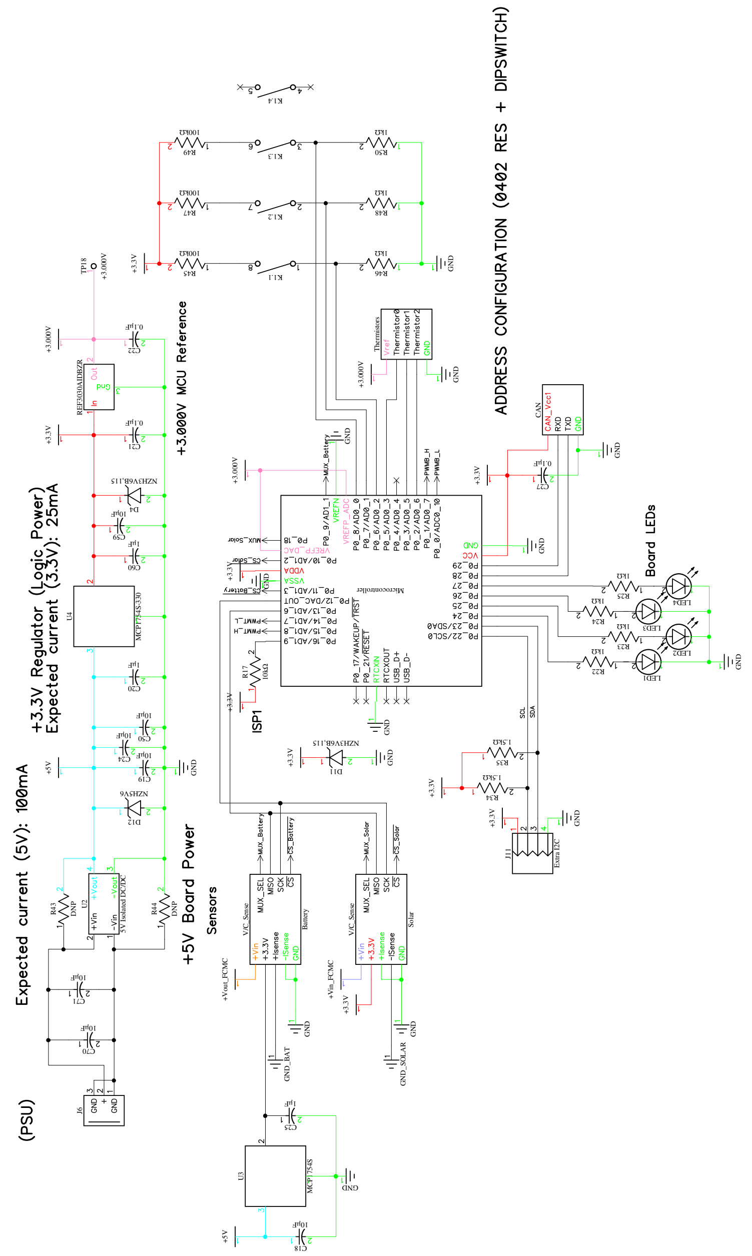

Figure C.2: Main logic schematics for MPPT.

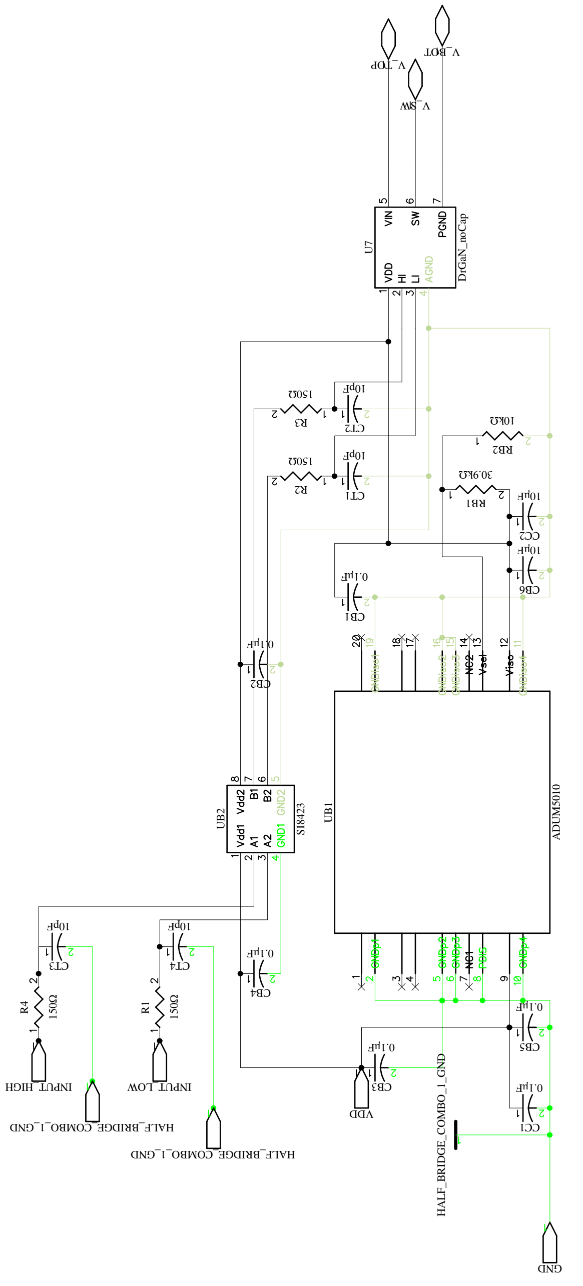

Figure C.3: Half-bridge isolation and gate drive interface schematics for MPPT.

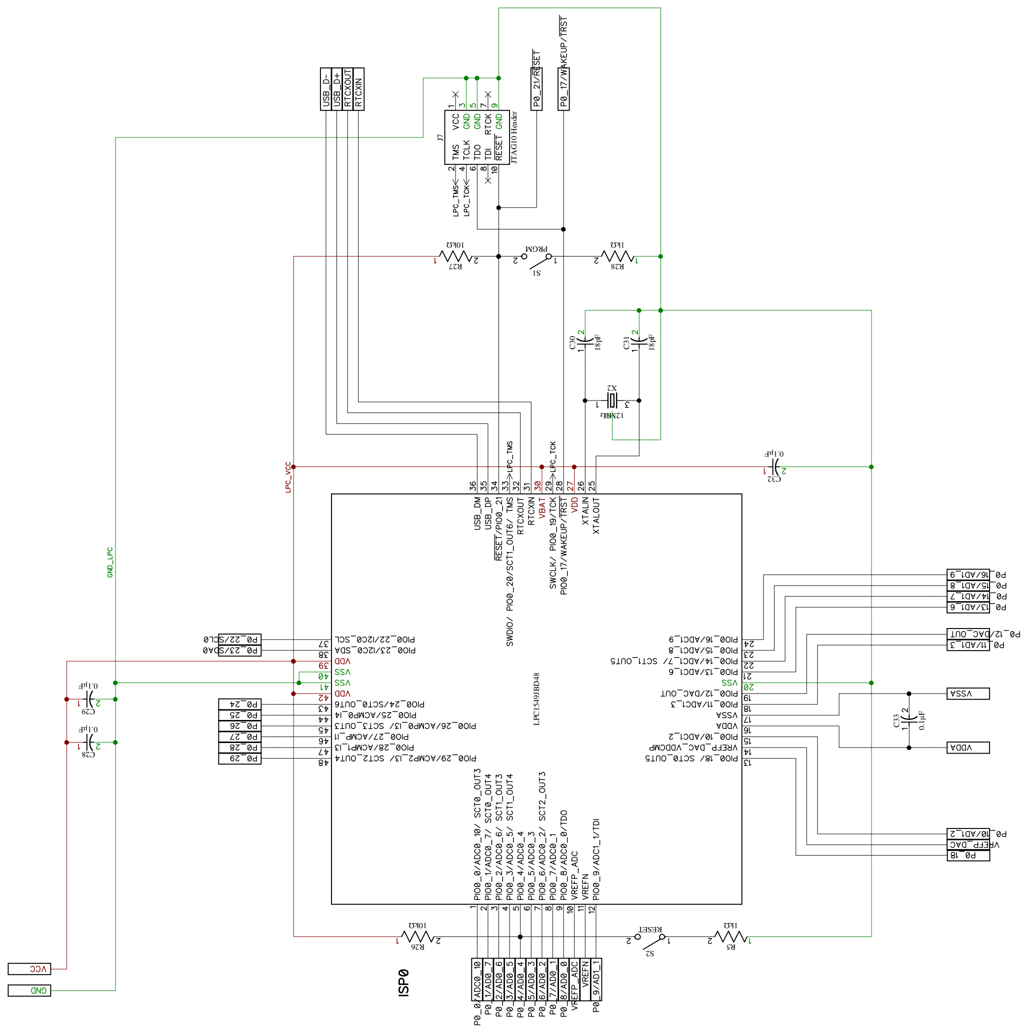

Figure C.4: Microcontroller block schematics for MPPT.

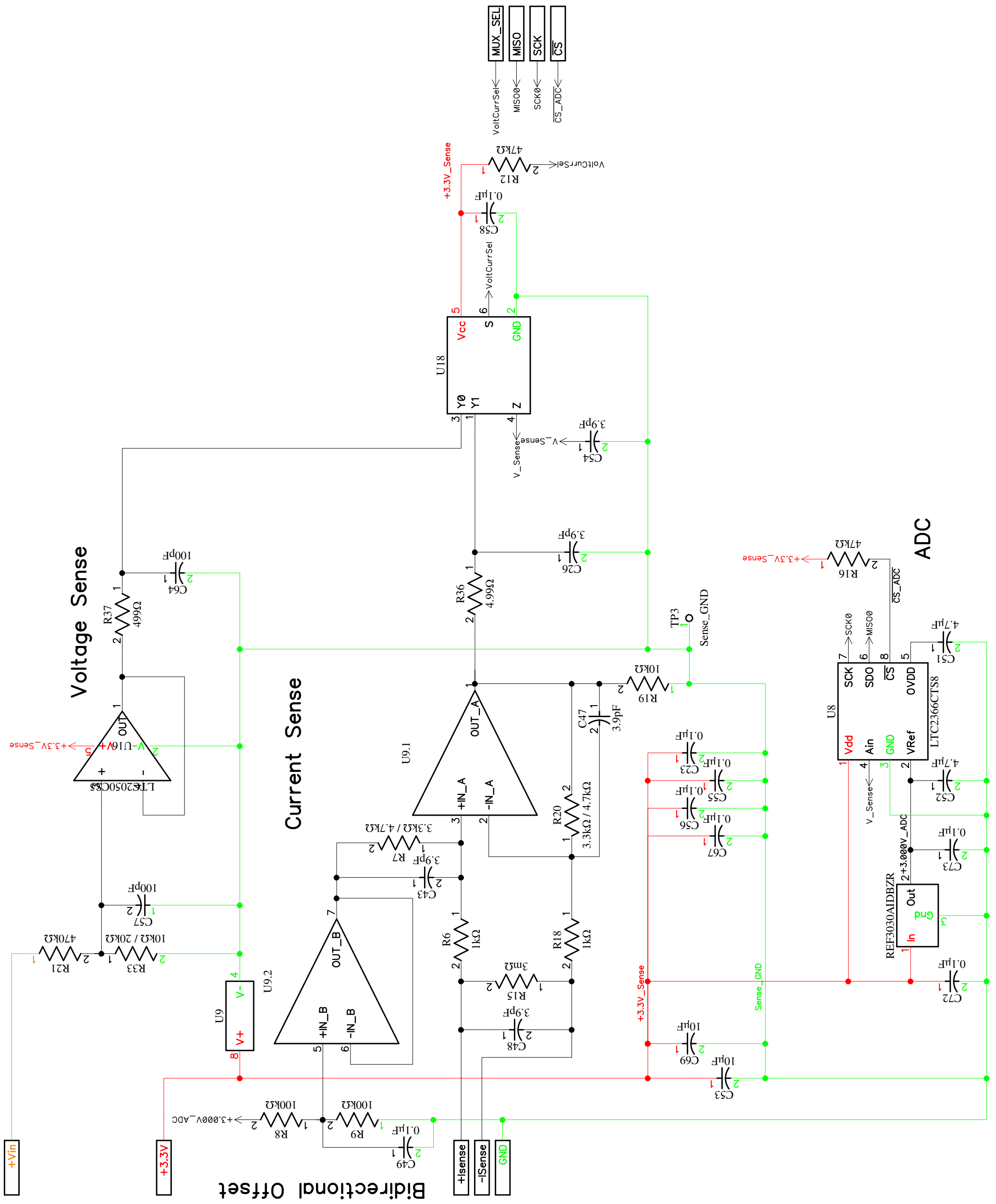

Figure C.5: Voltage and current sense schematics for MPPT.

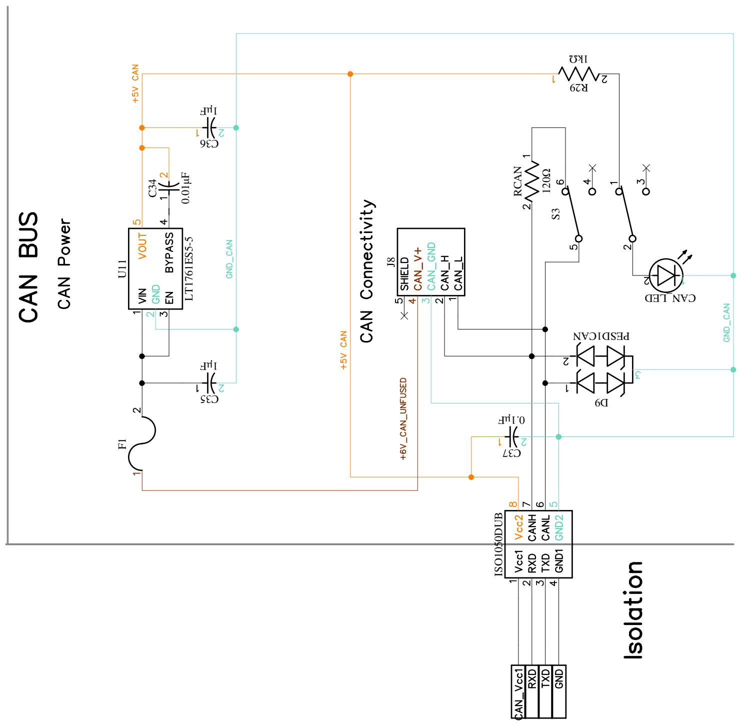

Figure C.6: Controller Area Network (CAN) interface schematics for MPPT.

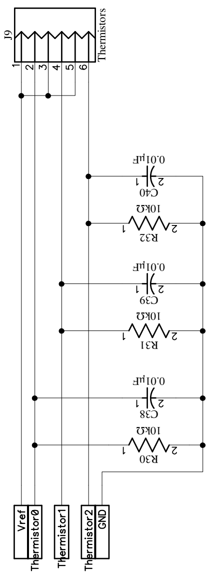

Figure C.7: Temperature sense schematics for MPPT.

C.2 PCB Layout

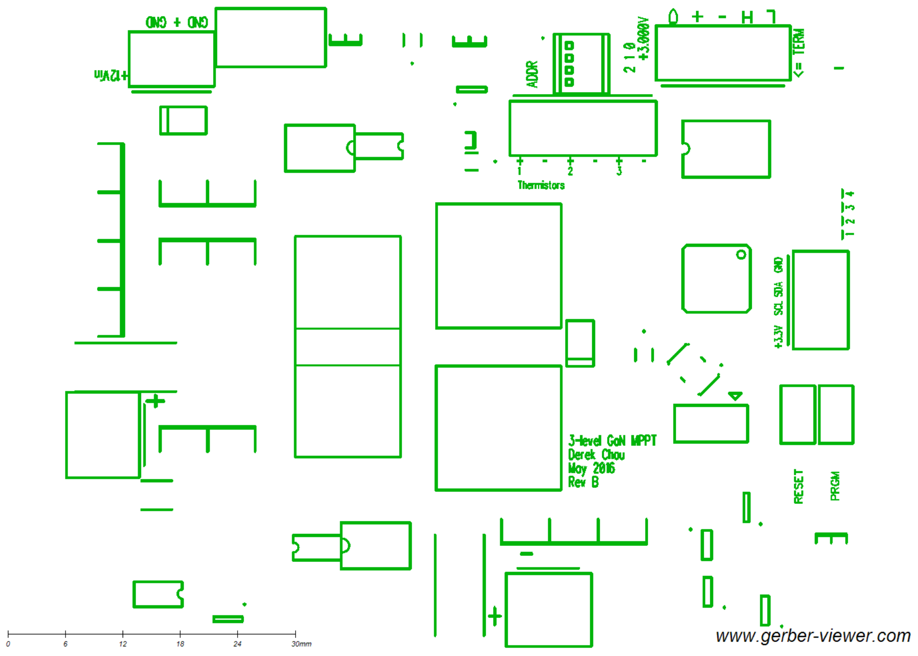

Figure C.8: Top silkscreen layer of the MPPT.

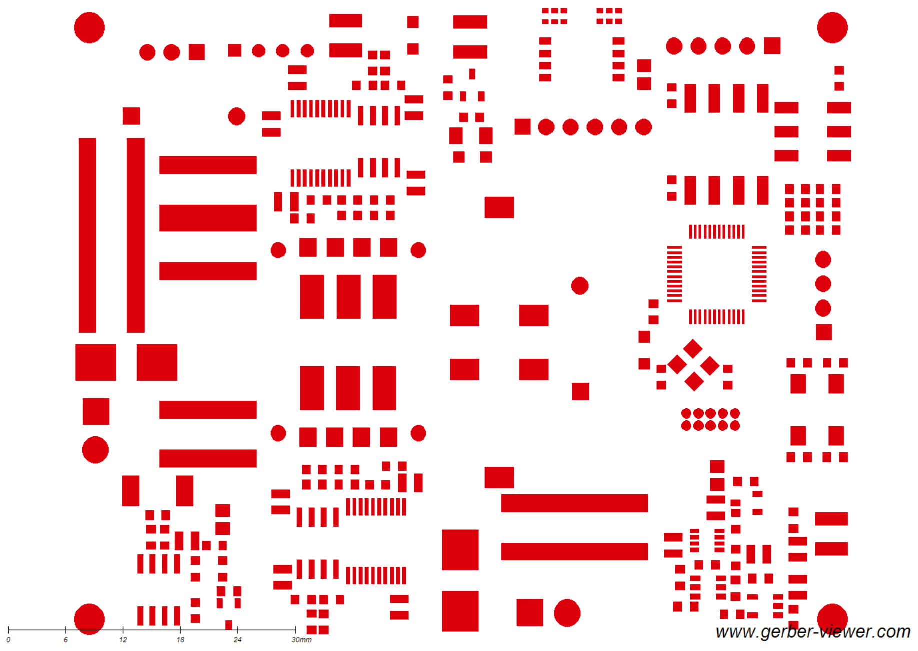

Figure C.9: Top soldermask layer of the MPPT.

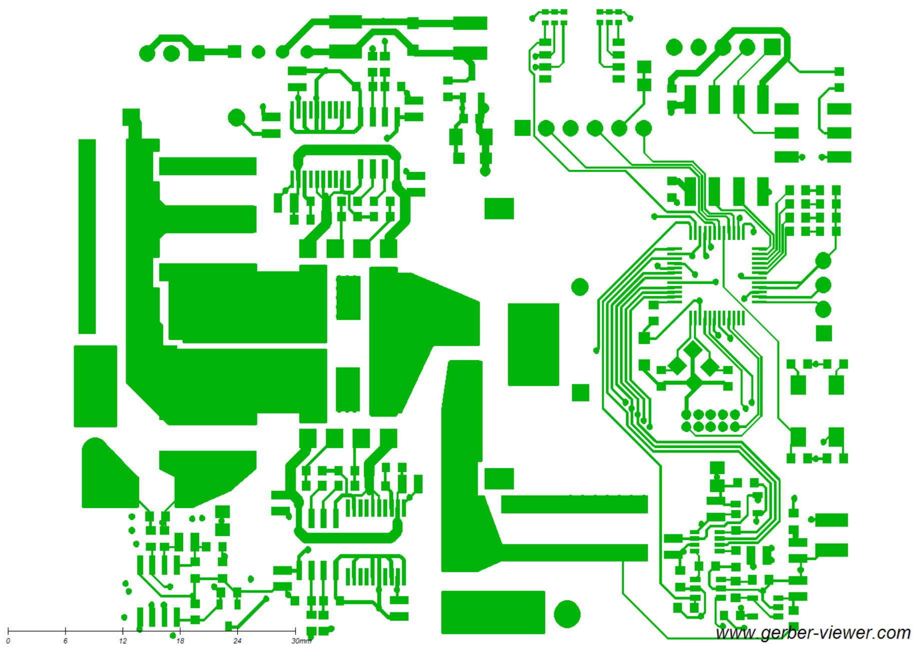

Figure C.10: Top copper layer of the MPPT.

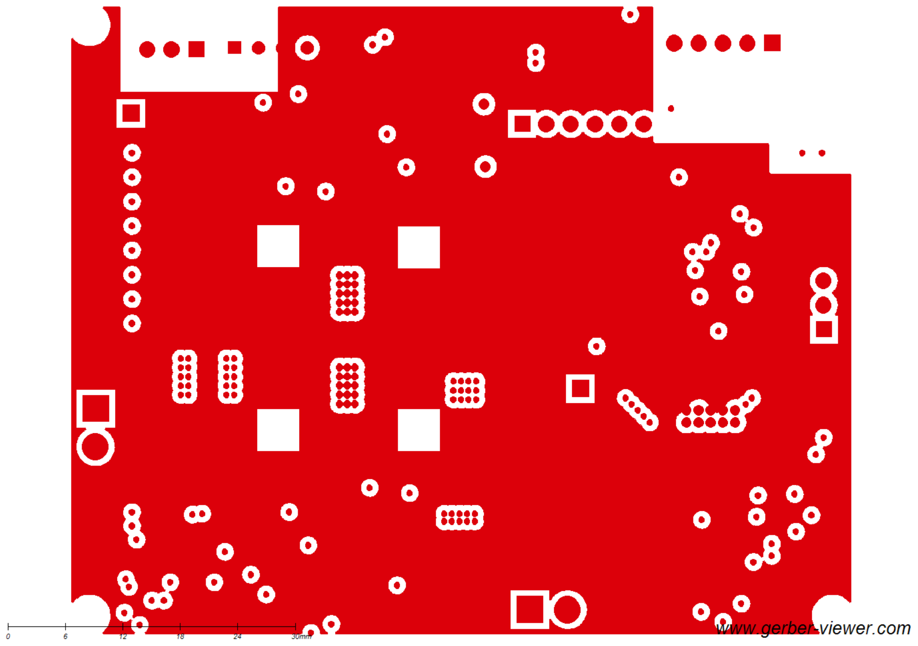

Figure C.11: Inner top copper layer of the MPPT.

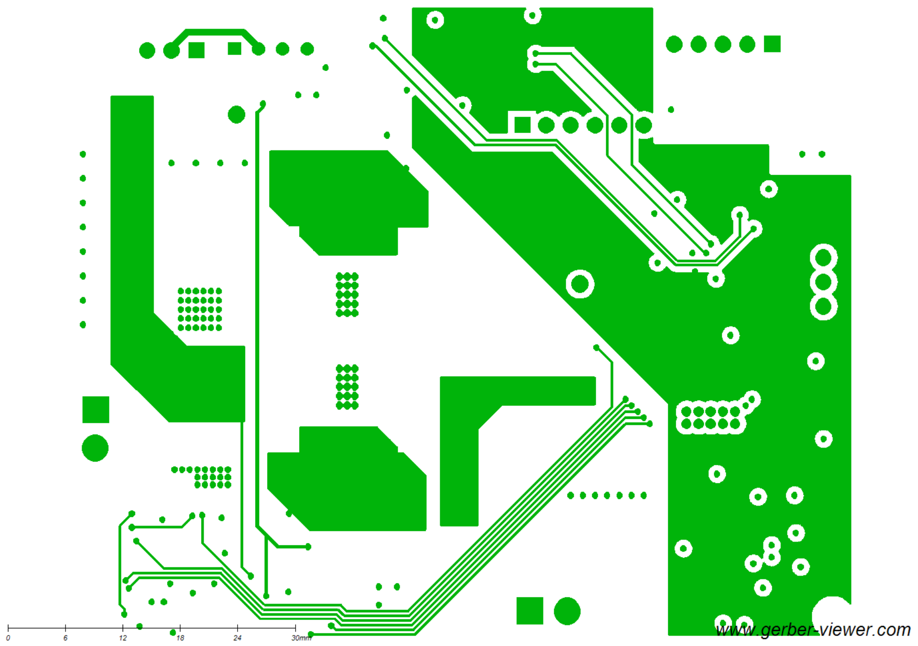

Figure C.12: Inner bottom copper layer of the MPPT.

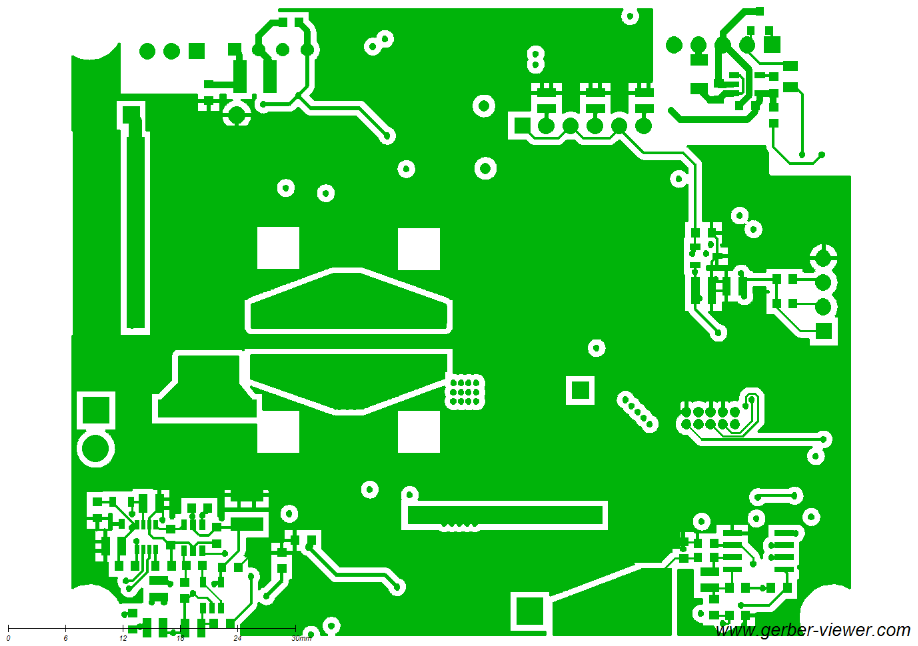

Figure C.13: Bottom copper layer of the MPPT.

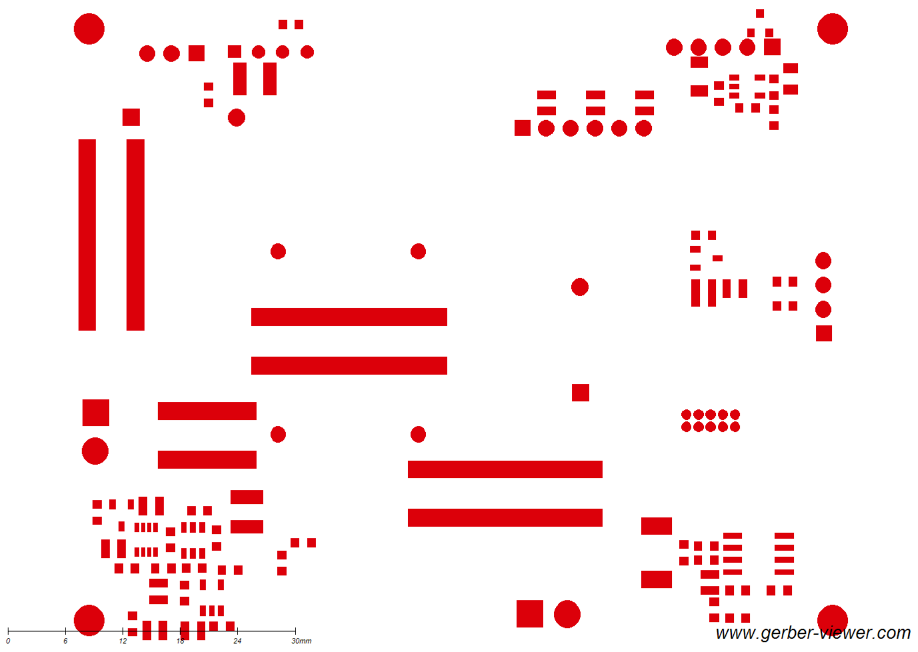

Figure C.14: Bottom soldermask layer of the MPPT.

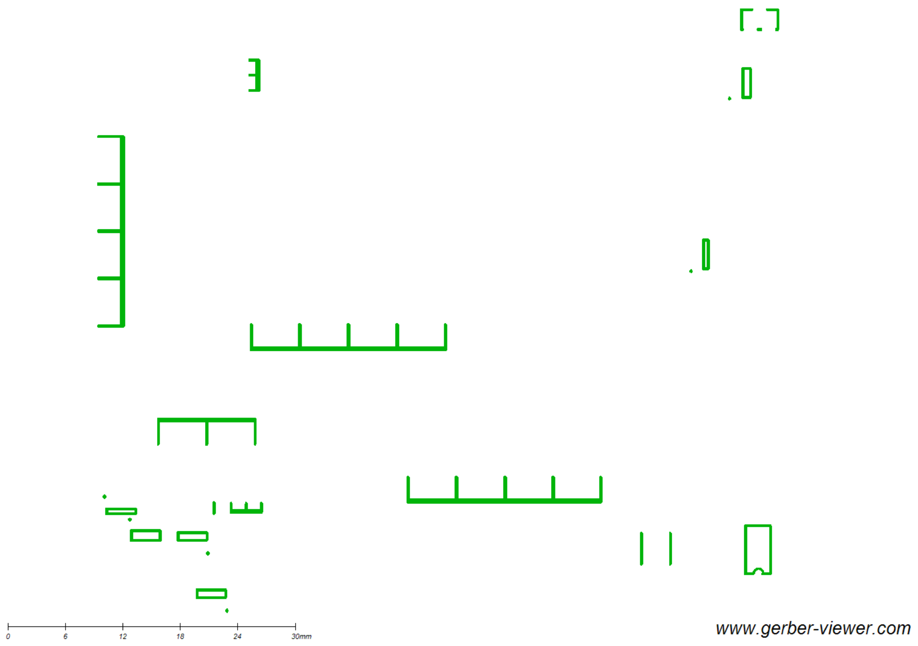

Figure C.15: Bottom silkscreen layer of the MPPT.

0
6
12
18
24
30mm
www.gerber-viewer.com

Figure C.16: Board outline of the MPPT.

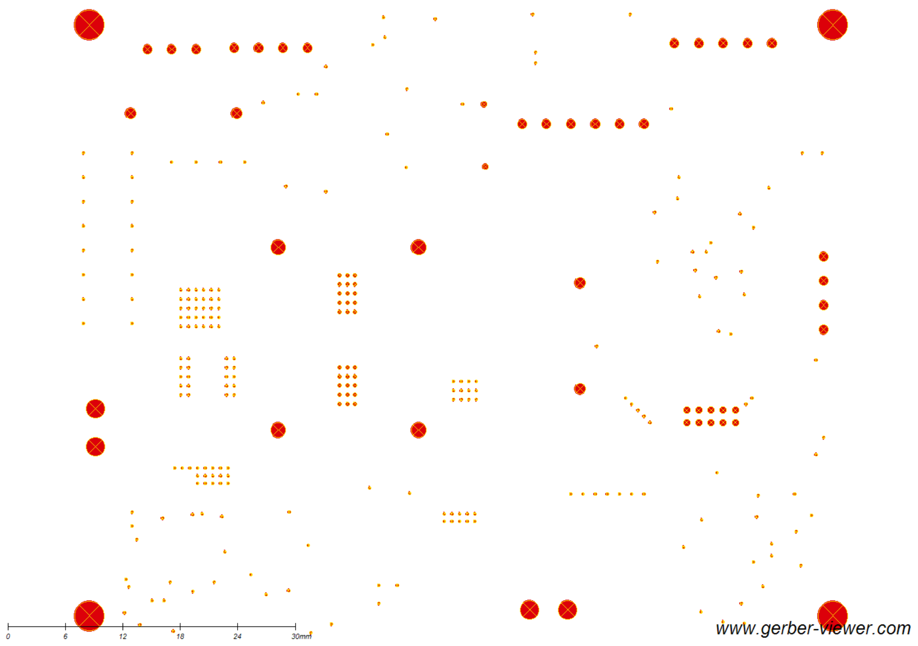

Figure C.17: Drill layer of the MPPT.

C.3 Microcontroller Code

C.3.1 Main

```
/**
 * MPPT Code
 *
 * Author: Derek Chou
 * February 2016
 *
 */

#include <mbed.h>
#include <pins.h>
#include <peripherals.h>
#include <can_id.h>
#include <can_struct.h>
#include <can_buffer.h>
#include <ADCMath.h>
#include <sct0.h>

/** Number of samples that we are going to take per ADC reading. */
const int SAMPLE_SIZE = 4;

/** Time between heartbeat CAN messages. */
const int HEARTBEAT_THRES_US = 1000000;

/** Time between checking CAN controller aliveness. */
const int CAN_CHECK_PERIOD_US = 1000000;

/** Time between sending CAN messages. */
const int SEND_THRES_US = 500000;

/** Time between reading sensors. */
const int READ_THRES_US = 500;

/** IV curve-trace threshold in microseconds - 120 seconds. */
const int TRACE_THRES_US = 30000000;

/** Off-time threshold in microseconds - 3 seconds. */
const int OFF_THRES_US = 3000000;

/** Time elapsed before stepping the duty ratio, in us. */
const int STEP_THRES_US = 10000;

/** Maximum duty ratio that we are comfortable with running at. */
const float MAX_DUTY = 0.95;

/** Minimum duty ratio for PWM. */
const float MIN_DUTY = 0.01;

/**
```

```
 * Maximum voltage that we are comfortable with running at; this is
 * on the battery output.
 */
const float MAX_VOLTAGE = 120;

/**
 * Drop the maximum voltage down to 115V before switching back into
 * the MPPT mode.
 */
const float VOLT_OFFSET = 5;

/**
 * Turn off switches if the current goes negative, into the solar
 * array.
 */
const float NEG_CURR_THRES = -0.1;

/** Last-seen power output. */
float last_power = 0;

/** Last-seen duty ratio. */
float last_duty = 0;

enum BoardState {
    /** Start in this state; no PWM allowed. */
    STATE_OFF,

    /** Startup state - read sensors, etc. No PWM allowed. */
    STATE_STARTUP,

    /** Tracing I-V curve. */
    STATE_SWEEP,

    /** Normal MPPT operations. */
    STATE_MPPT,

    /** Limit maximum voltage, no MPPT. */
    STATE_VOLT_TRACKING,

    /** Problem; fault state for transition. */
    STATE_FAULT
};

/** Different kinds of faults can occur with the MPPT. */
enum faultType {
    INPUT_DISCONNECTED, OUTPUT_DISCONNECTED, OV, OT
};

/** Mux for sensors. */
enum readType {
    VOLTAGE = false, CURRENT = true
};

/**
```

```
 * The ID of this MPPT - the board uses DIP switches to figure out its ID
     for
 * messages.
 */
int mppt_id = 0;

/** Message buffer. */
CANRXTXBuffer<64, 64> canBuffer(can);

/** CAN RX/TX IRQ handler */
void handleCANMessage() {
    canBuffer.handleIrq();
}

/** Persistent values for past readings of voltage and current. */
float lastBatteryVoltage, lastBatteryCurrent, lastSolarVoltage,
        lastSolarCurrent, lastBatteryPower, lastSolarPower;

/** Persistent temperature readings. */
float tempReadings[] = { 0, 0, 0 };

/**
 * Sends a heartbeat over CAN.
 */
void sendHeartbeatMessage() {
    CANMessage msg = makeMessage(CAN_HEART_MPPT(mppt_id), timer.read_us());
    canBuffer.write(msg);
}

/**
 * Check to see whether the CAN Controller is still alive or not.
 */
void checkCANController() {
    // If CNTL is 1, then we have a problem. Set it to 0 to reset the
        controller.
    if (LPC_C_CAN0->CANCNTL & (1 << 0)) {
        LPC_C_CAN0->CANCNTL &= ~(1 << 0);
    }
}

/**
 * Read the three thermistors attached to the board.
 */
float readThermistor(int i) {
    switch (i) {
    case 0:
        return getTemp(Therm0.read_u16());
    case 1:
        return getTemp(Therm1.read_u16());
    case 2:
        return getTemp(Therm2.read_u16());
    default:
        return 0;
    }
```

```
}

/**
 * Read the solar side sensors.
 * @param b Voltage = true, Current = false
 */
float readSolar(readType b) {
    uint16_t data;
    uint32_t dataAggregate = 0;
    for (int i = 0; i < SAMPLE_SIZE; i++) {
        solarADC.requestConversion(data);
        dataAggregate += data;
    }
    dataAggregate /= SAMPLE_SIZE;
    return b ?
            convertSolarCurrent(dataAggregate) :
            convertSolarVoltage(dataAggregate);
}

/**
 * Read the battery side sensors.
 * @param b Voltage = true, Current = false
 */
float readBattery(readType b) {
    uint16_t data;
    uint32_t dataAggregate = 0;
    for (int i = 0; i < SAMPLE_SIZE; i++) {
        batteryADC.requestConversion(data);
        dataAggregate += data;
    }
    dataAggregate /= SAMPLE_SIZE;
    return b ?
            convertBatteryCurrent(dataAggregate) :
            convertBatteryVoltage(dataAggregate);
}

/** Set default values of pins and other setup functions. */
void setup() {
    can.frequency(CAN_FREQUENCY);
    can.attach(handleCANMessage, CAN::RxIrq);
    can.attach(handleCANMessage, CAN::TxIrq);
    LED1 = LED2 = LED3 = LED4 = 0;
    /* Pin Assign 8 bit Configuration */
    /* SCT0_OUT0 */
    /* SCT0_OUT1 */
    /* We'll get a more general implementation in the future. But for now...
         */
    // LPC_SWM->PINASSIGN[7] = 0xff0a12ffUL;

    // Multilevel
    /* Pin Assign 8 bit Configuration */
    /* SCT0_OUT0 */
    /* SCT0_OUT1 */
    /* SCT0_OUT2 */
    LPC_SWM->PINASSIGN[7] = 0x010f0effUL;

    /* Pin Assign 1 bit Configuration */
    LPC_SWM->PINENABLE0 = 0xffffffffUL;
    /* SCT0_OUT3 */
    /* RESET */
    /* SWCLK */
    /* SWDIO */
    LPC_SWM->PINENABLE1 = 0xff1fffdfUL;

    /**
     * PWMBL ::  PWMTH
     * P0_0  ::  P0_15
     * OUT3  ::  OUT1
     *
     * PWMBH ::  PWMTL
     * P0_1  ::  P0_14
     * OUT2  ::  OUT0
     */


}

/**
 * Obtains all of the voltages and currents from the solar input and the
 * battery output.
 */
void readSensors(float& battV, float& solarV, float& battC, float& solarC) {
    solarV = readSolar(VOLTAGE);
    VC_Solar = CURRENT;
    battV = readBattery(VOLTAGE);
    VC_Battery = CURRENT;
    solarC = readSolar(CURRENT);
    VC_Solar = VOLTAGE;
    battC = readBattery(CURRENT);
    VC_Battery = VOLTAGE;
}

/**
 * Simple precharge method.
 */
void precharge() {
    SCT0_setMode(converterMode::ASYNCHRONOUS);
    SCT0_prechargeSequence();
    wait_ms(1000);
    // LED4 is the precharge toggle currently
    LED4 = 1;
    wait_ms(150);
    LED4 = 0;
    SCT0_setMode(converterMode::SYNCHRONOUS);
}

/**
```

```
 * Startup method for MPPT. If the MPPT has a disconnected output at the
     point
 * of startup, start by calculating the boost ratio to get a floating output
 * voltage. Otherwise, use the maximum power point tracking method which
 * starts at maximum boost and then sweeps downwards.
 */
void startup() {
    float battery_voltage, battery_current, solar_voltage, solar_current;
    battery_voltage = 0;
    battery_current = 0;
    solar_voltage = 0;
    solar_current = 0;
    VC_Solar = VOLTAGE;
    VC_Battery = VOLTAGE;
    readSensors(battery_voltage, solar_voltage, battery_current,
            solar_current);
    if (battery_voltage < solar_voltage) {
        /* Calculate ideal boost ratio to get to 120V output. */
        float boost = 1 - (solar_voltage / battery_voltage);

        for (float i = MIN_DUTY; i < boost; i+= 0.01) {
            SCT0_setDuty(i);
            wait_us(100);
            readSensors(battery_voltage, solar_voltage, battery_current,
                    solar_current);
            if (battery_voltage > MAX_VOLTAGE) {
                SCT0_stopPWM();
                break;
            }
            if (battery_current < NEG_CURR_THRES
                    || solar_current < NEG_CURR_THRES) {
                SCT0_stopPWM();
                break;
            }
        }
    } else {
        for (float i = MAX_DUTY; i > 0.0; i -= 0.01) {
            SCT0_setDuty(i);
            wait_us(100);
            readSensors(battery_voltage, solar_voltage, battery_current,
                    solar_current);
            if (battery_voltage > MAX_VOLTAGE) {
                SCT0_stopPWM();
                break;
            }
            if (battery_current < NEG_CURR_THRES
                    || solar_current < NEG_CURR_THRES) {
                SCT0_stopPWM();
                break;
            }
        }
    }

}
```

```
/**
 * Finds the maximum power point by doing an IV curve sweep.
 */
float max_power() {
    float solar_power, battery_power, battery_voltage, battery_current,
            solar_voltage, solar_current, efficiency, max_power, duty;
    solar_power = battery_power = battery_voltage = battery_current =
            solar_voltage = solar_current = efficiency = max_power = duty =
                0;
    VC_Solar = VOLTAGE;
    VC_Battery = VOLTAGE;
    for (float i = MAX_DUTY; i > 0.0; i -= 0.01) {
        timer.reset();
        timer.start();
        SCT0_setDuty(i);
        uint16_t start = timer.read_us();
        uint16_t now = start;
        while (now - start < STEP_THRES_US) {
            now = timer.read_us();
            readSensors(battery_voltage, solar_voltage, battery_current,
                    solar_current);
            solar_power = solar_voltage * solar_current;
            battery_power = battery_voltage * battery_current;
            efficiency = battery_power / solar_power;
            if (battery_voltage > MAX_VOLTAGE) {
                break;
            }
            if (solar_power >= max_power) {
                max_power = solar_power;
                duty = i;
            }
            if (battery_current < -0.1 || solar_current < -0.1) {
                SCT0_stopPWM();
                break;
            }
        }
    }
    last_power = max_power;
    last_duty = duty;
    return duty;
}

/** The classic perturb-and-observe algorithm for MPPT. */
void perturbAndObserve() {
    float kp = 0.001;
    float solar_power, battery_power, battery_voltage, battery_current,
            solar_voltage, solar_current, efficiency;
    solar_power = battery_power = battery_voltage = battery_current =
            solar_voltage = solar_current = efficiency = 0;
    VC_Solar = VOLTAGE;
    VC_Battery = VOLTAGE;
    readSensors(battery_voltage, solar_voltage, battery_current,
        solar_current);
    solar_power = solar_voltage * solar_current;
    battery_power = battery_voltage * battery_current;
    efficiency = battery_power / solar_power;
    float newDuty = last_duty + kp*(solar_power - last_power);
    last_power = solar_power;
    if (newDuty > MAX_DUTY) {
        newDuty = MAX_DUTY;
    }
    if (newDuty < MIN_DUTY) {
        newDuty = MIN_DUTY;
    }
    last_duty = newDuty;
    SCT0_setDuty(newDuty);
}

/**
 * State transition logic.
 * Given a previous state and the amount of time we've spent in the current
   state, do a transition
 * between states.
 * @param prevState The previous state that we are transitioning from.
 * @param stateTime The time that we've spent in the current state.
 */
BoardState doTransition(BoardState prevState, int stateTime) {
    switch (prevState) {
    case STATE_STARTUP:
        return STATE_SWEEP;
    case STATE_MPPT:
        if (stateTime > TRACE_THRES_US) {
            return STATE_SWEEP;
        } else {
            return STATE_MPPT;
        }
    case STATE_SWEEP:
        return STATE_MPPT;
    case STATE_VOLT_TRACKING:
        if (lastBatteryVoltage < MAX_VOLTAGE - VOLT_OFFSET) {
            return STATE_MPPT;
        } else {
            return STATE_VOLT_TRACKING;
        }
    case STATE_FAULT:
        return STATE_OFF;
    case STATE_OFF:
        if (stateTime > OFF_THRES_US) {
            return STATE_STARTUP;
        } else {
            return STATE_STARTUP;
        }
    default:
        return STATE_OFF;
    }
    return prevState;
}
```

```
/**
 * State action logic. Given the current state, do something.
 * @param currState The current state that we're in.
 * @param prevState The previous state, which we were in.
 */
void stateAction(BoardState currState, BoardState prevState) {
    switch (currState) {
    case STATE_STARTUP:
        startup();
        break;
    case STATE_MPPT:
        perturbAndObserve();
        break;
    case STATE_SWEEP:
        SCT0_setDuty(max_power());
        break;
    case STATE_FAULT:
        SCT0_stopPWM();
        break;
    case STATE_OFF:
        SCT0_stopPWM();
        break;
    default:
        SCT0_stopPWM();
        break;
    }
}

int main() {
    BoardState state = STATE_OFF;
    SCT0_Init();
    SCT0_stopPWM();
    setup();
    LED4 = 0;
    timer.start();
    PRGM.mode(PullUp);

    int old_time = timer.read_us();
    int last_heartbeat, last_check, last_read, last_state_time;
    last_read = last_heartbeat = last_check = last_state_time = old_time;
    CANMessage msg;

    while (true) {
        int now = timer.read_us();

        if (now < old_time) {
            last_heartbeat = last_check = last_read = last_state_time = now;
            old_time = now;
            continue;
        }

        if ((now - last_read) > READ_THRES_US) {
            last_read = now;
        }

        if ((now - last_heartbeat) > HEARTBEAT_THRES_US) {
            last_heartbeat = now;
            sendHeartbeatMessage();
        }

        while (canBuffer.read(msg)) {
            // Clear the buffer.
        }

        BoardState prev_state = state;
        state = doTransition(state, now - last_state_time);
        last_state_time = now;
        stateAction(state, prev_state);

        // Check if the CAN controller is alive or not. If it isn't, reset
            the controller.
        if ((now - last_check) > CAN_CHECK_PERIOD_US) {
            last_check = now;
            checkCANController();
        }
    }
}
```

C.3.2 PWM generation

```
/**
 * SCT0 driver for PWMs.
 * Adapted from the application note that was given.
 */

#include <mbed.h>
#include <LPC15xx.h>
#include <sct0.h>

#define     EN1_SCT0        (1<<2)

#define DC1        (1)                                  // duty cycle 1
#define DC2        (11)                                 // duty cycle 2
#define hperiod    (300)                                // 1 / 36000000 *
    300 = 120kHz
#define deadTime (1)                                    // 2 counts = 40ns
    deadtime
#define SCTPLLCTRL_Val        0x00000007                // SCT clock



void SCT0_Init(void) {
    LPC_SYSCON->PDRUNCFG      |= (1 << 24);             // Power down SCT
        PLL
    LPC_SYSCON->SCTPLLCTRL     = SCTPLLCTRL_Val;        // Change rate to
        custom value
```

```
LPC_SYSCON->PDRUNCFG      &= ~(1 << 24);              // Power up SCT PLL
while (!(LPC_SYSCON->SCTPLLSTAT   & 0x01));          // Wait Until PLL
    Locked

LPC_SYSCON->SYSAHBCLKCTRL1 |= EN1_SCT0;              // enable the SCT0
    clock
LPC_SCT0->CONFIG |= (1 << 17);                       // split timers,
    auto limit
LPC_SCT0->CTRL |= (1 << 4);                          // configure SCT0 as
     BIDIR

LPC_SCT0->MATCH0 = hperiod;                          // match on (half)
    PWM period
LPC_SCT0->MATCHREL0 = hperiod;

LPC_SCT0->EV0_STATE = 0xFFFFFFFF;                    // event 0 happens
    in all states
LPC_SCT0->EV0_CTRL = (1 << 0) | (1 << 12);           // match 1 (DC1)
    only condition

LPC_SCT0->EV1_STATE = 0xFFFFFFFF;                    // event 1 happens
    in all states
LPC_SCT0->EV1_CTRL = (2 << 0) | (1 << 12);           // match 2 (DC2)
    only condition

LPC_SCT0->EV2_STATE = 0xFFFFFFFF;                    // event 0 happens
    in all states
LPC_SCT0->EV2_CTRL = (3 << 0) | (1 << 12);           // match 3 (DC1)
    only condition

LPC_SCT0->EV3_STATE = 0xFFFFFFFF;                    // event 1 happens
    in all states
LPC_SCT0->EV3_CTRL = (4 << 0) | (1 << 12);           // match 4 (DC2)
    only condition


LPC_SCT0->OUT0_SET = (1 << 0);                       // event 0 sets OUT0
LPC_SCT0->OUT0_CLR = (1 << 0);                       // event 0 clears
    OUT0

LPC_SCT0->OUT1_SET = (1 << 1);                       // event 1 sets OUT1
LPC_SCT0->OUT1_CLR = (1 << 1);                       // event 1 clears
    OUT1

LPC_SCT0->OUT2_SET = (1 << 2);                       // event 2 sets OUT2
LPC_SCT0->OUT2_CLR = (1 << 2);                       // event 2 clears
    OUT2

LPC_SCT0->OUT3_SET = (1 << 3);                       // event 3 sets OUT3
LPC_SCT0->OUT3_CLR = (1 << 3);                       // event 3 clears
    OUT3

LPC_SCT0->RES |= 0x000000FF;                         // toggle OUT0 and
    OUT1 on conflict
    NVIC_EnableIRQ(SCT0_IRQn);                          // enable SCT0
        interrupt

}

/**
 * PWMBL  ::  PWMTH
 * P0_0   ::  P0_15
 * OUT3   ::  OUT1
 * 1B     ::  1A
 * MATCH4 ::  MATCH2
 *
 * PWMBH  ::  PWMTL
 * P0_1   ::  P0_14
 * OUT2   ::  OUT0
 * 2B     ::  2A
 * MATCH3 ::  MATCH1
 *
 */

void SCT0_setDuty(float duty) {
    int newDuty1 = (int) ((duty) * hperiod);
    if (newDuty1 <= 0) {
        newDuty1 = 1;
    }
    if (newDuty1 >= hperiod) {
        newDuty1 = hperiod - 1;
    }
    LPC_SCT0->CONFIG |= (1 << 7);        // Temporarily stop reload

    /** 1B */
    LPC_SCT0->MATCH4 = newDuty1;
    LPC_SCT0->MATCHREL4 = newDuty1;

    /** 1A */
    LPC_SCT0->MATCH2 = newDuty1 + deadTime;
    LPC_SCT0->MATCHREL2 = newDuty1 + deadTime;

    /** 2B */
    LPC_SCT0->MATCH3 = hperiod - newDuty1;
    LPC_SCT0->MATCHREL3 = hperiod - newDuty1;

    /** 2A */
    LPC_SCT0->MATCH1 = hperiod - newDuty1 - deadTime;
    LPC_SCT0->MATCHREL1 = hperiod - newDuty1 - deadTime;



    LPC_SCT0->CTRL &= ~(1 << 2 | 1 << 18);        // Start timer
    LPC_SCT0->CONFIG &= ~(1 << 7 | 1 << 8);       // Start reload
}

/** Stop PWM signals from running. */
```

```
void SCT0_stopPWM() {
    LPC_SCT0->CTRL |= (1 << 2 | 1 << 18);
    LPC_SCT0->CONFIG |= (1 << 7 | 1 << 8);
    LPC_SCT0->OUTPUT &= (0xFFF0);
}

/** Start PWM signals again at duty ratio D. */
void SCT0_startPWM(float duty) {
    LPC_SCT0->OUTPUT |= 0b1001;
    SCT0_setDuty(duty);
}

/** Precharge the flying capacitor to V=Vin. */
void SCT0_prechargeSequence() {
    LPC_SCT0->OUTPUT |= (0b1000);
}

void SCT0_setMode(converterMode m) {
    if (m == ASYNCHRONOUS) {
        LPC_SCT0->EV0_STATE = 0x0;
        LPC_SCT0->EV1_STATE = 0x0;
    } else {
        LPC_SCT0->EV0_STATE = 0xFFFFFFFF;
        LPC_SCT0->EV1_STATE = 0xFFFFFFFF;
    }
}

/** Startup the converter by running in asynchronous mode. */
void SCT0_startupSequence() {
    LPC_SCT0->EV2_STATE = 0x0;
    LPC_SCT0->EV3_STATE = 0x0;
}
```

C.3.3 ADC reading

```
/**
 * ADCMath.h
 * Math that the microcontroller needs to do to interpret the different
 * readings from the MPPT correctly.
 *
 * Author: Derek Chou
 * March 2016
 *
 */


const float R_REF = 10000.0;
const float V_REF = 3.0;
const float T0 = 298.15;
const float BETA = 3428.0;
const float R_INF = R_REF * exp(-BETA / T0);
const float voltsPerBit = (3.000 / 4096.0);
```

```
/** Gets the battery voltage based on the bits read from the ADC. */
float convertBatteryVoltage(uint16_t readingBits) {
    return readingBits / 4096.0 * 3.0 * 480000 / 10000;
}

/** Gets the battery current based on the bits read from the ADC. */
float convertBatteryCurrent(uint16_t readingBits) {
    return (readingBits * 3.0 / 4096.0) / 68 / 0.003;
}

/** Gets the solar voltage based on the bits read from the ADC. */
float convertSolarVoltage(uint16_t readingBits) {
    return readingBits / 4096.0 * 3.0 * 490000 / 20000;
}

/** Gets the solar current based on the bits read from the ADC. */
float convertSolarCurrent(uint16_t readingBits) {
    return (readingBits * 3.0 / 4096.0) / 68 / 0.003;
}

/**
 * @param theReading ADC reading in bits.
 */
float getTemp(uint16_t ADC_Reading = 0) {
    float voltage = 0, resistance = 0, temp = 0;

    // Interpret analog into the voltage:
    voltage = ADC_Reading * voltsPerBit;

    // Use voltage to get resistance
    resistance = (R_REF * V_REF / voltage) - R_REF;

    // Get temperature:
    temp = BETA / log((resistance) / R_INF);

    return temp - 273.15; // This is in celsius
}
```