Zero-Voltage Switching for Flying Capacitor Multi-Level Converters

Margaret Blackwell

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2019-35
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-35.html

May 14, 2019

Zero-Voltage Switching for Flying Capacitor Multi-Level Converters

by

Margaret Elizabeth Blackwell

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Engineering - Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Associate Professor Robert Pilawa-Podgurski, Chair
Professor Seth Sanders
Associate Professor Duncan Callaway

Spring 2019

Zero-Voltage Switching for Flying Capacitor Multi-Level Converters

Abstract

Zero-Voltage Switching for Flying Capacitor Multi-Level Converters

by

Margaret Elizabeth Blackwell

Master of Science in Engineering - Electrical Engineering and Computer Sciences

University of California, Berkeley

Associate Professor Robert Pilawa-Podgurski, Chair

This thesis presents a control technique to improve power density and efficiency of a specific power converter topology, the flying capacitor multi-level (FCML) topology. Controlling these converters in such a way to achieve zero-voltage switching (ZVS) across the full range of duty cycles, reduces switching losses and therefore can be used to allow for more dense designs, or more efficient operation. Previous works have used variable frequency control to enable ZVS at specific duty cycles in FCML converters, but have not been able to use these methods to enable ZVS across the full range. This work uses dynamic level selection and variable frequency control to increase inductor current ripple at duty cycle ranges for which ZVS was previously unattainable. Furthermore, a mathematical analysis to determine parameters for active voltage balancing of the flying capacitors during a dynamic level transition is presented. An experimental 5-level FCML prototype was built using GaN devices on a single-sided printed circuit board (PCB) to demonstrate this control technique. We demonstrate 4-level and 5-level operation with ZVS at duty cycles that are not possible with 5-level operation alone, as well as dynamic level transitioning with active flying capacitor voltage balancing.

To Mom and Dad, for your infinite love and encouragement
and
To Devin, for your incessant love and encouragement

Contents

List of Figures

List of Tables

Acknowledgments

Firstly, I would like to thank the University of Illinois Urbana-Champaign SURGE Fellowship program for funding my first year of graduate school and ARPA-E for funding this work during the second year.

I would like to thank Dr. Prasad Enjeti at Texas A&M University for providing me with the opportunity to jump into power electronics research as an undergraduate student, and for being a proponent of my success even after graduation.

I am also extremely grateful to my research adviser, Dr. Robert Pilawa-Podgurski. Not only did he take a chance on inviting me to join his group, but when a job opportunity arose at UC Berkeley, he extended the offer for me to join him in the move before ever having worked with me. His confidence in me and in my potential has been much needed encouragement over the past two years (and likely will continue to be in the years to come). Thank you for taking the time to be involved with my research, as well as caring about my life outside of the lab.

I want to thank the members of the "Pilawa Research Group" as well: Derek Chou, Nate Pallo, Zichao Ye, Zitao Liao, Chris Barth, Rose Abramson, Yizhe Zhang, Kelly Fernandez, Pourya Assem, Wen-Cheun (Joseph) Liu, Joseph Schaadt, Tom Foulkes, and Pei Han Ng. In addition to helping answer questions, discussing ideas, as well as helping in the lab, they have continually supported my success and have been sincere friends to me. A special "thank you" to both Nathan Brooks, for challenging my thoughts, ideas, and methods, and for helping to think up solutions, and to Dr. Enver Candan for being so genuine with including me in your work from the moment I entered the group, and for taking the time to teach me.

I want to thank Sam Coday, for being by my side and navigating graduate school with me. When I started graduate school, I never expected to make such an amazing friend as Sam and if not for her, I surely would not have even made it through the first semester. Thank you for sharing in my struggles and accomplishments and for motivating me, especially through the writing of this thesis.

The member of my research group I would most like to thank is Andrew Stillwell. Without Andrew, this thesis would not exist. From proposing the idea, to working with me to complete it, he has been my guide and mentor through these first two years of graduate school. Thank you for not only encouraging me and having confidence in my abilities, but letting me know that when I needed to hear it. I want to thank you for teaching me, helping me through my struggles, being excited about my successes, and for being my friend.

Finally, I would like to thank my friends and family, who provide motivation and encouragement. I want to especially thank Mom, Dad, Matthew, Timothy, Andrew, and Sabrina for loving me and not forgetting me even though I am several states away. Thank you for being proud of me; it is what keeps me continuing on in my studies and career. Last, but certainly not least, I want to send a million thanks to Devin. Thank you for your dedication to me and to helping me pursue my dreams. Thank you for loving me with your whole heart and for always being there for me. You know I would be lost without you, so I am eternally grateful that I have found a partner in you.

Chapter 1

Introduction

Power electronics, the field of utilizing switching devices to convert between forms and levels of power, is continually growing and open to crucial advancements. Oak Ridge National Laboratory estimates that by 2030, about 80% of electricity could flow through power electronics [1], either on the side of power generation or consumption. Applications that are heavily dependent on power electronics, such as electrification of transportation, grid integration of renewable energy sources, and data center power delivery are expanding, consequently pushing the advancement of power electronics, specifically in the areas of power density and efficiency [2]. A few potential methods of approaching these challenges are: increasing switching frequency, changing topologies, or targeting and reducing component losses. Increasing switching frequency or changing topologies can allow for reduced component sizing. Reducing specific component losses, such as those associated with magnetic components or switching devices, can increase efficiency, as well as allow for further increase in power density. In this work, we combine each of these methods using a novel control technique with the flying capacitor multi-level (FCML) topology to improve both efficiency and power density.

FCML converters utilize one or more flying capacitors, which are capacitors that are connected to various voltage potentials in the circuit via a network of switching devices. These capacitors in an FCML converter act as energy storage elements to reduce the switch voltage stress of each transistor and to reduce the volt-second on the inductor [3–8]. These benefits allow the use of lower voltage rated switches, which permits higher switching frequencies as a result of lower switching losses. The increase in switching frequency, in conjunction with the reduction in inductor volt-second, due to inherent qualities of the FCML topology, leads to a reduction in the volume of the inductor and the total volume of the converter. However, with this decrease in volume comes a necessity to increase efficiency because the surface area for heat transfer is reduced. Further reduction in volume can be achieved through higher frequency switching at the cost of higher switching losses. To mitigate these switching losses, zero-voltage switching (ZVS) can be employed at selected duty cycles as shown in [9, 10] through variable frequency control. However, both works noted the challenges inherent to FCML operation of obtaining ZVS at specific duty cycles. For DC/AC or AC/DC con-

verter applications, or for applications with wide input voltage ranges, the duty cycle of the switches must vary across a wide range. However, due to the nature of FCML operation detailed in this paper, maintaining ZVS across the full range is a challenge. In [11], the inductor current ripple is *minimized* by dynamically varying the number of levels of the FCML converter, which is suitable for hard-switched operation. Here, we propose to dynamically vary the number of levels to *increase* the inductor current ripple and, in conjunction with variable frequency control, maintain the necessary conditions for ZVS across the full duty cycle range.

We derive the underlying mechanisms in FCML converters which make ZVS difficult or impossible at specific duty cycle ranges, and show how dynamic level selection overcomes this challenge. Additionally, we detail the capacitor voltage considerations necessary to decide the number of converter levels and switch implementation. Our control strategy is validated in hardware through a 5-level experimental prototype, which demonstrates ZVS at duty cycles previously unattainable. Level transitioning is demonstrated with active balancing through the use of duty cycle adjustment. This thesis presents a method of ensuring ZVS operation across a full range of conversion ratios for an FCML converter, and demonstrates this method in a compact and flat hardware prototype [12].

The remainder of this thesis is organized as follows: Chapter 2 reviews the basics of a conventional two-level buck converter, as well as details FCML operation. Chapter 3 describes quasi-square-wave ZVS operation and how this approach applies to FCML converters. Additionally, this chapter derives the fundamental characteristics of FCML converters that prevent ZVS operation at specific duty cycles. Chapter 4 explores current solutions to FCML ZVS challenges and proposes dynamic level selection to overcome these challenges. This chapter also steps through the design process of implementing level transitioning. Chapter 5 describes the active balancing method for level transitions and presents a method to determine parameters for active balancing corresponding to the shortest settling time. Chapter 6 demonstrates the method of dynamic level selection for a wide duty cycle range in hardware, as well as the efficiency benefits of using this method for wide-range ZVS. Finally, Chapter 7 summarizes the contribution of this thesis and proposes future work on this topic.

Chapter 2

Background

Several applications including data center power delivery rely on power electronics to perform voltage step-down processes. Converting energy from the electrical grid at higher voltages to the lower voltages used by various systems (e.g. the servers and individual CPUs within the data center architecture) require highly efficient power converters. In this chapter, we discuss a simple step-down power converter as the basis for an expanded multi-level step-down converter: the flying capacitor multi-level (FCML) topology. We detail FCML operation as well as the advantages of the FCML topology over the simple buck converter.

2.1 Conventional Buck Converter

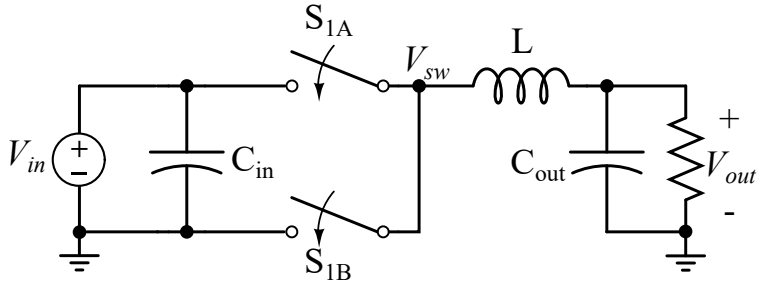


Figure 2.1: Conventional two-level buck converter schematic.

A standard switching power converter for voltage step-down is the buck converter [13]. Fig. 2.1 shows the circuit schematic for the conventional buck converter. The two switches, S_{1A} and S_{1B}, are operated as a complementary pair; that is, when S_{1A} turns on, S_{1B} turns off, and vice-versa. The percentage of time within the switching period T_{sw} that each switch is turned on is the duty cycle, D. For the buck converter, the voltage conversion ratio from the input to the output is equivalent to the duty cycle, as given by (2.1). A method of varying the duty cycle, called pulse width modulation (PWM), can be used to adjust the conversion ratio across a 60/50 Hz AC line cycle for AC/DC and DC/AC conversion.

In the conventional buck converter, the maximum voltage stress (neglecting ringing) across each of the switches is equal to the full input voltage, V_{in}. Furthermore, the voltage across the inductor during the on-time ($D \cdot T_{sw}$) of S_{1A} is $(1 - D) \cdot V_{in}$, and the current ripple found by using the inductor voltage, (2.2), is given by (2.3) where L is the inductance and $f_{sw} = 1/T_{sw}$ is the switching frequency. The voltage swing across the inductor for the buck converter is equal to V_{in}. This voltage swing is the difference in voltage between the highest voltage across the inductor and the lowest voltage across the inductor during one switching period. While the conventional buck converter is relatively simple, a few drawbacks include the large voltage ratings necessary for the switches, as well as the large voltage at the switch-node (V_{sw} in Fig. 2.1) which requires a larger filter inductor, L. Furthermore, the large voltage swing and therefore, large dV/dt transitions at the switch-node can induce voltage overshoots at the switching transistions because of parasitic inductances, as well as can pose a problem for filtering electromagnetic interference (EMI). These limitations can be addressed by investigating other circuit topologies.

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

$$V_L = L\frac{di_L}{dt} \tag{2.2}$$

$$\Delta i_L = \frac{V_{in} \cdot (1 - D) \cdot D}{L \cdot f_{sw}} \tag{2.3}$$

2.2 Flying Capacitor Multi-Level Converter

One potential way to address the limitations of the conventional buck converter is to extend the converter to a multi-level topology, for example, the flying capacitor multi-level (FCML) topology, introduced in [3]. The FCML can be configured to step-up [14–17] or step-down [9, 18–21] the input voltage. or have bi-directional capabilities [10, 22]; here we use the buck configuration. Fig. 2.2 shows a schematic drawing of the 5-level FCML buck converter used in this work with flying capacitors labeled C_1, C_2, and C_3. The voltage conversion ratio of the buck FCML is equivalent to that of the traditional two-level buck converter, given by (2.1) [3]. Phase-shifted PWM (PS-PWM) [3, 18] is typically used for FCML converters of N levels with each switch pair (labeled S_{iA} and S_{iB}) operated complementary to each other at duty cycle, D, and phase shifted by $\Phi = 360°/(N - 1)$. Inherent to the FCML operation are both the converter duty cycle, D, and an effective duty cycle at the switching-node, D_{eff} given by (2.4), which affects the inductor current ripple, Δi_{pp}, given by (2.5). Fig. 2.3a and Fig. 2.3b show the switch control signals for two different duty cycles that generate the same effective duty cycle at the switch-node. At these two duty cycles, the switch-node voltage swing remains the same in magnitude, but the absolute voltage levels are shifted. This voltage shift is a characteristic of the multi-level nature of the FCML topology.

$$D_{eff} = D \cdot (N - 1) - floor(D \cdot (N - 1)) \tag{2.4}$$

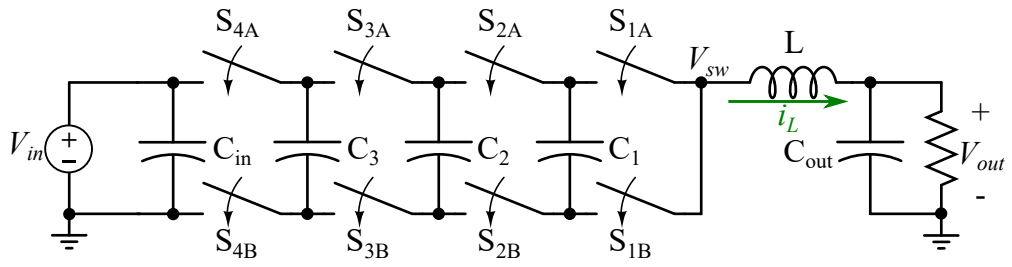
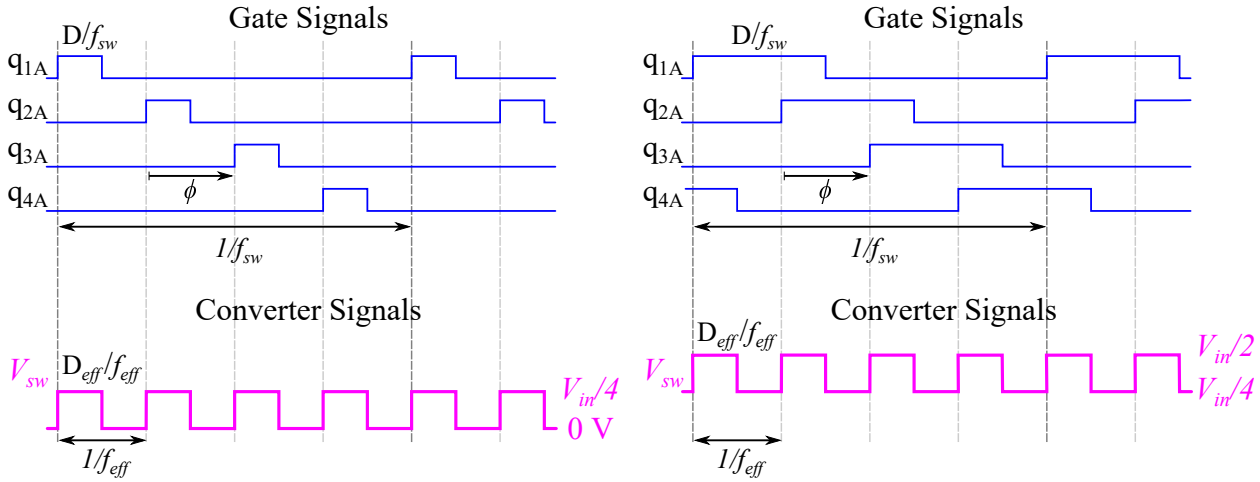


Figure 2.2: 5-Level FCML Converter Schematic.



(a) For lower duty cycles (0 - 25%), the switch-node voltage in a 5-level FCML alternates between 0 V and $\frac{1}{4}$ of the input voltage. Here, a 12.5% duty cycle at a switching frequency, f_{sw}, yields a 50% effective duty cycle and an effective switching frequency $f_{eff} = 3 \cdot f_{sw}$ at the switch node.

(b) For a range of duty cycles higher than those in Fig. 2.3a (e.g. 25 - 50%), the switch-node voltage in a 5-level FCML alternates between $\frac{1}{4}$ and $\frac{1}{2}$ of the input voltage. Here, a 37.5% duty cycle at a switching frequency, f_{sw}, yields a 50% effective duty cycle and an effective switching frequency $f_{eff} = 3 \cdot f_{sw}$ at the switch node.

Figure 2.3: Five-level FCML PS-PWM control signals at different duty cycles and the associated switch-node voltage exhibiting the effective duty cycle and reduced inductor voltage swing.

$$\Delta i_{pp} = \frac{V_{in} \cdot (D_{eff} \cdot (1 - D_{eff}))}{L \cdot f_{sw} \cdot (N-1)^2} \tag{2.5}$$

One advantage of the FCML converter with PS-PWM control is the reduced switch voltage stress, $V_{in}/(N-1)$, because the flying capacitors, C_k, are held at a steady-state voltage, (2.6). The capacitors that flank each switch are separated by only a fraction of the input voltage based on the number of levels. This voltage differential is the voltage that the switch must be rated to block (neglecting margins for overshoot/ringing). Because of the reduced voltage requirement, higher power density converters can be designed by using smaller transistors [6, 18].

$$V_{C_k} = \frac{k \cdot V_{in}}{(N-1)}, k = 1, 2...(N-2) \tag{2.6}$$

Additionally, the voltage across the inductor swings by only $V_{in}/(N-1)$ as compared to the conventional buck inductor which swings by the full input voltage. Fig. 2.4 shows this reduced voltage step on the inductor and demonstrates the multi-level structure of this topology as evident by the number of discrete voltage levels at the inductor. Across a period of changing duty cycles, the switch-node voltage alternates between different voltage levels that are determined by the level number, N, of the converter, but experiences a constant voltage swing, reduced from that of the two-level buck converter. Furthermore, the FCML topology has an inherent frequency multiplication at the switch-node, V_{sw} in Fig. 2.2, that allows for a reduction in filter inductance. For a given switching frequency, f_{sw}, the effective switching frequency, f_{eff}, seen at the inductor is $(N-1) \cdot f_{sw}$, shown in Fig. 2.3. Both the frequency multiplication and voltage reduction lead to a required inductance decrease by $(N-1)^2$. A reduction in the passive component requirements can allow for converters of higher power density. However, decreased passive component volume is only one aspect to address when designing high efficiency, high density power converters. The following chapter will explore another technique that can be used to reduce converter power losses and increase efficiency.

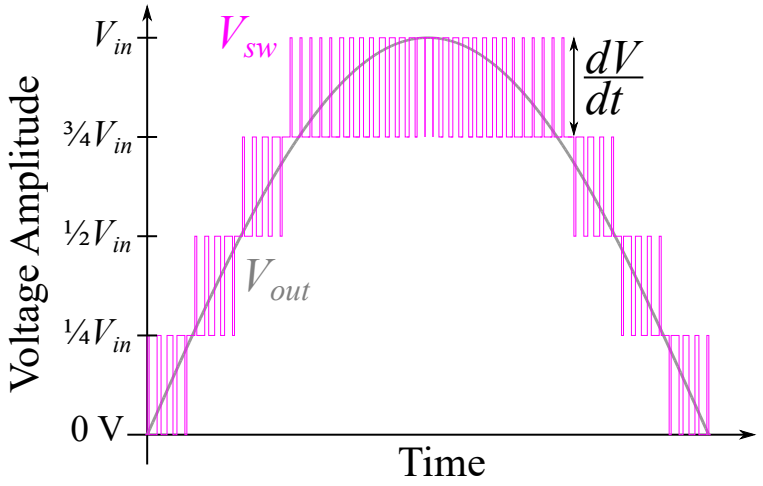


Figure 2.4: Multiple discrete voltage levels can be generated at the switch-node with an FCML converter, in this example with a 5-level FCML, thereby reducing the dV/dt transitions at the switch-node and across the inductor, allowing for a reduction in filter requirements.

Chapter 3

Zero-Voltage Switching

One approach to increase converter power density is to reduce the size of the passive components (capacitors and inductors). Chapter 2 describes how the inductor size can be reduced by using a multi-level topology; in this chapter, we discuss a control technique that can be used to reduce converter losses. Loss reduction can be used to increase efficiency in the same converter volume or allow the same efficiency in a smaller volume by reducing passive components further.

By rearranging (2.3), we can see that either increasing the allowed current ripple or increasing the switching frequency can lead to a smaller inductance requirement. However, increasing the peak-to-peak current ripple on the inductor increases inductor core losses and AC conduction losses. Similarly, increasing the switching frequency increases the losses associated with switching. Although, through control techniques, such as implementing soft-switching, we can reduce switching losses and allow faster switching without incurring excessive penalties.

Zero-voltage switching (ZVS) is one method of achieving soft-switching conditions by switching when the voltage across the transistor is zero [13,23]. ZVS can be realized through resonant operation [24–26] or by using quasi-square-wave (QSW) control [9, 23, 27–29]. The fundamental operation of QSW ZVS is described in Section 3.1, as well as design and control considerations for ZVS. Sections 3.2 and 3.3 detail how QSW ZVS can be applied to the FCML topology and the challenges that arise in maintaining ZVS across wide operating conditions.

3.1 Quasi-Square-Wave Zero-Voltage Switching

The quasi-square-wave (QSW) control method entails adjusting the on-times and dead-times (time when neither switch is on) of the transistors such that the inductor current charges/discharges the parasitic capacitances of the transistor, and allows a zero-voltage switch transition [23, 27]. If either the voltage across the transistor or the current through the transistor is zero, then the power loss ($P = I \cdot V$) is zero. In this case, we control the

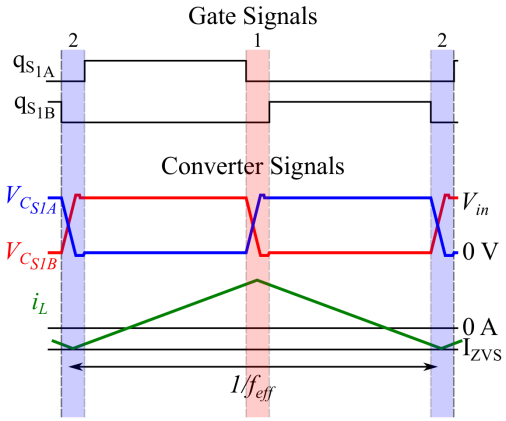
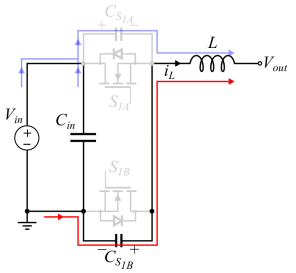
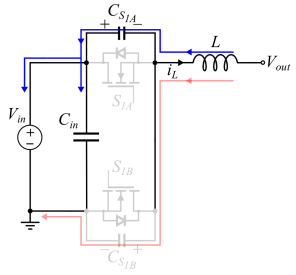


Figure 3.1: Two-level buck converter waveforms for ZVS conditions. The inductor current must have enough ripple to reach a peak negative value, I_{ZVS}, which can discharge the parasitic capacitance $C_{S_{1A}}$ of the buck switch-pair and allow for ZVS.



(a) Positive inductor current in Region 1 of Fig. 3.1 discharges the parasitic capacitance of the low-side switch and allows ZVS.



(b) Negative inductor current is needed in Region 2 of Fig. 3.1 to discharge the parasitic capacitance of the high-side switch and allow ZVS.

Figure 3.2: Two-level buck switch-pair with parasitic capacitances.

switch voltage to be zero before the transition.

The QSW ZVS approach is described here, first using a two-level buck converter, and then extended to a multi-level buck (the FCML topology). Fig. 3.1 shows the control signals and drain-source voltages, V_{ds}, for the switch-pair in a two-level buck converter, shown in Fig. 2.1. The switch-pair consists of a high-side switch, which is closer in potential to the input than the low-side switch, which is closer to the ground connection. The voltage, V_{ds}, is equivalent to the voltage across the parasitic output capacitance of the switch, $V_{C_{S1A/B}}$. For QSW ZVS, a sufficiently long deadtime (Regions 1 and 2 in Fig. 3.1) is imposed based on the inductor current and parasitic capacitances. Only the switching losses for the switch turn-on transition are considered because when the switch is on and about to turn off, there is no voltage present across the switch. There is, however, voltage overshoot during the turn-off transition due to stray inductances, but the losses due to this transition are neglected because they are much smaller in magnitude than losses due to the turn-on transition [30]. In buck-mode operation, because the inductor current is naturally positive during the deadtime before the low-side switch, S_{1B}, turns on (Region 1 in Fig. 3.1), ZVS is easily attainable for the low-side switches. The small parasitic capacitance of the low-side transistor, $C_{S_{1B}}$ in

Fig. 3.2a, can discharge quickly from $V_{C_{S1B}} = V_{in}$ to 0 V with this positive current, i_L, thus enabling a zero-voltage at the time of switching. Once the parasitic capacitance has fully discharged, the body diode (or body diode-like mechanism in GaN transistors) of the switch becomes forward biased and begins to conduct, contributing a small diode voltage drop shown in Fig. 3.1. Therefore, the deadtime should be designed to be sufficiently long to allow the switch capacitance to fully discharge, but short enough to minimize the length of the body diode conduction time.

Conversely, ZVS for the high-side switch, S_{1A}, is more difficult because a negative current during the deadtime of Region 2 is required to discharge the parasitic capacitance, $C_{S_{1A}}$, to 0 V before switching, shown in Fig. 3.2b. The ZVS mechanism for the high-side switch is the same as that for the low-side switch, however it requires the inductor current to be in the opposite direction. Having a negative inductor current for a portion of the switching cycle requires a sufficiently large inductor current ripple. This ripple may need to be rather large as the average output current of the converter increases. It is possible in some cases to design the inductor and control the switching frequency such that the inductor current does go negative. This control method and its shortcomings in some applications are discussed below.

The deadtimes set for Region 1 and Region 2 are determined based on the current at the switch turn-on and turn-off transitions in these regions, and the total effective capacitance and voltage that needs to be charged or discharged. In [31], the total charge equivalent capacitance is found by (3.1) and is used to determine the minimum amount of negative current, I_{ZVS}, needed to discharge the parasitic switch capacitances. The amount of energy stored in the inductor at the time of the transition must be sufficient to discharge the parasitic switch capacitance from its full V_{ds} voltage, which in the case of the two-level buck is V_{in}, to zero volts. By comparing the energy stored in the inductor with the energy needed to be discharged from the total charge equivalent parasitic capacitance (3.2), the minimum negative current peak needed for ZVS can be found by (3.3) [31]. For the two-level buck converter, the total equivalent capacitance is the parallel combination of two switch output capacitors since while one capacitor is discharging, the other is charging in parallel, as seen in Fig. 3.2. The time needed to discharge the equivalent capacitance is found using an analysis of resonance between the inductor and switch capacitance [32, 33].

$$C_{eqv,ch} = \frac{\int_0^{V_{ds}} C_{oss}(v)dv}{V_{ds}} \tag{3.1}$$

$$\frac{1}{2} L I_{ZVS} \geq C_{tot} V_{ds}^2 \tag{3.2}$$

$$I_{ZVS} = \sqrt{\frac{2 \cdot (C_{tot}) V_{ds}^2}{L}} \tag{3.3a}$$

$$C_{tot} = 2 \cdot C_{eqv,ch} \tag{3.3b}$$

The minimum deadtime for ZVS conditions can be calculated by (3.4) and (3.5) [30, 34]. The solution to the second-order differential equation that describes the current waveshape during the deadtime is dependent on the initial voltage across the inductor. The initial inductor voltage, $V_{L_{ZVS}}$, differs for the high-side and low-side switches based on how the switches connected the circuit components before the ZVS deadtime and is given by (3.5). The deadtime should be designed to minimize body diode conduction as mentioned above since this unnecessary diode conduction leads to power loss and lower efficiency. Furthermore, selection of the switches should take into account the equivalent output capacitance of the switches. Larger parasitic capacitance requires more energy to discharge, which means either a longer deadtime, which can hurt efficiency, or a larger negative inductor current/ larger inductor current ripple, which consequently leads to increased inductor core losses and AC conduction losses.

$$t_d \geq \frac{1}{\omega_0}(tan^{-1}(\frac{V_{L_{ZVS}}}{Z_0 \cdot I_{ZVS}}) + \frac{\pi}{2}) \tag{3.4a}$$

$$\omega_0 = \sqrt{L \cdot C_{tot}} \tag{3.4b}$$

$$Z_0 = \sqrt{\frac{L}{C_{tot}}} \tag{3.4c}$$

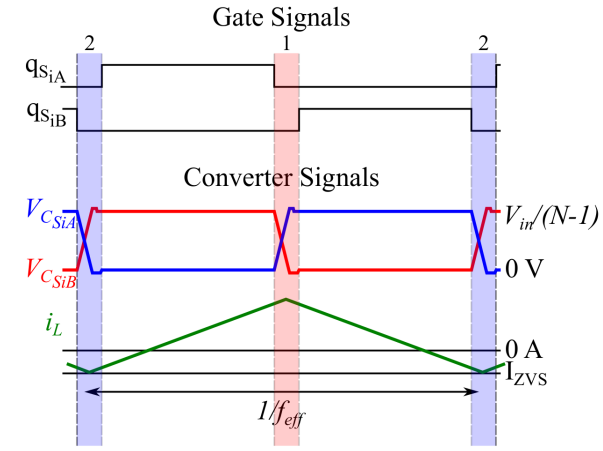
$$\tag{3.4d}$$

$$V_{L_{ZVS}} = \begin{cases} -D \cdot V_{in}, & \text{for } S_{1_A} \\ (1-D) \cdot V_{in}, & \text{for } S_{1_B} \end{cases} \tag{3.5}$$

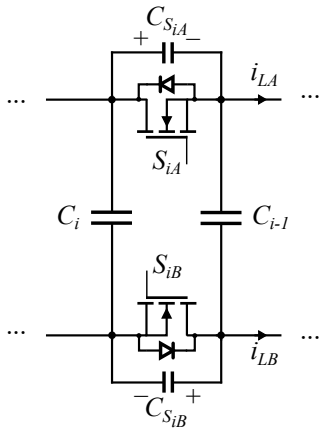
3.2 QSW ZVS for FCML Converters

The above quasi-square-wave technique can be applied to the phase-shifted PWM (PS-PWM) control scheme typically used with FCML converters. However, there are a few differences between QSW ZVS for a two-level topology and for a multi-level topology. As seen in Fig. 2.2, there are now several switch-pairs that need ZVS. Because PS-PWM is utilized, the parasitic switch capacitances do not necessarily charge/discharge directly through the input source, instead there may be flying capacitors through which the inductor current also flows. However, because only one pair, Fig. 3.3b, transitions at a time and the two switches in that pair are complementary, only the commutation loop between the two switches in a single pair affects ZVS operation. Furthermore, the voltage needed to be discharged from the parasitic capacitances is reduced from the full input voltage (for a two-level buck) to only a fraction, of the input voltage (for the FCML buck), shown in Fig. 3.3a, due to the flying capacitors adjacent to the switch-pair having fractional voltages of the input voltage.

Because FCML ZVS functions similarly to ZVS in the two-level buck converter [30, 34], this deadtime minimum is given by the same (3.4). However, the initial voltage across the



(a) FCML converter waveforms for ZVS conditions.



(b) Arbitrary FCML switch pair with parasitic capacitances.

Figure 3.3: Inductor current must have enough ripple to reach a peak negative value, I_{ZVS} which can discharge the parasitic capacitance $C_{S_{iA}}$ of an arbitrary switch pair and allow for ZVS.

inductor, $V_{L_{ZVS}}$, varies with the switching pattern, so for the high-side and low-side switches, the initial inductor voltage is different, given by (3.6). The inductor voltage also changes as the duty cycle changes and the multi-level characteristics become evident. The initial inductor voltage equation for FCML converters is equivalent to that of the conventional buck converter (3.5) when $N = 2$.

$$V_{L_{ZVS}} = \begin{cases} \dfrac{V_{in}}{(N-1)} \cdot floor(D(N-1)) - D \cdot V_{in}, & \text{for } S_{i_A} \\ \dfrac{V_{in}}{(N-1)} \cdot ceil(D(N-1)) - D \cdot V_{in}, & \text{for } S_{i_B} \end{cases} \tag{3.6}$$

3.3 ZVS Challenges for FCML Converters

Previous works, [23] and [28], have shown that a sufficiently large inductor current ripple is required to provide a negative current, i_L, during a specified deadtime which discharges the transistor parasitic capacitance and allows ZVS operation. However, due to the multi-level operation of the FCML, certain duty cycles inherently exhibit low or no current ripple, inhibiting the ability to achieve ZVS without going to extremely low switching frequencies.

Revisiting (2.4), it is apparent that D_{eff} is zero for certain values of D (when $D \cdot (N-1)$ is an integer value) and therefore, the inductor current ripple approaches zero as well, (2.5). Fig. 3.4 shows the inductor current ripple at a fixed switching frequency, f_{sw}, for a 4- and 5-level FCML normalized to the conventional two-level buck converter, with current ripple

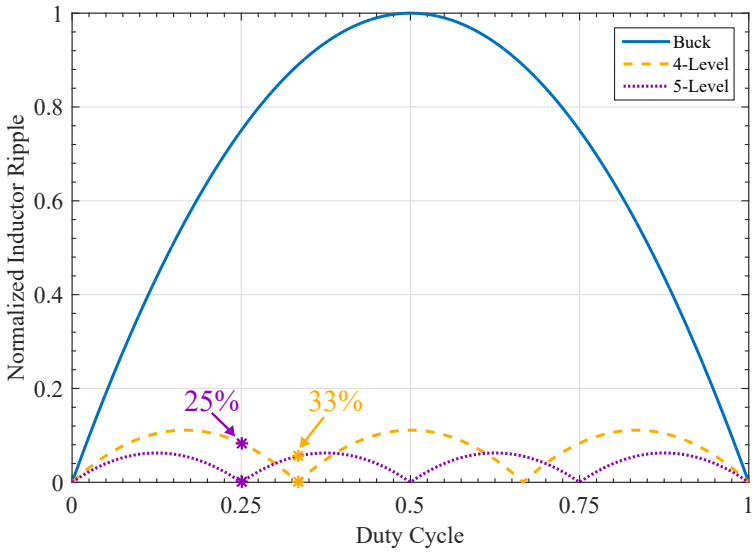


Figure 3.4: Higher-level FCML converters inherently exhibit lower inductor current than two-level buck converters, but introduce inductor ripple valleys at certain duty cycles.

valleys at duty cycles of 0.33 and 0.66 for the 4-level FCML, and 0.25, 0.5, and 0.75 for the 5-level FCML. Compared to the two-level buck converter, one advantage of the FCML is evident by the reduced magnitude of inductor current ripple. For the same inductor, a reduced inductor current ripple reduces the core losses and AC conduction losses of the inductor. However, this reduction in inductor current ripple poses a challenge for maintaining ZVS conditions, which as discussed above requires a sufficiently large current ripple.

Previous works [9], [10] have shown that by varying the switching frequency along the duty cycle range, the inductor current ripple can be changed to keep ZVS operation. However, the switching frequency can only be decreased to limits imposed by the flying capacitor voltage ripple, inductor saturation, or practical limitations [10]. These switching frequency limitations are summarized in Section 4.3 along with an evaluation of how the resonant frequency should be factored in to the ZVS frequency limitation. Moreover, the valleys of the inductor current ripple plot in Fig. 3.4, cannot be avoided by decreasing the switching frequency and consequently, ZVS cannot be maintained at these operating points by using only QSW ZVS and variable frequency control (VFC).

Chapter 4

Dynamic Level Selection

As demonstrated in Chapter 3, zero-voltage switching conditions cannot be maintained across the full duty cycle range due to the current ripple minimums at specific duty cycles. If we compare the inductor current ripple across the full duty cycle range, Fig. 3.4, for FCMLs with an adjacent number of levels (e.g. 4-Level and 5-Level FMCL), we can see that for duty cycles where one configuration has a current ripple minimum, the other configuration does not. In this work, we propose to take advantage of this fact and use dynamic level selection [11] to maintain a minimally sufficient inductor current ripple required for ZVS operation. Dynamically re-configuring a 5-level FCML to operate as a 4-level FCML can avoid the current ripple minimums of each configuration and maintain ZVS conditions [12].

4.1 Dynamic Level Selection

Being able to re-configure the FCML converter through control techniques alone, can enable customization based on specific operating conditions, such as maintaining inductor current ripple as the duty cycle changes. The inductor current ripple is important for maintaining ZVS conditions to reduce switching losses, and its relationship with the duty cycle differs between different FCML level counts. Fig. 4.1 illustrates the proposed method for selecting the number of levels to operate across all duty cycles. By rearranging (2.5), we can solve for and plot the switching frequency required to achieve ZVS for 4- and 5-level operation with constant output current and constant inductor current ripple across the full range of duty cycles. Also plotted is a minimum switching frequency, $f_{lim,4/5}$, for which the converter is not designed to operate below [10] for 4- and 5-level operation, respectively. This plot is for a constant peak negative inductor current which can be controlled [35] to maintain ZVS. Furthermore, in this work, a constant negative inductor current peak, I_{ZVS}, is chosen along with a constant deadtime, by (3.1 - 3.4). With constant output current, designing for a constant deadtime leads to a constant peak-to-peak inductor current ripple in (2.5).

At each duty cycle, we prioritize 5-level operation because the switch voltage stress and therefore, the switching loss per device, is reduced in the case of a higher number of levels.

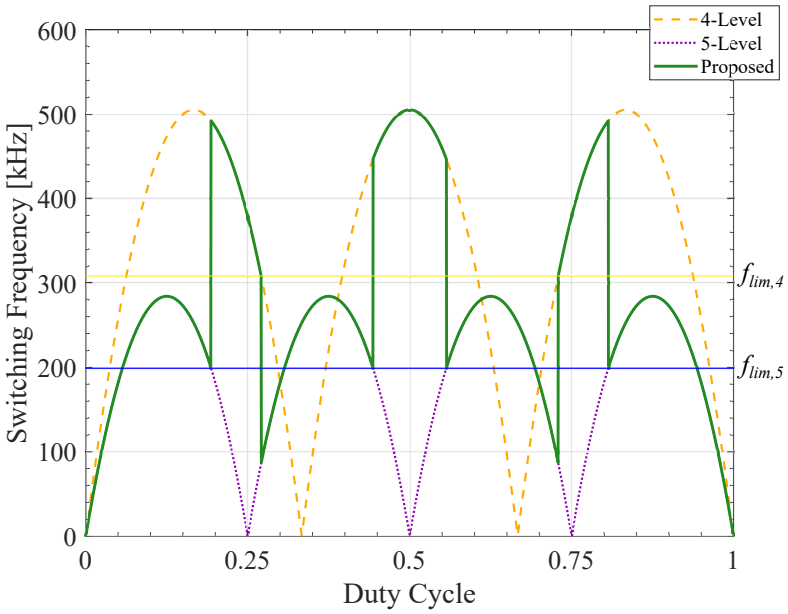


Figure 4.1: The proposed method implements dynamic level changing to avoid operation at the inductor current ripple valleys and to maintain ZVS across the entire duty cycle range.

Moreover, as shown in [36], lower device operating voltage also reduces dynamic $R_{ds,on}$ effects in GaN transistors, another important design consideration. The voltage swing of the inductor is also reduced for the case of a higher number of levels, consequently reducing inductor core losses. If the 5-level switching frequency must be below $f_{lim,5}$ to maintain ZVS, the converter transitions to 4-level operation at a new switching frequency to maintain ZVS. However, there are some duty cycles for which both the 4- and 5-level converter have ZVS frequencies below their respective f_{lim} values; in these cases, we operate as a 5-level converter due to efficiency benefits of a higher level count as described above.

4.2 Level Evaluation

An analysis of the steady-state capacitor voltages for different number of FCML levels is used as reasoning for choosing a 5/4 level converter over a different number of levels. The steady-state capacitor voltages for 5-level operation, as well as for 4-level operation with different switches operated as a pair are shown in Table 4.1. Additionally, the amount of capacitor voltage change required to transition from 5-level to 4-level operation is also shown. This analysis was performed for 5/4, 6/5, and 7/6 level converters. A generalized analysis for any two adjacent number of levels can be expanded from the form of Table 4.1, which shows a 5/4 analysis and of Table 4.2, which shows a 6/5 analysis. Transitioning from a higher odd number of levels down to an even number of levels reduces the capacitor voltage change required. For the 5/4 converter and the 7/6 converter, the minimum voltage change required

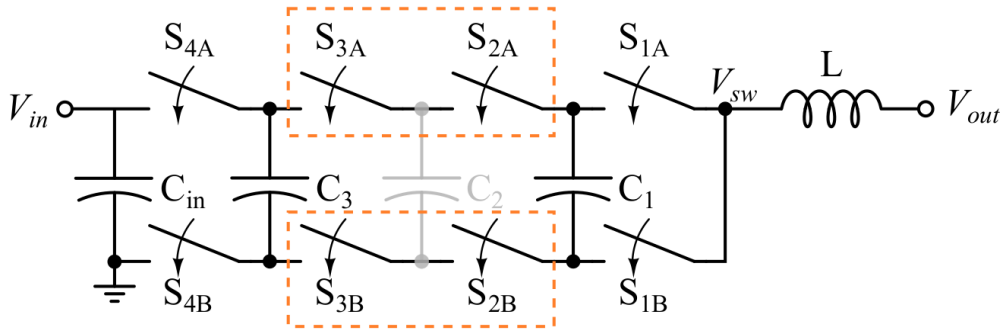


Figure 4.2: The 5-level FCML operated as a 4-level with C_2 voltage maintained at the 5-level value while C_1 and C_3 re-balance to 4-level operation.

is $\frac{1}{12}V_{in}$ and $\frac{1}{15}V_{in}$, respectively compared to the 6/5 level converter which requires $\frac{1}{10}V_{in}$. For the 7/6 level converter, two flying capacitors would need to change by $\frac{1}{30}V_{in}$ and two by $\frac{1}{15}V_{in}$, with one remaining unchanged. However, for the 5/4 level converter, the capacitors which need re-balancing all require the same change in voltage, therefore simplifying the active balancing technique used to re-balance the flying capacitors.

Furthermore, this evaluation of the steady-state flying capacitor voltages is used to select which switch-pair to operate in phase when in the 4-level mode [11]. Configurable-level operation requires switches to be controlled similarly in phase so that the effective number of switches coincides with the desired level operation. For the 5/4 FCML, when the two middle switch pairs, S_3 and S_2 of Fig. 4.2, are operated as one switch pair, configuration 4*b* in Table 4.1, the blocking voltage of the transistors is more evenly distributed, therefore distributing the voltage stress on each of the transistors. Considering the amount of capacitor voltage change required to re-balance on a new number of levels, the configuration with the middle pairs acting as one yields the smallest ΔV, $\frac{1}{12}V_{in}$.

In 4-level operation, the two middle pairs of switches (labeled S_2 and S_3 in Fig. 4.2) are controlled in phase as shown by control signals q_{2A} and q_{3A} in Fig. 4.3a. This switch pair is chosen so that the amount the flying capacitor voltages need to adjust by is minimized. The remaining switch pairs are operated as a 4-level FCML with a phase shift of 120°, shown in Fig. 4.3a. Consequently, the voltage on the middle flying capacitor (labeled C_2 in Fig. 4.2) remains constant at $V_{in}/2$ from the 5-level operation, while the remaining flying capacitors, C_1 and C_3 are re-balanced (actively or passively) to $V_{in}/3$ and $2 \cdot V_{in}/3$, respectively, in accordance with 4-level FCML operation, as shown in Table 4.1.

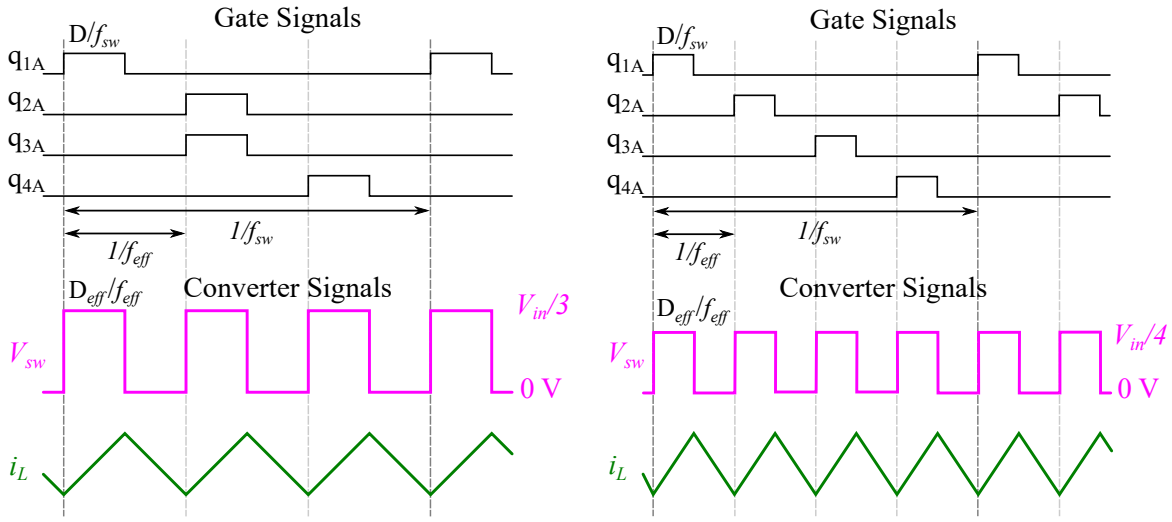
Similarly, when the converter needs to transition from 4-level to 5-level operation, the capacitors are re-balanced to 5-level voltages either by active balancing techniques or passive natural balancing of the converter. The middle switch-pairs are no longer controlled by similar PWM signals and the control scheme returns to that of the 5-level FCML, shown in Fig. 4.3b. When sizing the switches and capacitors, the voltage ratings of the 4-level operation should be used since they are of greater magnitude, as shown in Table 4.1.

Table 4.1: 4/5 Level Switch Pair Configurations and Flying Capacitor Impact

Level	Pair	S_4	S_3	S_2	S_1	V_{C3}	V_{C2}	V_{C1}	ΔV_C		
									C_3	C_2	C_1
5		$\frac{1}{4}V_{in}$	$\frac{1}{4}V_{in}$	$\frac{1}{4}V_{in}$	$\frac{1}{4}V_{in}$	$\frac{3}{4}V_{in}$	$\frac{1}{2}V_{in}$	$\frac{1}{4}V_{in}$	0	0	0
4a	$S_4,\ S_3$	$\frac{1}{4}V_{in}$	$\frac{1}{12}V_{in}$	$\frac{1}{3}V_{in}$	$\frac{1}{3}V_{in}$	$\frac{3}{4}V_{in}$	$\frac{2}{3}V_{in}$	$\frac{1}{3}V_{in}$	0	$+\frac{1}{6}V_{in}$	$+\frac{1}{12}V_{in}$
4b	$S_3,\ S_2$	$\frac{1}{3}V_{in}$	$\frac{1}{6}V_{in}$	$\frac{1}{6}V_{in}$	$\frac{1}{3}V_{in}$	$\frac{2}{3}V_{in}$	$\frac{1}{2}V_{in}$	$\frac{1}{3}V_{in}$	$-\frac{1}{12}V_{in}$	0	$+\frac{1}{12}V_{in}$
4c	$S_2,\ S_1$	$\frac{1}{3}V_{in}$	$\frac{1}{3}V_{in}$	$\frac{1}{12}V_{in}$	$\frac{1}{4}V_{in}$	$\frac{2}{3}V_{in}$	$\frac{1}{3}V_{in}$	$\frac{1}{4}V_{in}$	$-\frac{1}{12}V_{in}$	$-\frac{1}{6}V_{in}$	0

Table 4.2: 5/6 Level Switch Pair Configurations and Flying Capacitor Impact

Level	Pair	S_5	S_4	S_3	S_2	S_1	V_{C4}	V_{C3}	V_{C2}	V_{C1}	ΔV_C			
											C_4	C_3	C_2	C_1
6		$\frac{1}{5}V_{in}$	$\frac{1}{5}V_{in}$	$\frac{1}{5}V_{in}$	$\frac{1}{5}V_{in}$	$\frac{1}{5}V_{in}$	$\frac{4}{5}V_{in}$	$\frac{3}{5}V_{in}$	$\frac{2}{5}V_{in}$	$\frac{1}{5}V_{in}$	0	0	0	0
5a	$S_5,\ S_4$	$\frac{1}{5}V_{in}$	$\frac{1}{20}V_{in}$	$\frac{1}{4}V_{in}$	$\frac{1}{4}V_{in}$	$\frac{1}{4}V_{in}$	$\frac{4}{5}V_{in}$	$\frac{3}{4}V_{in}$	$\frac{1}{2}V_{in}$	$\frac{1}{4}V_{in}$	0	$+\frac{3}{20}V_{in}$	$+\frac{1}{10}V_{in}$	$+\frac{1}{20}V_{in}$
5b	$S_4,\ S_3$	$\frac{1}{4}V_{in}$	$\frac{3}{20}V_{in}$	$\frac{1}{10}V_{in}$	$\frac{1}{4}V_{in}$	$\frac{1}{4}V_{in}$	$\frac{3}{4}V_{in}$	$\frac{3}{5}V_{in}$	$\frac{1}{2}V_{in}$	$\frac{1}{4}V_{in}$	$-\frac{1}{20}V_{in}$	0	$+\frac{1}{10}V_{in}$	$+\frac{1}{20}V_{in}$
5c	$S_3,\ S_2$	$\frac{1}{4}V_{in}$	$\frac{1}{4}V_{in}$	$\frac{1}{10}V_{in}$	$\frac{3}{20}V_{in}$	$\frac{1}{4}V_{in}$	$\frac{3}{4}V_{in}$	$\frac{1}{2}V_{in}$	$\frac{2}{5}V_{in}$	$\frac{1}{4}V_{in}$	$-\frac{1}{20}V_{in}$	$-\frac{1}{10}V_{in}$	0	$+\frac{1}{20}V_{in}$
5d	$S_2,\ S_1$	$\frac{1}{4}V_{in}$	$\frac{1}{4}V_{in}$	$\frac{1}{4}V_{in}$	$\frac{1}{20}V_{in}$	$\frac{1}{5}V_{in}$	$\frac{3}{4}V_{in}$	$\frac{1}{2}V_{in}$	$\frac{1}{4}V_{in}$	$\frac{1}{5}V_{in}$	$-\frac{1}{20}V_{in}$	$-\frac{1}{10}V_{in}$	$-\frac{3}{20}V_{in}$	0



(a) Four-level operation. (b) Five-level operation.

Figure 4.3: Simulated converter waveforms for the proposed method.

4.3 Frequency Limitations

As mentioned in Section 3.3, there are limitations placed on the converter switching frequency, which prevent QSW zero-voltage switching conditions for all duty cycles in an FCML converter. These limits, summarized in Table 4.3 for 5-level operation, are due to converter components [10], such as inductor current saturation, flying capacitor voltage ripple, and due to converter operation (e.g. resonance) [26]. In [10], the component frequency limitations are derived for 4-level operation. Moreover, the resonant frequency of the converter must also be accounted for when determining the lower limit on switching frequency. As shown in Fig. 4.4, when operating near resonant frequency, the inductor current is no longer linear which causes the negative peaks of the inductor current to vary throughout the switching

Table 4.3: Frequency Limits

	$0 < D < \frac{1}{4}$	$\frac{1}{4} < D < \frac{3}{4}$	$\frac{3}{4} < D < 1$
f_{swCfly}	$\frac{I_L \cdot D_{eff}}{2 \cdot C_{fly} \cdot \%V_r \cdot V_{in,pk}}$	$\frac{I_L}{2 \cdot C_{fly} \cdot \%V_r \cdot V_{in,pk}}$	$\frac{I_L \cdot (1-D_{eff})}{2 \cdot C_{fly} \cdot \%V_r \cdot V_{in,pk}}$
f_{swIsat}	$\frac{V_{in,pk} \cdot (D_{eff} \cdot (1-D_{eff}))}{2 \cdot L \cdot (N-1)^2 \cdot (I_{sat}-I_L)}$	$\frac{V_{in,pk} \cdot (D_{eff} \cdot (1-D_{eff}))}{2 \cdot L \cdot (N-1)^2 \cdot (I_{sat}-I_L)}$	$\frac{V_{in,pk} \cdot (D_{eff} \cdot (1-D_{eff}))}{2 \cdot L \cdot (N-1)^2 \cdot (I_{sat}-I_L)}$
f_{swRes}	$\frac{1}{2\pi\sqrt{L \cdot C_{eff}}}$	$\frac{1}{2\pi\sqrt{L \cdot C_{eff}}}$	$\frac{1}{2\pi\sqrt{L \cdot C_{eff}}}$
f_{swZVS}	$\frac{V_{in,pk} \cdot (D_{eff} \cdot (1-D_{eff}))}{2 \cdot L \cdot (N-1) \cdot (I_L - I_{ZVS})}$	$\frac{V_{in,pk} \cdot (D_{eff} \cdot (1-D_{eff}))}{2 \cdot L \cdot (N-1) \cdot (I_L - I_{ZVS})}$	$\frac{V_{in,pk} \cdot (D_{eff} \cdot (1-D_{eff}))}{2 \cdot L \cdot (N-1) \cdot (I_L - I_{ZVS})}$

period. Because of this variation, the converter is unable to maintain ZVS in quasi-resonant operation without additional implementation complexity such as valley current detection and setting specific deadtimes for each current valley. As detailed in [26], there are two resonant frequencies for the FCML converter based on the switching configuration when current flows through either one or two flying capacitors. The resonant frequency, f_{swRes}, is given by the equation in Table 4.3, where C_{eff} is given by either one or two series-connected flying capacitors, depending on the switch configuration. To avoid quasi-resonant operation and maintain linear inductor current, a switching frequency limit is chosen to be sufficiently larger than the resonant frequency of the two flying capacitors in series (about 1.5 to 2.5 times higher).

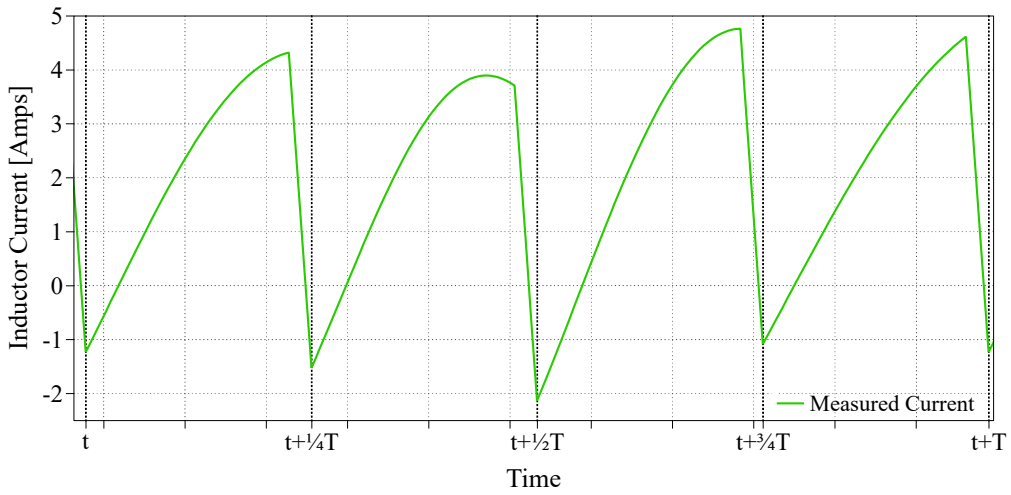


Figure 4.4: When operating near resonant frequency, the inductor current is not linear, and therefore, only quasi-ZVS may be possible.

Chapter 5

Flying Capacitor Active Balancing

Level transitioning, as proposed in Chapter 4, requires flying capacitor re-balancing because the steady-state voltages on flying capacitors, C_1 and C_3 of Fig. 4.2, are at different values based on the number of levels, as shown in Table 4.1. The FCML topology has natural balancing qualities [37–41], which re-align the capacitor voltages with steady-state operation after some time without implementing a new control strategy. However, an extended number of switching cycles spent in an unbalanced condition, leads to more uneven voltage stress on the transistors. To reduce the amount of re-balancing time necessary, active balancing control techniques can be used.

5.1 Constant Effective Duty Cycle Active Balancing

Previous work [11] on level transitioning in FCML converters has used repeated switch states within each cycle to actively increase/decrease the charge on the capacitors. However, here we utilize a technique of duty cycle adjustment [8, 35, 42] to increase or decrease the charge/discharge time of the flying capacitors that require re-balancing. By using duty cycle adjustment instead of repeated states, PS-PWM is maintained with each switch only turning on and off once within a switching cycle. With only one on/off transition in a cycle, the turn-on switching losses, which should be minimized, are limited to occur only once in a switching cycle for each switch. Furthermore, this control technique can be more easily implemented with a micro-controller instead of needing a Field-Programmable Gate Array (FPGA). This control technique, Constant Effective Duty Cycle (CEDC), is done such that the effective duty cycle seen by the inductor, D_{eff} in Fig. 2.3 is kept constant from before, during, and after active balancing by changing the relative phase difference of the control signals as the duty cycles of each signal are adjusted.

In this work, active balancing for a level transition is done while in 4-level operation because FCML converters have more balanced performance on even-numbered levels [41]. When transitioning from 5-level operation to 4-level operation, the active balancing is working in unison with the relatively strong natural tendency of the even-level converter to

re-balance the flying capacitor voltages. To begin a transition from 5- to 4-levels, the duty cycles of each control signal (q_{1A}-q_{4A} and their complements q_{1B}-q_{4B}), as well as the relative phase delay are updated to 4-level PS-PWM values. Gate signals for the two middle switch pairs $q_{2A/B}$ and $q_{3A/B}$ are controlled in phase to emulate 4-level operation. If active balancing is to be used, the duty cycles and relative phase difference of each control signal are adjusted using the CEDC method (altered to have the middle switch pairs with the same phase delay and same duty cycle). For the purpose of maintaining ZVS across the full duty cycle range, the switching frequency needed for ZVS (from Fig. 4.1) is updated as well from the 5-level to 4-level switching frequency so that before and after the transition, the current ripple is maintained and ZVS can occur. During the transition, ZVS conditions are not maintained due to the re-balancing needs.

Here, when transitioning from 4- to 5-level operation, the flying capacitors are re-balanced to 5-level voltages using a 4-level PS-PWM control scheme before the control signals are changed to the 5-level configuration. This choice of active re-balancing is applied for simple implementation in the microcontroller, as well as for more clear juxtaposition with the 5- to 4-level transition re-balancing. However, by actively adjusting the flying capacitors to 5-level steady-state voltages while still in the 4-level configuration, the active balancing control may be fighting against the natural tendency of the flying capacitors to balance at 4-level voltages.

Table 5.1 shows the charge/discharge behavior of the flying capacitors for sub-periods in the lowest duty cycle range of 4-level operation (0 - 33%). This analysis can be similarly extended for the larger duty cycle ranges [35]. For the transition from 5- to 4-level operation, Fig. 5.1, the voltage on capacitor C_1 needs to increase and the voltage on capacitor C_3 needs to decrease, while capacitor C_2 is maintained. To achieve this voltage differential, the sub-period where C_1 charges (indicated by a '+' in Table 5.1), $d2$, when the middle switch pairs (S_2/S_3) are on (indicated by '1'), should be increased, while the sub-period where C_1 discharges (indicated by a '$-$'), $d1$, when switch S_1 is on, should be decreased. Similarly, to decrease the voltage on C_3, sub-periods $d2$ and $d4$ should be increased and decreased, respectively.

In contrast, active re-balancing for the transition from 4- to 5-level operation is accomplished by decreasing $d2$ while increasing $d1$ and $d4$. The sub-periods $d1$ and $d4$ are adjusted equivalently and are changed with respect to $d2$ so that the effective duty cycle at the switch node remains equivalent to that of normal 4-level operation [35].

Equation (5.1) shows the relationship between the switching sub-periods in 4-level oper-

Table 5.1: Flying Capacitor Charge and Discharge Sub-periods

Sub-period	S_{4A}	S_{3A}/S_{2A}	S_{1A}	V_{C3}	V_{C2}	V_{C1}
d1	0	0	1			$-$
d2	0	1	0	$-$		$+$
d4	1	0	0	$+$		

ation and (5.2) shows the relative phase calculation in 4-level operation for duty cycles less than 33%. CEDC expanded for other duty cycle ranges is included in [35]. Applying this duty cycle adjustment technique across multiple switching cycles can re-balance the voltages to the new steady-state operation values as shown in Fig. 5.1. The amount to adjust the sub-periods by is chosen along with the number of active re-balancing cycles to achieve the shortest settling time to balanced flying capacitor voltages.

$$D_{eff} = d2 + d1 + d4 = d2 + 2 \cdot d1 \tag{5.1}$$

$$\Phi_{S_i} = (i-1) \cdot \frac{360^\circ}{(N-1)}; i = 1, 2, ..., (N-1) \tag{5.2}$$

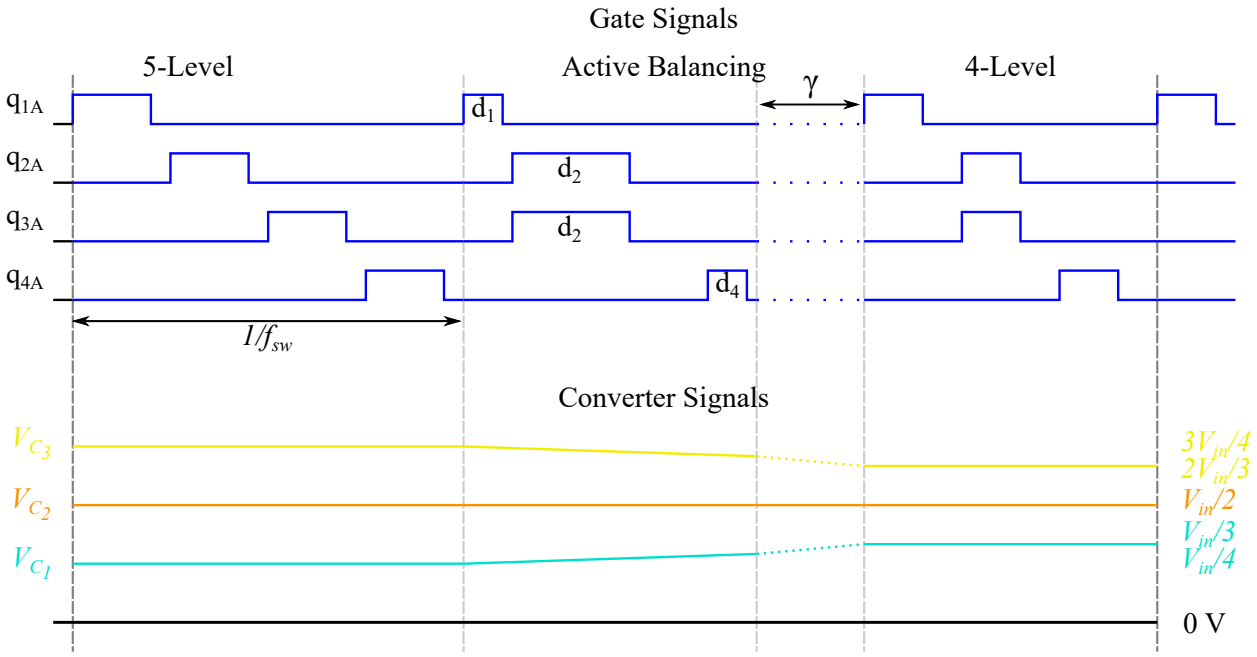


Figure 5.1: Active balancing through duty cycle adjustment for γ number of cycles is implemented at transitions between different numbers of levels.

5.2 Active Balancing Parameter Calculation

The two parameters governing the speed at which the converter balances using constant effective duty cycle (CEDC) active balancing [35] are the duty cycle adjustment, α, and the number of cycles of active balancing, γ. Equations (5.3) - (5.5) show the relationship between the duty cycle adjustment parameter, α, and the duty cycles and phase delays of each control signal, while maintaining the constant effective duty cycle (5.1). These equations are given

in terms of the sub-periods, T_1, T_2, T_4 shown in Fig. 5.2. A mathematical approach is used to find the shortest settling time of the flying capacitor voltages by determining the best combination of α and γ.

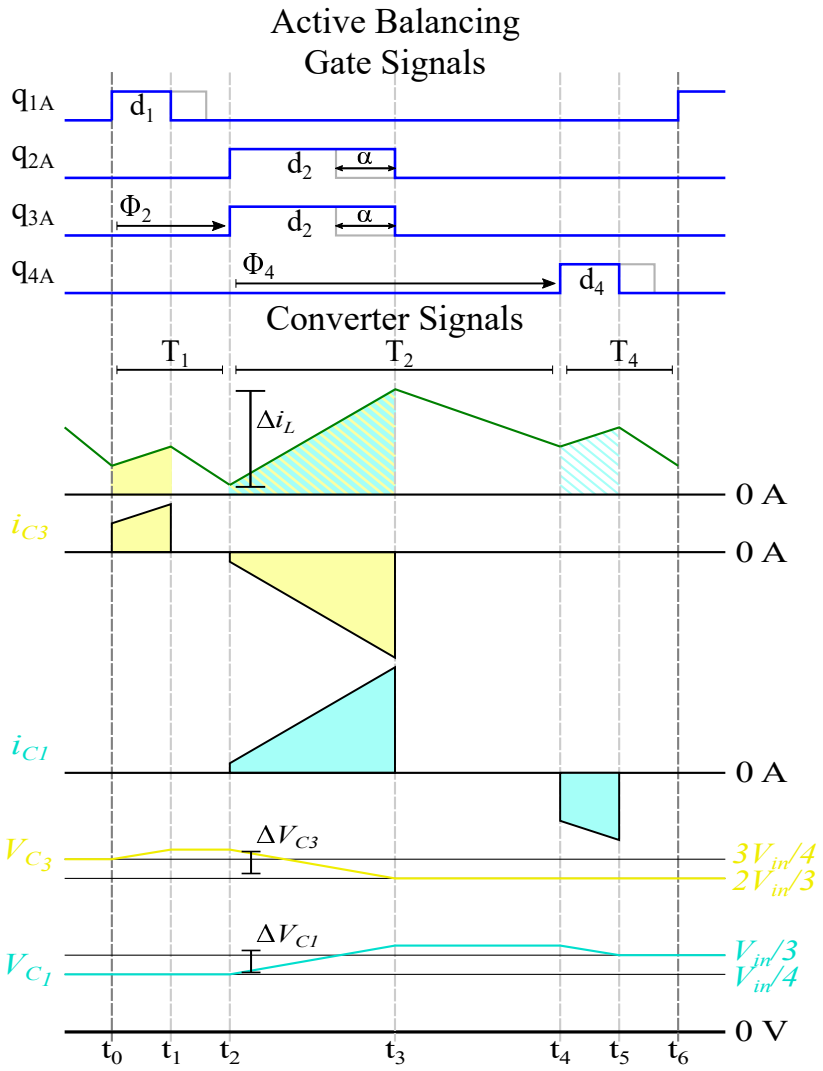


Figure 5.2: Sub-periods for the lowest duty-cycle range of a 5/4-level FCML for calculating active balancing capacitor voltages.

$$T_0 = \frac{1}{(N-1)} \tag{5.3a}$$
$$T_2 = \alpha \cdot T_0, \tag{5.3b}$$
$$T_1 = \frac{1 - T_2}{2}, \tag{5.3c}$$
$$T_4 = T_1 \tag{5.3d}$$

$$d2 = T_2 \cdot D_{eff} = \alpha \cdot D, \tag{5.4a}$$
$$d1 = T_1 \cdot D_{eff} = \frac{D_{eff} - d2}{2}, \tag{5.4b}$$
$$d4 = d1 \tag{5.4c}$$

$$\Phi_{S_1} = 0, \tag{5.5a}$$
$$\Phi_{S_2} = T_2 = \alpha \cdot \frac{360^\circ}{(N-1)}, \tag{5.5b}$$
$$\Phi_{S_3} = T_2 = \Phi_{S_2}, \tag{5.5c}$$
$$\Phi_{S_4} = T_2 + T_1 = 180^\circ(1 - \frac{\alpha}{(N-1)}) \tag{5.5d}$$
$$\tag{5.5e}$$

Fig. 5.3 describes the steps for determining the combination of duty cycle adjustment (α) and number of active balancing cycles (γ) that corresponds to the shortest settling time during level transitioning. Converter parameters are defined, such as input voltage, V_{in}, average output current, I_{out}, inductance, L, flying capacitance, $C_{fly,k}$, output capacitance, C_{out}, and the levels for transitioning, starting level, N_0, ending level, N_1. The valley current, I_{ZVS}, is defined for calculating the ZVS frequencies. Furthermore, the design space is set up to limit the duty cycle adjustment parameter, α, and the number of active balancing cycles, γ. The parameter, α is constrained by not allowing any of the sub-periods, T_1, T_2, T_4 to be greater than 1 (where 1 corresponds to a full switching period). This limitation is also equivalent to maintaining sub-period duty cycles, $d1, d2, d4$, below D_{eff}. Equations (5.6a-d) show the calculation of this limit for the lowest duty cycle range. The limit on the number of cycles, γ is decided by the designer.

$$d2 \leq D_{eff}, \tag{5.6a}$$
$$d2 = \alpha D, \tag{5.6b}$$
$$D_{eff} = (N-1)D, \tag{5.6c}$$
$$\alpha \leq (N-1) \tag{5.6d}$$

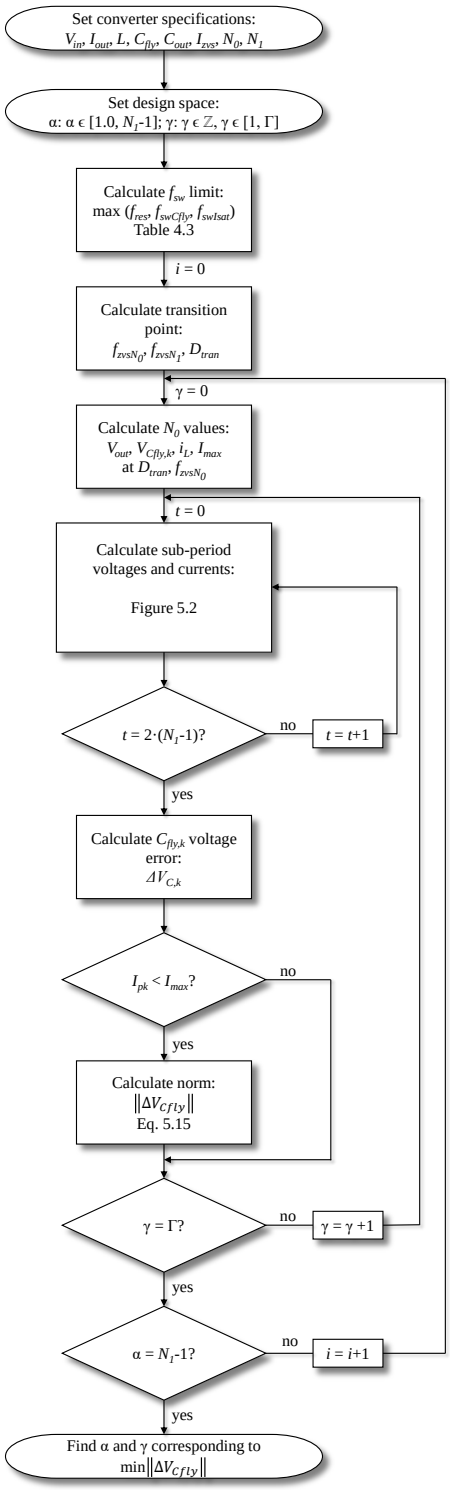


Figure 5.3: Flowchart for determining the α and γ combination for the fastest active rebalancing during dynamic level transitioning.

With all of the parameters specified, the switching frequency limitation for the converter is calculated based on operating conditions (V_{in}, I_{out}) and component parameters (L, C_{fly}). A switching frequency limitation is chosen for both 4- and 5-level operation by choosing the maximum switching frequency of the calculated switching frequencies (f_{swCfly}, f_{swIsat}, and f_{swRes} in Table 4.3) across all duty cycles. If the converter operates above the maximum of these limits, none of the limits will be violated at any duty cycle. Then, based on these switching frequency limits, the duty cycle, D_{tran}, when the ZVS frequency crosses the limit, $f_{lim,N}$, is calculated. At this duty cycle the converter will dynamically transition levels to maintain ZVS. Because 5-level operation is prioritized, if the ZVS switching frequency falls below the 5-level limit, the converter transitions from 5- to 4-level only until the 5-level ZVS frequency is above the 5-level limit, at which point the converter transitions back to 5-level operation. If both the 4- and 5-level ZVS frequencies violate their respective limits, 5-level operation is used as discussed in Section 4.1.

$$I_{max} = I_{zvs} + \Delta i_{pp,max} \tag{5.7}$$

$$\Delta V_{out} = \frac{1}{N}\frac{\Delta i_{pp,max}}{8 f_{sw} C_{out}} \tag{5.8}$$

The maximum inductor current allowed, I_{max}, is calculated (5.7) based on the output capacitance allowable ripple (5.8) [26] and the negative inductor current peak, I_{zvs}. The N_0 voltages and currents are calculated for the starting point of the level transitioning and active re-balancing. Then the converter voltages and currents are calculated for the region of active balancing based on a value-pair: the duty cycle adjustment α and the number of active balancing cycles, γ. Fig. 5.2 shows an example active balancing switching cycle with sub-periods used in calculating the voltages and currents. For an N-level converter, there are $2 \cdot (N-1)$ sub-periods within a switching cycle. For each sub-period, the following are calculated:

1. Inductor voltage, $V_L(t)$, from $V_{sw}(t-1)$ and $V_{out}(t-1)$
2. Change in inductor current, $\Delta i_L(t)$, from (5.9), where d_x is the ratio of the sub-period to the full switching period
$$\Delta i_L(t) = \frac{V_L \cdot d_x \cdot T}{L} \tag{5.9}$$
3. Updated inductor current:
$$i_L(t) = i_L(t-1) + \Delta i_L(t) \tag{5.10}$$
4. Capacitor de-rating for $C_{fly,k}(t)$ and $C_{out}(t)$, based on $V_{C,k}(t-1)$ and $V_{Cout}(t-1)$
5. Capacitor current, $\pm\Delta i_L(t)$, depending on the sub-period

6. Change in flying capacitor voltage, (5.11), where dt is the length of time of the subperiod (area under the capacitor current curve)

$$\Delta V_{C,k}(t) = \frac{1}{C_{fly,k}} \int i_{C,k} dt \tag{5.11}$$

7. Change in output capacitor voltage, (5.12), where dt is the length of time of the subperiod (area under the capacitor current curve)

$$\Delta V_{Cout}(t) = \frac{1}{C_{out}} \int (i_L - I_{out}) dt \tag{5.12}$$

8. Updated capacitor voltages:

$$V_{C,k}(t) = V_{C,k}(t-1) + \Delta V_{C,k}(t) \tag{5.13}$$

$$V_{Cout}(t) = V_{Cout}(t-1) + \Delta V_{Cout}(t) \tag{5.14}$$

Once a full switching cycle has been calculated, the differences between the calculated voltages on the flying capacitors and the goal voltages (of N_1 level) are calculated. This process is then repeated for the number of cycles determined by γ. The peak current during active balancing is checked against the maximum allowed current based on the output capacitor voltage ripple. If the peak current that occurs during active balancing violates the maximum current limit, then the $\alpha - \gamma$ pair is deemed invalid and no further calculations are done with this pair.

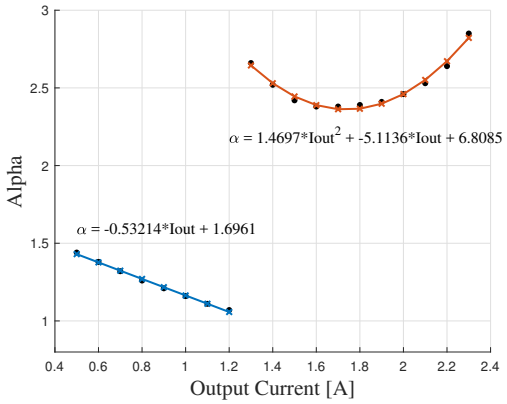
Next, for all valid combinations, the Euclidean norm for the flying capacitor voltage differences is calculated based on (5.15). It is assumed that the α-γ value-pair with the minimum deviation, or minimum Euclidean norm, is the combination that will correspond to the shortest settling time since the flying capacitor voltages are the closest to the goal at the end of the active balancing stage. However, this does not take into account the length of any residual settling time for natural balancing needed if the capacitor voltages are not exactly at the goal voltages after active balancing. It is possible that the number of cycles needed for the smallest deviation after active balancing plus additional settling time corresponds to a longer physical time than a different $\alpha - \gamma$ pair (with a larger voltage deviation from the goal) with fewer active balancing cycles, but more natural balancing time. While the analysis is set up to calculate for natural balancing, as discussed later in Section 6.4, the current implementation of the active balancing calculations is not very accurate for natural balancing settling time because parasitic resistances and switch resistances are neglected.

$$||V_c||_2 = \sqrt{\Delta V_{c1}^2 + \Delta V_{c3}^2} \tag{5.15}$$

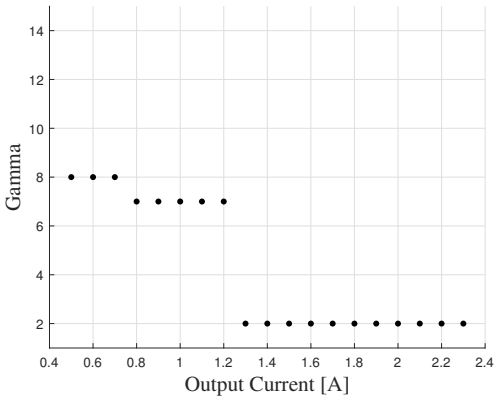
This process, calculated in Matlab (see Appendix A), was repeated for several average output current values (at an input voltage of 50 V and 75 V with a 5.6 μH inductor) and

a curve-fit was done for the parameter values of α and γ, shown in Figs. 5.4a and 5.4b for 50 V and in Figs. 5.4c and 5.4d for 75 V. The number of active balancing cycles, γ, exhibits a staircase form. For each gamma value, the corresponding α values exhibit a linear relationship with the average output current, up to a point. After a certain output current value, the γ drops drastically, and the α parameter becomes quadratic. This relationship is because the active balancing mechanism is severely limited by the constraint on the output voltage ripple. In this range of output current, the shortest settling time is achieved by allowing the maximum output voltage disturbance, and therefore the largest inductor current ripple (which corresponds to a large α value), but only for a short number of cycles. For the average output current values that are not severely limited, a lower duty cycle increase is needed as the current increases because there is more current and therefore more charge that can be controlled to charge/discharge the capacitors in a smaller sub-period. Similarly, for increasing current, a shorter number of cycles can be used due to the increased available charge for re-balancing. However, even though the parameters α and γ are decreasing with increased average output current, the output voltage ripple is increasing due to the excess charge. Once the output voltage ripple limit is reached, the relationship among the active balancing parameters changes as discussed above.

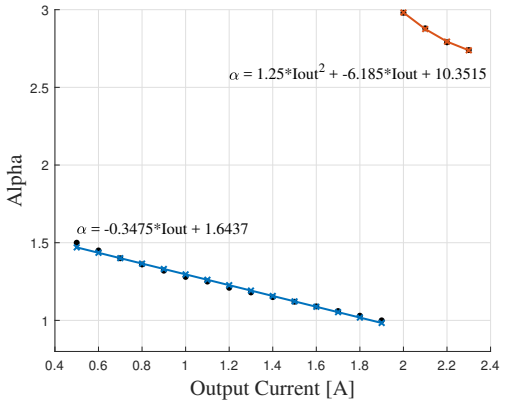
As the voltage is increased, the point at which active balancing is limited by the output voltage ripple is extended because the absolute magnitude of voltage deviation is larger with a larger input voltage (and consequently larger output voltage for the same initial duty cycle). The slope of the linear region of α values is less steep as voltage increases because these values correspond to higher γ values. For a larger input voltage, higher γ values are needed because the flying capacitors need to re-adjust by a larger magnitude as a proportion of the input voltage. Using these two plots, the value-pair that will give the shortest settling time can be determined for any output current for a specified input voltage value.



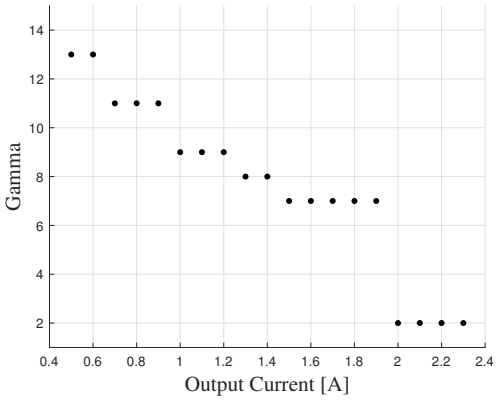
(a) The magnitude of duty cycle adjustment that corresponds to the fastest active re-balancing in combination with the gamma value in Fig. 5.4b for several output current values at an input voltage 50 V.



(b) The number of active balancing cycles that correspond to the fastest active re-balancing in combination with the alpha value in Fig. 5.4a for several output current values at an input voltage 50 V.



(c) The magnitude of duty cycle adjustment that corresponds to the fastest active re-balancing in combination with the gamma value in Fig. 5.4b for several output current values at an input voltage 75 V.



(d) The number of active balancing cycles that correspond to the fastest active re-balancing in combination with the alpha value in Fig. 5.4c for several output current values at an input voltage 75 V.

Figure 5.4: The active balancing parameters corresponding to the shortest settling time exhibit a similar relationship with output current for different input voltages.

Chapter 6

Experimental Results

A 5-level FCML converter was built to demonstrate this control technique that maintains ZVS across the full duty cycle range. Operation as a 5-level FCML and as a 4-level FCML was tested, as well as ZVS operation at various duty cycles. Of most interest are the duty cycles highlighted in Fig. 3.4 for which ZVS is not possible on a certain level count. The dynamic transitioning between levels was tested with natural balancing and with CEDC active balancing implemented. Finally, the efficiency benefits of employing this technique of ensuring ZVS are also examined.

6.1 Experimental Prototype

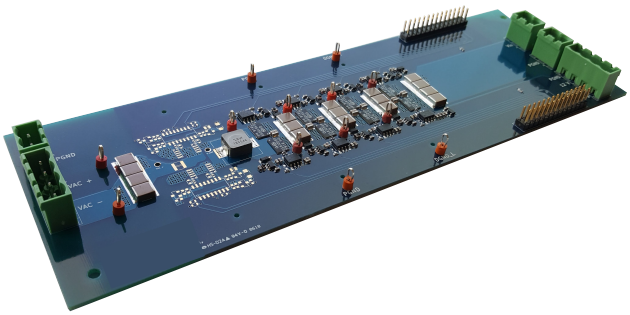


Figure 6.1: Hardware Prototype.

The constructed 5-level FCML converter, Fig. 6.1 and 6.3, was built to demonstrate this control technique that maintains ZVS across the full duty cycle range. The circuit schematic and printed circuit board (PCB) layout are included in Appendix B. The prototype was built using 100 V GaN devices from GaN Systems due to their low conduction and switching losses, as well as their low output capacitance which is important for ZVS design. Because these GaN devices are bottom-side cooled, the FCML was constructed on a single-sided PCB to facilitate a heat sink across the whole bottom side. Assembling the FCML on a single side

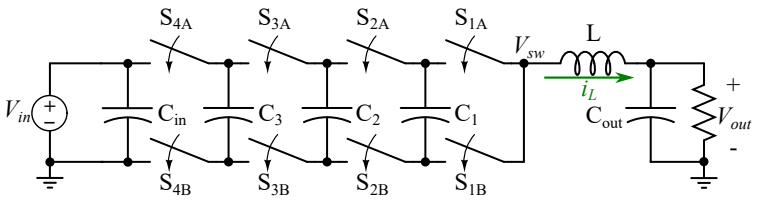


Figure 6.2: Five-level FCML circuit schematic drawing.

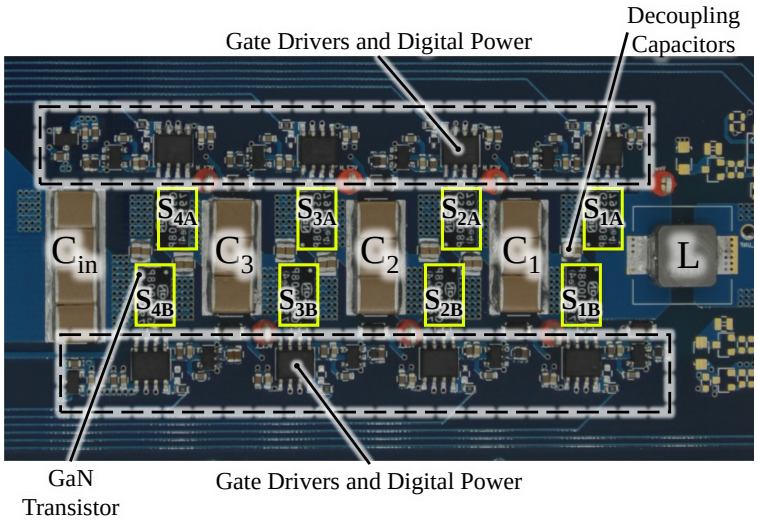


Figure 6.3: Annotated photograph of the experimental prototype.

increases the commutation loop and introduces more parasitic inductance into the conduction path. To decrease the commutation loop area and absorb the excess parasitic energy, local decoupling capacitors are used for each switch pair [10]. Additionally, previous work [43] has demonstrated the merit of using a cascaded bootstrap technique to power the isolated gate drivers for each switch of the FCML. The cascaded bootstrap technique has a reduced area and better efficiency when compared to the conventional single IC isolated gate driver [43]. This prototype was designed with an unfolder stage for DC to AC capabilites, however, in this work, the unfolder stage was unused. A Texas Instruments C2000 microcontroller, TMX320F28377D, was chosen for its low-cost, number of PWM pins, and simplicity of code implementation. A control card was used to interface the microcontroller with the FCML PCB. Table 6.1 shows the full component listing of the hardware prototype.

6.2 Zero-Voltage Switching

To demonstrate the proposed method, the experimental prototype was tested in multiple operating conditions. Fig. 6.4 shows the converter operating as a 4-level FCML at an input voltage of 100 V, 10 W, a switching frequency of 350 kHz, and a duty ratio of 25%, which, as

Table 6.1: Component Listing of the Hardware Prototype

Function Block	Component	Mfr. & Part Number	Parameters
FCML	GaN FETs	GaN Systems GS61008P	100 V, 7mΩ
	Capacitors (C_1, C_2, C_3)	TDK C5750X6S2W225K250KA × 3	450 V, 2.2μF
	Capacitors (C_{in})	TDK C5750X6S2W225K250KA × 8	450 V, 2.2μF
	Capacitors (C_{out})	TDK C5750X6S2W225K250KA × 8	450 V, 2.2μF
	Inductor (L)	Vishay IHLP4040DZ-01	26 A, 2.2μH
Cascaded Bootstrap	Isolated gate drivers	Silicon Labs SI8271GB-IS	
	Bootstrap Diodes	Vishay VS-2EFH02HM3	400 V
	LDO	Texas Instruments LP2985IM5-6.1/NOPB	
Controller Board	Logic level shifters	Texas Instruments SN74LV4T125PWR	
	Microcontroller	Texas Instruments TMX320F28377D	

shown in Fig. 3.4, is an operation point where the 5-level FCML has no current ripple and cannot maintain ZVS. The inductor current ripple is shown to go negative which discharges the parasitic capacitances of the high-side transistors and allows ZVS, which is evident by the minimal overshoot on the rising edge of the switch-node voltage, V_{sw}. Likewise, Fig. 6.5 shows the converter operating in ZVS as a 5-level FCML at the same voltage and loading condition, a 255 kHz switching frequency and a 33% duty ratio, which is a current ripple valley of the 4-level converter. These results show that ZVS is possible at two different duty cycles for which ZVS is not possible with a fixed number of levels.

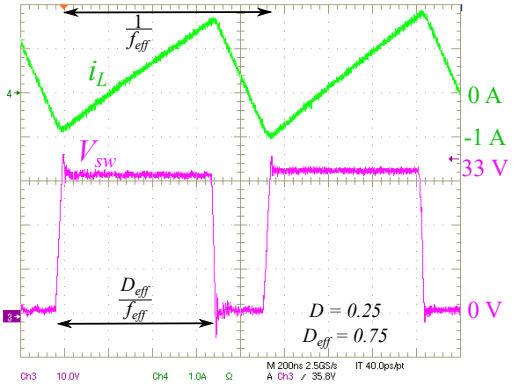


Figure 6.4: ZVS is achieved for 4-level operation at a duty cycle for which 5-level operation cannot achieve ZVS.

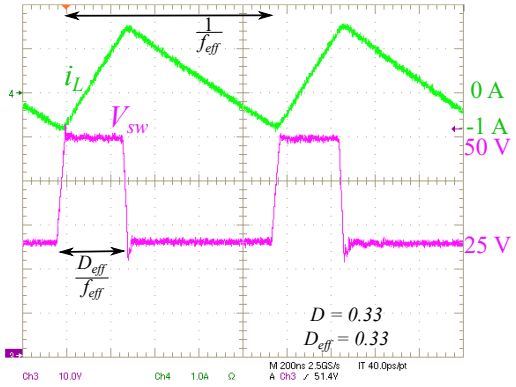
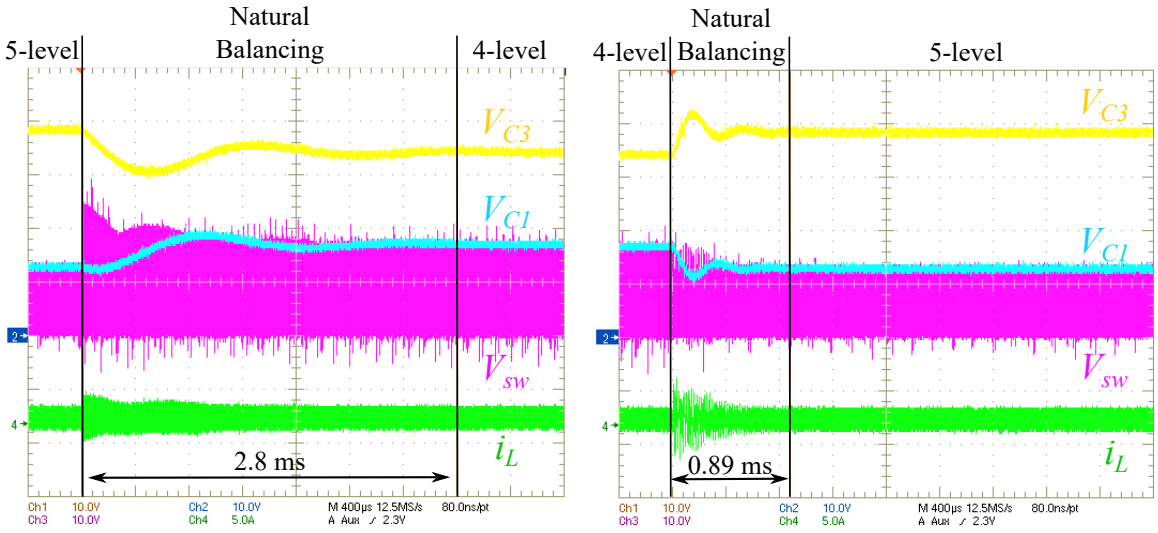


Figure 6.5: ZVS is achieved for 5-level operation at a duty cycle for which 4-level operation cannot achieve ZVS.

6.3 Dynamic Level Transitioning

A dynamic transition between levels is demonstrated in Fig. 6.6. Due to microcontroller limitations in implementing level transitioning, the dynamic level transitioning was tested with a 5.6 μH Vishay inductor (IHLP-3232DZ-5A) at an input voltage of 50 V and an average output current of 0.5 A. When implementing level transitioning and CEDC active balancing on the Texas Instruments C2000 microcontroller, there were timing challenges when updating the switching frequency, duty cycle, and phase delay at the same time. The microcontroller code used in this work is given in Appendix C. In the case demonstrated in Fig. 6.6, the converter transitions from 5-level to 4-level operation (Fig. 6.6a) and vice-versa (Fig. 6.6b) with only natural balancing. The measured settling time of the capacitor voltages, V_{C1} and V_{C3} for the 5- to 4-level transition is about 2.8 ms and from 4- to 5-levels is about 0.89 ms. Here, we see that the transition to the odd-level FCML is faster than to the even-level FCML, this is contrary to our extension of [41], mentioned previously, that even-level FCMLs have better natural balancing. Further investigation is required to explain how the number of levels affects the flying capacitor balancing after a relatively large transient, such as that due to dynamically transitioning levels.



(a) Five to four level transition.

(b) Four to five level transition.

Figure 6.6: Level transitioning with natural balancing.

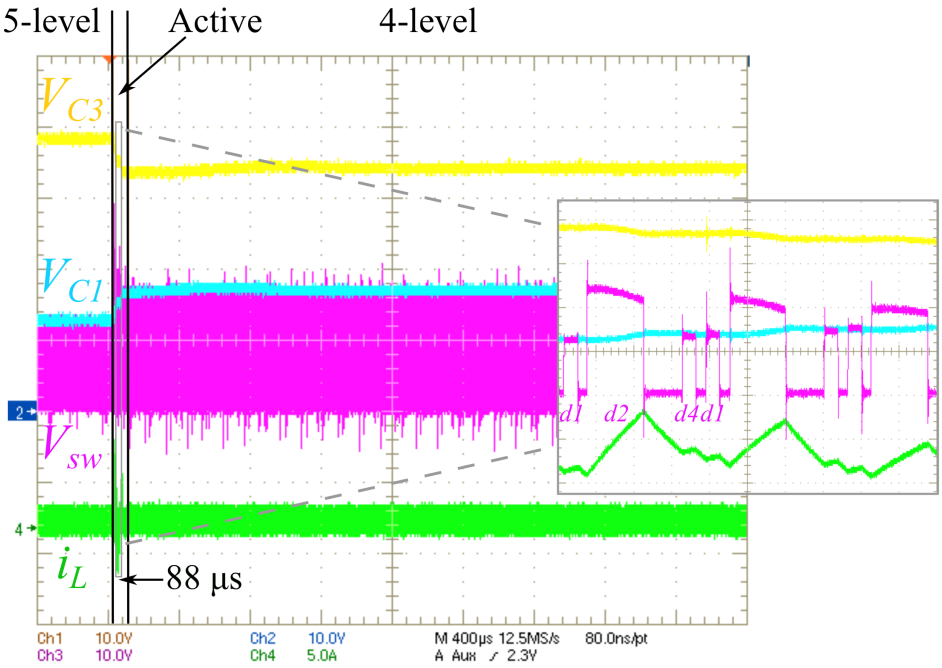


Figure 6.7: Active balancing decreases the settling time of capacitors C_1 and C_3 during a transition from 5- to 4-level operation.

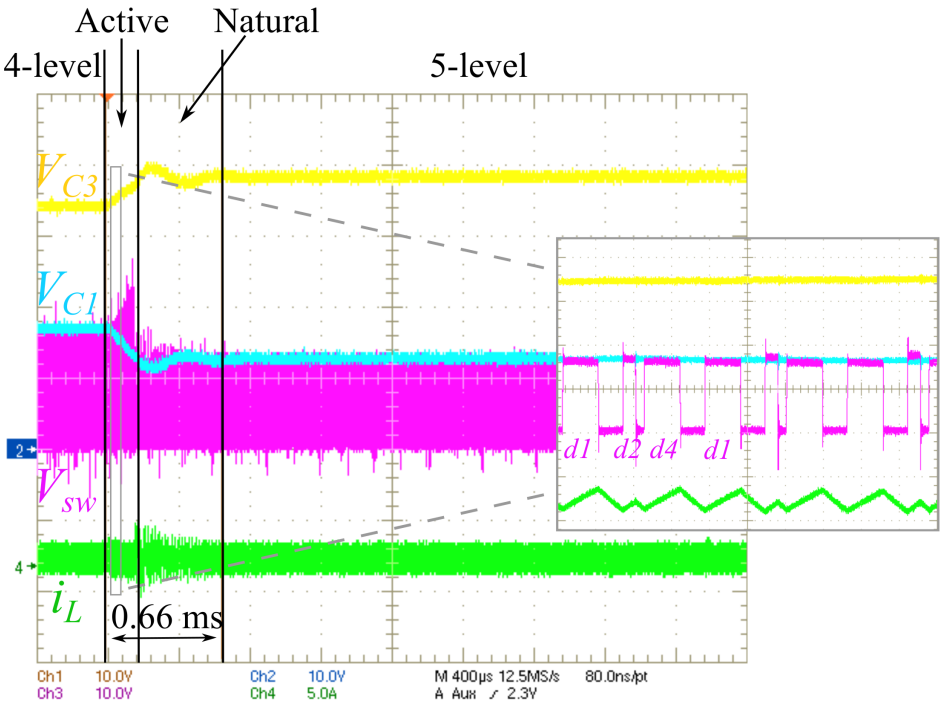


Figure 6.8: Active balancing decreases the settling time of capacitors C_1 and C_3 during a transition from 4- to 5-level operation.

Fig. 6.7 shows the transition from 5- to 4-level operation with a region of active balancing by duty cycle adjustment (CEDC) to charge C_1 and discharge C_3 from 5-level steady-state voltages to 4-level voltages. The capacitor voltages balance to steady-state in 88 μs, which is over 30 times faster than the settling time using natural balancing. Fig. 6.8 shows the transition from 4- to 5-level operation with active balancing, which takes 0.66 ms, which is about 1.3 times faster than natural balancing alone. As mentioned in Section 5.1, the settling time can possibly be reduced if active balancing is employed in 5-level configuration instead of 4-level configuration when transitioning to 5-level operation.

When performing active balancing, two parameters can be tuned for different balancing characteristics — the magnitude of duty cycle adjustment (α), and the number of active balancing cycles (γ). If rapid balancing is desired, the percent change of duty cycles is set high, with a corresponding low number of active balancing cycles. Alternatively, slower, but with less inductor current ripple induced, balancing operation can be achieved with low percent change of duty cycles, and a higher number of active balancing cycles. In the case of the 5- to 4-level transition, shown in Fig. 6.7, seven cycles of active balancing were used and the sub-periods were adjusted: $d2$ was 40%, or twice the width of the baseline duty cycle of 20%, and $d1$ and $d4$ where each 10%, maintaining a constant effective duty cycle at the switch node of 60%. This approach demonstrates a more aggressive duty cycle adjustment with a smaller number of active balancing cycles, which leads to a shorter settling time,

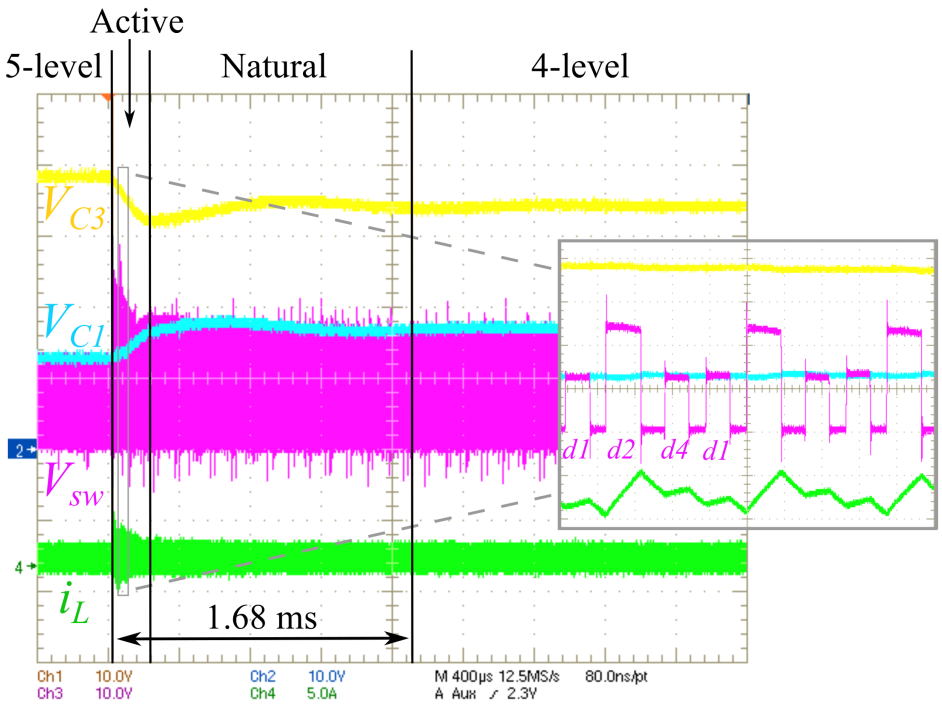


Figure 6.9: Level transitioning with less aggressive active balancing has a longer settling time, but a lower magnitude of increased inductor current ripple.

with the trade-off of a brief time period with increased current ripple. However, if a more moderate adjustment to duty cycle and more cycles of active balancing can be permitted, then the magnitude of the increased current ripple can be lower as shown in Fig. 6.9 ($d2$ at 25% for 25 cycles) as compared to aggressive re-balancing in Fig. 6.7. Furthermore, as shown below in Section 6.4, due to the relatively large current ripple at the beginning of the active balancing transition, there is also a transient response on the output voltage. Constraints can be placed on the allowable inductor current ripple, detailed in Section 5.2, during active balancing to limit the voltage deviation on the output capacitor. Both moderate and aggressive implementations of active balancing still reduce the settling time when compared to natural balancing.

6.4 Active Balancing Parameter Calculations

Section 5.2 details the modeling of dynamic level transitioning and the process by which the active balancing parameters, α and γ, can be determined. To validate this model, the calculated current and voltage waveforms were compared to experimental measurements. Fig. 6.10 shows an overlay of the simulated/calculated flying capacitor voltages, V_{C1} and V_{C3}, output voltage, V_{Cout}, and the inductor current, i_L, on to experimental measured waveforms.

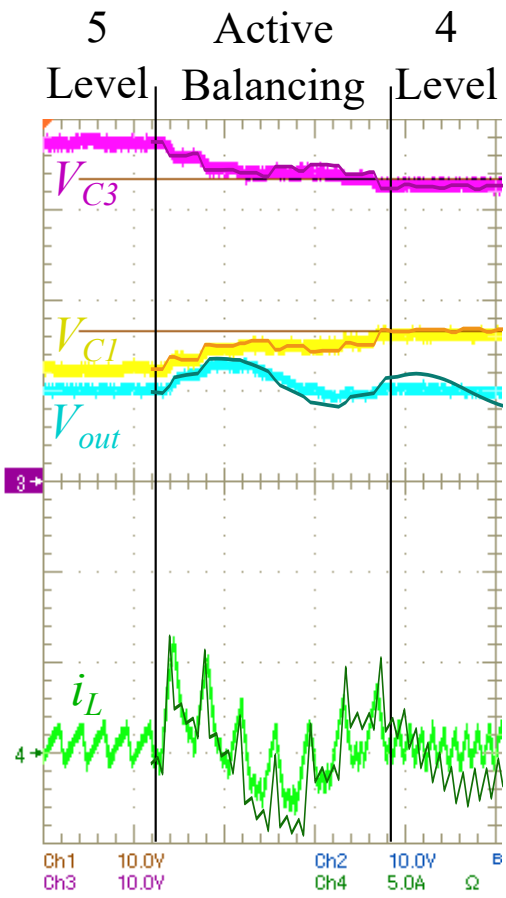
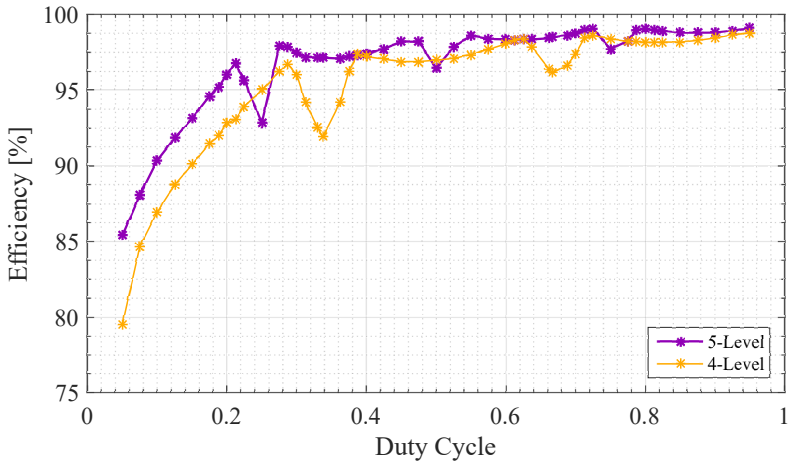
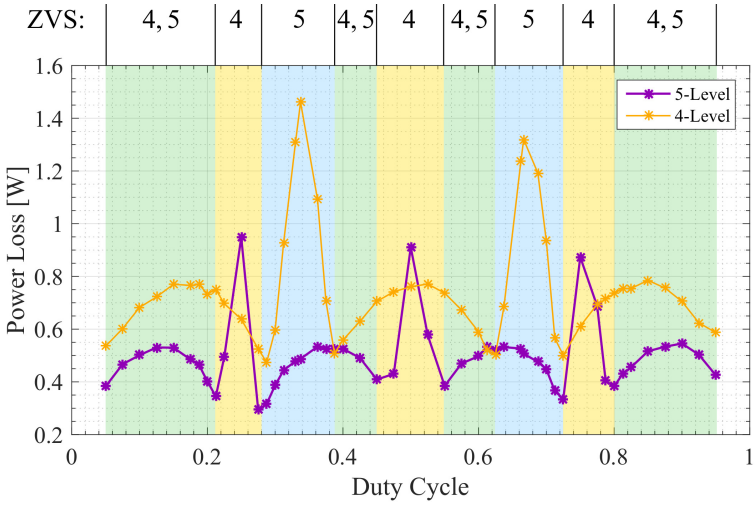


Figure 6.10: The active balancing model calculated in Matlab closely corresponds to measured waveforms for this 5- to 4-level transition with $\alpha = 2.0$ and $\gamma = 7$.

The operating conditions and active balancing parameters shown here are the same as those in Fig. 6.7. During the region of active balancing, the calculated waveforms are very close to the measured waveforms. Again, due to the increased inductor current ripple, there is a voltage deviation on the output capacitor that can be limited by limiting the intensity of the active balancing parameter, α. During normal operation (or natural balancing), the calculations continue to oscillate and do not match closely with experimental data. This discrepancy is because damping, in the form of capacitor equivalent series resistance (ESR) or switch on-state resistance ($R_{ds,on}$) was not factored into the calculations. Implementing the damping factors would allow the simulated system to converge to the balanced steady-state and the total settling time could be calculated. The assumption is made in Section 5.2 that the active balancing parameters corresponding to final flying capacitor voltages closest to the goal are the parameters that yield the shortest settling time.



(a) Efficiency measurements of 4- and 5-level operation, maintaining ZVS where possible without violating converter switching frequency limitations.



(b) Corresponding power loss for 4- and 5-level operation.

Figure 6.11: Higher efficiency points closely correspond with the proposed method in Fig. 4.1.

6.5 Converter Efficiency

To demonstrate the efficiency benefits of the proposed control method, we tested 4- and 5-level operation over a wide range of duty cycles. The efficiency at each duty cycle was measured in 4- and 5-level operation at 100 V_{in} and 0.5 A load with constant negative inductor current peak, I_{ZVS}, with a high precision power analyzer (Keysight PA2201A).

The frequency was adjusted to achieve ZVS conditions, if possible, without violating the converter frequency limitation. When the switching frequency limit is reached, instead of further reducing the frequency, the converter is operated at that limit (without a large enough current ripple for ZVS) until a sufficient ripple can be maintained with a larger

switching frequency (i.e. a frequency above the limit). In regions where both the 4- and 5-level ZVS switching frequency violates the limit, the converter operates in the 5-level mode, with a relaxed switching frequency limit. In the tested operating conditions, because the resonant frequency is the critical frequency for choosing the limit, operating slightly below this limit does not violate the f_{swCfly} or f_{swIsat} limits. However, operating below the switching frequency limit means that the converter is in a quasi-resonant mode, and as describe above, ZVS may only occur on some switching edges instead of all edges. This quasi-resonant operation is allowed until either level mode has a ZVS frequency above its corresponding limit. The switching frequency required for each 4- and 5-level operation is different, which is necessary as discussed above. Fig. 6.11 shows the efficiency of 5- and 4-level operation at each duty cycle, which aligns with the proposed level transitioning technique in Fig. 4.1.

Operation as a 4-level converter is more efficient for duty cycle ranges around 25%, 50%, and 75%, shaded yellow in Fig. 6.11b. The 4-level converter is more efficient than the 5-level converter when the 5-level converter exhibits a current ripple minimum and cannot maintain ZVS, but where the 4-level can. In this case, even though the 4-level converter is operating at a higher switching frequency than the 5-level converter, the switching losses and core losses in non-ZVS 5-level operation are greater than the core losses on the 4-level converter. Furthermore, operation as a 5-level converter is more efficient for regions surrounding 33% and 66%, shaded blue in Fig. 6.11b, which are the regions where the 4-level converter cannot achieve ZVS. In the green-shaded regions, both the 4- and 5-level converters achieve ZVS. In these regions, the 5-level is more efficient because, as shown in Fig. 4.1, in these duty cycle ranges, the switching frequency needed to maintain ZVS for the 5-level converter is lower than for 4-level operation. Due to the higher level count and lower switching frequency, the 5-level converter has lower switching losses and lower inductor core loss.

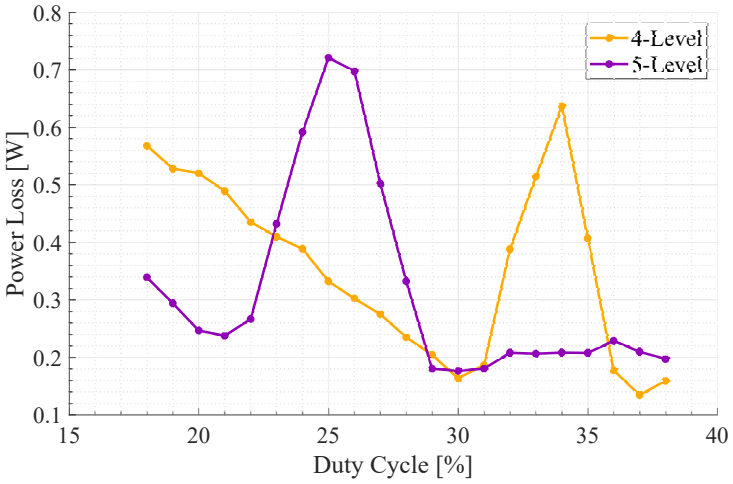


Figure 6.12: Power loss for 4- and 5-level operation with manually tuned ZVS switching frequency.

To further improve the efficiency benefits of level transitioning to maintain ZVS, the ZVS frequency that is calculated can be adjusted based on sensing the current ripple valleys to ensure ZVS conditions, assuming a constant deadtime. A manual adjustment of the switching frequency was performed for the lowest duty cycle range, Fig. 6.12. It can be seen that the ranges for which the converter that has ZVS conditions out-performs the non-ZVS converter are wider than in Fig. 6.11b.

For both the 4- and 5-level converters, when ZVS can be maintained, the losses display a nearly sinusoidal characteristic similar to that of the proposed method and equations of [10]. Despite the 4-level converter operating at a much higher switching frequency, the switching losses can be reduced by maintaining ZVS, therefore demonstrating the benefit of dynamic level transitioning in order to maintain ZVS across duty cycles.

Chapter 7

Conclusions

This thesis presented a method for maintaining ZVS across the full range of duty cycles for an FCML converter by both controlling the switching frequency and dynamically changing the number of levels. An analysis of flying capacitor voltages and switch configurations was used to determine the number of levels and the switching scheme to achieve dynamic level transitioning. Additionally, a method of dynamic level transitioning with active capacitor voltage balancing through duty cycle adjustment was detailed. A hardware prototype was constructed using bottom-side cooled GaN Systems devices, a single-sided PCB for improved cooling methods, and a cascaded bootstrap to supply the isolated gate drivers. The prototype achieved ZVS operation under 4-level and 5-level conditions at duty cycles not possible for a fixed number of levels. Dynamic level transitioning with active re-balancing of the flying capacitors was demonstrated in hardware. A method for determining the active balancing parameters was derived including a curve-fit for simple implementation in a controller. Transitioning between numbers of levels to avoid inductor current ripple valleys and maintain ZVS conditions improves converter efficiency by reducing switching losses, which allows for more power dense designs.

7.1 Future Work

Potential improvements to the current hardware implementation include designing/choosing an optimal inductor for the testing conditions or using an air-core inductor to eliminate the additional inductor core losses incurred by increasing the inductor current ripple. Using an air-core inductor would allow the ZVS benefits to be more evident.

Additionally, future work can delve into the 4-level to 5-level transition. In this work, for both directions of level transitioning (i.e. 4 to 5 and 5 to 4), active balancing was done using CEDC in 4-level PS-PWM configuration. However, it is possible that the 4- to 5-level transition can be improved by actively re-balancing using CEDC in 5-level PS-PWM configuration. Moreover, the dynamic level transitioning can be tested at higher duty cycles and other loading conditions. To test this method at other operating conditions, it

is necessary to resolve microcontroller timing issues pertaining to the frequency and phase delay updates associated with level transitioning and CEDC active balancing. Alternatively, variable frequency control and level transitioning with active balancing can be implemented with an FPGA instead of a microcontroller.

Furthermore, dynamic level transitioning for ZVS can be implemented across a full AC line cycle with load variation and compared to the two static level cases for any efficiency improvements. Closed-loop feedback can also be implemented with ZVS detection and switching frequency adjustment based on sensing the current ripple valleys. The curve-fit active balancing parameters can also be implented in a controller to perform active balancing for level transitioning. Dynamic level transitioning for maintaining current ripple can be compared to the case of minimizing inductor current ripple [11]. However, because the benefits of each of these methods are dependent on which type of converter losses are dominant (inductor core losses or switching losses), perhaps level transitioning for ZVS can be combined with level transitioning for ripple minimization. At low load, minimizing switching losses by level transitioning to maintain ZVS can be prioritized, whereas at higher loads, where inductor core losses and switch conduction losses dominate, level transitioning for ripple minimization can be prioritized.

Bibliography

[1] L. Tolbert, "Power electronics for distributed energy systems and transmission and distribution applications: Assessing the technical needs for utility applications," -, 12 2005.

[2] Half-Cooked Research Reports, "Power electronics market research report- global forecast to 2023," -, Jan 2019.

[3] 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, pp. 397–403 vol.1.

[4] J.-S. Lai and F. Z. Peng, "Multilevel converters-a new breed of power converters," in *Industry Applications Conference, 1995. Thirtieth IAS Annual Meeting, IAS '95., Conference Record of the 1995 IEEE*, vol. 3, Oct 1995, pp. 2348–2356 vol.3.

[5] F. Z. Peng, "A generalized multilevel inverter topology with self voltage balancing," *Industry Applications, IEEE Transactions on*, vol. 37, no. 2, pp. 611–618, Mar 2001.

[6] S. Modeer, Y. Lei, and R. C. N. Pilawa-Podgurski, "An analytical method for evaluating the power density of multilevel converters," in *2016 IEEE 17th Workshop on Control and Modeling for Power Electronics (COMPEL)*, 2016.

[7] Y. Lei, C. Barth, S. Qin, W. c. Liu, I. Moon, A. Stillwell, D. Chou, T. Foulkes, Z. Ye, Z. Liao, and R. C. N. Pilawa-Podgurski, "A 2 kw, single-phase, 7-level, gan inverter with an active energy buffer achieving 216 w/in3 power density and 97.6% peak efficiency," in *2016 IEEE Applied Power Electronics Conference and Exposition (APEC)*, March 2016, pp. 1512–1519.

[8] J. S. Rentmeister and J. T. Stauth, "A 48v:2v flying capacitor multilevel converter using current-limit control for flying capacitor balance," in *2017 IEEE Applied Power Electronics Conference and Exposition (APEC)*, March 2017, pp. 367–372.

[9] D. Chou, Y. Lei, and R. C. N. Pilawa-Podgurski, "A zero-voltage switching, physically flexible multilevel gan dc-dc converter," in *2017 IEEE Energy Conversion Congress and Exposition (ECCE)*, Oct 2017, pp. 3433–3439.

[10] A. Stillwell, M. E. Blackwell, and R. C. N. Pilawa-Podgurski, "Design of a 1 kv bidirectional dc-dc converter with 650 v gan transistor," in *2018 IEEE Applied Power Electronics Conference and Exposition (APEC)*, March 2018, pp. 1155–1162.

[11] N. Vukadinović, A. Prodić, B. A. Miwa, C. B. Arnold, and M. W. Baker, "Ripple minimizing digital controller for flying capacitor dc-dc converters based on dynamic mode levels switching," in *2017 IEEE Applied Power Electronics Conference and Exposition (APEC)*, Mar 2017, pp. 1090–1096.

[12] M. E. Blackwell, A. Stillwell, and R. C. N. Pilawa-Podgurski, "Dynamic level selection for full range zvs in flying capacitor multi-level converters," in *2018 IEEE 19th Workshop on Control and Modeling for Power Electronics (COMPEL)*, June 2018, pp. 1–8.

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

[14] Z. Liao, N. C. Brooks, Z. Ye, and R. C. N. Pilawa-Podgurski, "A high power density power factor correction converter with a multilevel boost front-end and a series-stacked energy decoupling buffer," in *2018 IEEE Energy Conversion Congress and Exposition (ECCE)*, Sep. 2018, pp. 7229–7235.

[15] Z. Liao, Y. Lei, and R. C. N. Pilawa-Podgurski, "Analysis and design of a high power density flying-capacitor multilevel boost converter for high step-up conversion," *IEEE Transactions on Power Electronics*, vol. 34, no. 5, pp. 4087–4099, May 2019.

[16] S. Qin, Y. Lei, Z. Ye, D. Chou, and R. C. N. Pilawa-Podgurski, "A high power density power factor correction front end based on seven-level flying capacitor multilevel converter," *IEEE Journal of Emerging and Selected Topics in Power Electronics*, pp. 1–1, 2018.

[17] E. Candan, A. Stillwell, N. C. Brooks, R. A. Abramson, J.Strydom, and R. C. N. Pilawa-Podgurski, "A 6-level flying capacitor multi-level converter for single phase buck-type power factor correction," in *2019 IEEE Applied Power Electronics Conference and Exposition (APEC)*, Mar 2019, pp. –.

[18] Y. Lei, C. Barth, S. Qin, W. c. Liu, I. Moon, A. Stillwell, D. Chou, T. Foulkes, Z. Ye, Z. Liao, and R. C. N. Pilawa-Podgurski, "A 2 kw, single-phase, 7-level, gan inverter with an active energy buffer achieving 216 w/in3 power density and 97.6% peak efficiency," *IEEE Transactions on Power Electronics*, vol. 32, no. 11, pp. 8570–8581, 2017.

[19] T. Modeer, C. B. Barth, N. Pallo, W. H. Chung, T. Foulkes, and R. C. N. Pilawa-Podgurski, "Design of a gan-based, 9-level flying capacitor multilevel inverter with low inductance layout," in *2017 IEEE Applied Power Electronics Conference and Exposition (APEC)*, March 2017, pp. 2582–2589.

[20] N. Pallo, T. Foulkes, T. Modeer, S. Coday, and R. Pilawa-Podgurski, "Power-dense multilevel inverter module using interleaved gan-based phases for electric aircraft propulsion," in *2018 IEEE Applied Power Electronics Conference and Exposition (APEC)*, March 2018, pp. 1656–1661.

[21] A. Stillwell and R. C. N. Pilawa-Podgurski, "A 5-level flying capacitor multi-level converter with integrated auxiliary power supply and start-up," *IEEE Transactions on Power Electronics*, 2018, in press.

[22] D. Chou, K. Fernandez, and R. C. N. Pilawa-Podgurski, "An interleaved 6-level gan bidirectional converter for level ii electric vehicle charging," in *2019 IEEE Applied Power Electronics Conference and Exposition (APEC)*, Mar 2019, pp. –.

[23] C. P. Henze, H. C. Martin, and D. W. Parsley, "Zero-voltage switching in high frequency power converters using pulse width modulation," in *Proc. 1988. Third Annual IEEE Applied Power Electronics Conf and Exposition APEC '88*, 1988, pp. 33–40.

[24] J. G. Kassakian, "A new current mode sine wave inverter," in *1980 IEEE Power Electronics Specialists Conference*, June 1980, pp. 168–173.

[25] K. Liu and F. C. Lee, "Zero-voltage switching technique in dc/dc converters," in *1986 17th Annual IEEE Power Electronics Specialists Conference*, June 1986, pp. 58–70.

[26] K. Kesarwani and J. T. Stauth, "Resonant and multi-mode operation of flying capacitor multi-level dc-dc converters," in *2015 IEEE 16th Workshop on Control and Modeling for Power Electronics (COMPEL)*, July 2015, pp. 1–8.

[27] V. Vorperian, "Quasi-square-wave converters: topologies and analysis," *IEEE J PWRE*, vol. 3, no. 2, pp. 183–191, 1988.

[28] Y. Naeimi and A. Huang, "Design and optimization of high conversion ratio quasi square wave buck converters," in *2017 IEEE 5th Workshop on Wide Bandgap Power Devices and Applications (WiPDA)*, Oct. 2017, pp. 148–152.

[29] D. Neumayr, D. Bortis, E. Hatipoglu, J. W. Kolar, and G. Deboy, "Novel efficiency-optimal frequency modulation for high power density dc/ac converter systems," in *2017 IEEE 3rd International Future Energy Electronics Conference and ECCE Asia (IFEEC 2017 - ECCE Asia)*, Jul. 2017, pp. 834–839.

[30] A. Avila, A. Garcia-Bediaga, A. Rodriguez, L. Mir, and A. Rujas, "Analysis of optimal operation conditions for gan-based power converters," in *2018 IEEE Energy Conversion Congress and Exposition (ECCE)*, Sep. 2018, pp. 1932–1939.

[31] M. Kasper, R. M. Burkart, G. Deboy, and J. W. Kolar, "Zvs of power mosfets revisited," *IEEE Transactions on Power Electronics*, vol. 31, no. 12, pp. 8063–8067, Dec 2016.

[32] D. Maksimovic, "Design of the zero-voltage-switching quasi-square-wave resonant switch," in *Proceedings of IEEE Power Electronics Specialist Conference - PESC '93*, June 1993, pp. 323–329.

[33] S. Bandyopadhyay and J. Morroni, "Quasi-square wave converters-modeling and performance benefits of gan over silicon," in *2017 IEEE Applied Power Electronics Conference and Exposition (APEC)*, March 2017, pp. 2700–2705.

[34] C. Yeh, X. Zhao, and J. Lai, "An investigation on zero-voltage-switching condition in synchronous-conduction-mode buck converter," in *2017 IEEE Energy Conversion Congress and Exposition (ECCE)*, Oct 2017, pp. 1728–1732.

[35] A. Stillwell, E. Candan, and R. C. N. Pilawa-Podgurski, "Active voltage balancing in flying capacitor multi-level converters with valley current detection and constant effective duty cycle control," in *IEEE Transactions on Power Electronics*, 2019.

[36] T. Foulkes, T. Modeer, and R. C. N. Pilawa-Podgurski, "Developing a standardized method for measuring and quantifying dynamic on-state resistance via a survey of low voltage gan hemts," in *2018 IEEE Applied Power Electronics Conference and Exposition (APEC)*, March 2018, pp. 2717–2724.

[37] R. Wilkinson, T. Meynard, and H. du Toit Mouton, "Natural balance of multicell converters: The general case," *Power Electronics, IEEE Transactions on*, vol. 21, no. 6, pp. 1658–1666, Nov 2006.

[38] X. Yuan, H. Stemmler, and I. Barbi, "Self-balancing of the clamping-capacitor-voltages in the multilevel capacitor-clamping-inverter under sub-harmonic pwm modulation," *Power Electronics, IEEE Transactions on*, vol. 16, no. 2, pp. 256–263, Mar 2001.

[39] A. Ruderman and B. Reznikov, "Five-level single-leg flying capacitor converter voltage balance dynamics analysis," in *Industrial Electronics, 2009. IECON '09. 35th Annual Conference of IEEE*, Nov 2009, pp. 486–491.

[40] S. Thielemans, A. Ruderman, B. Reznikov, and J. Melkebeek, "Improved natural balancing with modified phase-shifted pwm for single-leg five-level flying-capacitor converters," *IEEE Transactions on Power Electronics*, vol. 27, no. 4, pp. 1658–1667, April 2012.

[41] Z. Ye, Y. Lei, Z. Liao, and R. C. N. Pilawa-Podgurski, "Investigation of capacitor voltage balancing in practical implementations of flying capacitor multilevel converters," in *2017 IEEE 18th Workshop on Control and Modeling for Power Electronics (COMPEL)*, July 2017, pp. 1–7.

[42] G. Gateau, M. Fadel, P. Maussion, R. Bensaid, and T. Meynard, "Multicell converters: active control and observation of flying-capacitor voltages," *Industrial Electronics, IEEE Transactions on*, vol. 49, no. 5, pp. 998–1008, Oct 2002.

[43] Z. Ye, Y. Lei, W. c. Liu, P. S. 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 *2017 IEEE Applied Power Electronics Conference and Exposition (APEC)*, Mar 2017, pp. 2925–2931.

Appendix A

Matlab Active Balancing Calculations

Included here are the Matlab files used for calculating the active balancing parameters for a 4/5-level FCML converter using dynamic level selection.

Active Balancing Optimization Routine

Contents


```
% Active Balancing Optimization
clear
tic
```

Constant Parameters

```
c.Vin = 50;
c.L = 5.6e-6;
c.cap = 2.2e-6;
c.num_cap = 3;
c.num_capOut = 4;
c.C = c.num_cap*c.cap;

c.Iv = -0.7; % ZVS
```

```
c.Isat = 9;  % saturation current of inductor

```

Design Space Parameters

```
N = [4,5]; % levels switching between
Iout_avg = linspace(0.5, 4, 8); % average output current

% a = abfactor = alpha
a = linspace(1.0,3.0,201); % factor of duty cycle adjustment on d2

% g = abcycle = gamma
g = 1:40; % number of cycles in Active Balancing
```

Initialize

```
flim = zeros(length(Iout_avg),2);
Dtran = zeros(length(Iout_avg),1);
fsw4 = zeros(length(Iout_avg),1);
fsw5 = zeros(length(Iout_avg),1);
norm = zeros(length(Iout_avg),length(a),length(g));
norm_alt = zeros(length(Iout_avg),length(a),length(g));
diff_Vc1 = zeros(length(Iout_avg),length(a),length(g));
diff_Vc3 = zeros(length(Iout_avg),length(a),length(g));
curr_max = zeros(length(Iout_avg),length(a),length(g));
mini = zeros(length(Iout_avg),1);
index_k = zeros(length(Iout_avg),1);
index_j = zeros(length(Iout_avg),1);
opt = zeros(length(Iout_avg),7);
```

De-rated capacitor look-up table

```
    % C5750X6S2W225K250KA
    c.V_base = linspace(00,500,50001);
    c.C_int = [2.2e-06...]; % interpolated values from data-sheet
```

Loop Statements

```
% for every Iout, calculate Duty cycle where first 5 to 4 transition occurs
for i = 1:length(Iout_avg)
    clock
    i
```

```
47      c.Iout_avg = Iout_avg(i);
48      d = dutyCalc(c,N); % call dutyCalc function
49      flim(i,:) = d.flim;
50      Dtran(i) = d.Dtran;
51      fsw4(i) = d.fsw4;
52      fsw5(i) = d.fsw5;
53
54      for j = 1:length(a) % for each alpha value loop through
55          c.a = a(j);
56          for k = 1:length(g) % for each gamma value loop through
57              c.g = g(k);
58
59              % call main active balancing function
60              abmain = ABmain(Dtran(i), fsw4(i), fsw5(i), N, c);
61
62              norm(i,j,k) = abmain.norm;
63              diff_Vc1(i,j,k) = abmain.diff_Vc1;
64              diff_Vc3(i,j,k) = abmain.diff_Vc3;
65              curr_max(i,j,k) = abmain.curr_max;
66              c.Imax = abmain.c.Imax;
67
68          end
69      end
```

70 Plot Norm of Capacitor Voltages

```
71     % Norm graphs
72 %     figure
73 % hold on
74 % scatter(diff_Vc3(i,:),diff_Vc1(i,:),'k.')
75 % grid on
76 % xlabel('\Delta V_{c3} [V]','FontSize',16,'FontName','Times New Roman');
77 % ylabel('\Delta V_{c1} [V]','FontSize',16,'FontName','Times New Roman');
78 % title(['V_{in}= ' num2str(c.Vin),' V, I_{out}= '
79     num2str(Iout_avg(i)),' A, I_{zvs}= ' num2str(c.Iv),' A
80     '],'FontSize',16,'FontName','Times New Roman');
81 %
82 % figure
83 % hold on
84 % surf(g(1,:),a(1,:),squeeze(norm(i,:,:)))
85 % xlabel('Gamma','FontSize',16,'FontName','Times New Roman');
86 % ylabel('Alpha [cycles]','FontSize',16,'FontName','Times New Roman');
```

```
% zlabel('Norm','FontSize',16,'FontName','Times New Roman');
% title(['V_{in}= ' num2str(c.Vin),' V, I_{out}= '
    num2str(Iout_avg(i)),' A, I_{zvs}= ' num2str(c.Iv),' A
    '],'FontSize',16,'FontName','Times New Roman');
```

Optimization

```
    for j = 1:length(a)
        for k = 1:length(g)
        % Should update this to check against a current maximum based on
        % the output voltage ripple, not saturation current.

        % check peak current is below maximum current allowed
          if (curr_max(i,j,k) > c.Imax)
            norm_alt(i,j,k) = inf; % if above limit, set norm to inf
                                   % so is not a choice for optimal solution
          else
            norm_alt(i,j,k) = norm(i,j,k);
          end
        end
    end

% Find minimum of norm and set optimal parameters
    [mini(i), index_k(i)] = min(min(norm_alt(i,:,:)));
    [mini(i), index_j(i)] = (min(norm_alt(i,:,index_k(i))));
    opt(i,:) = [Iout_avg(i), mini(i),
    curr_max(i,index_j(i),index_k(i)),
    diff_Vc1(i,index_j(i),index_k(i)),
    diff_Vc3(i,index_j(i),index_k(i)), a(index_j(i)), g(index_k(i))];

end

% Plot alpha and gamma vs Iout
%sys_plot(opt)


save('')
toc
```

Transition Duty Cycle Calculation

Contents


```
function duty = dutyCalc(c, N)

Vout_pk = c.Vin;
f_grid = 60;
```

Frequency limitation

```
flim = zeros(1,length(N));

% allowable error between linear and slightly non-linear
error_lim = linspace(0.005, 0.6, 5951)  current
error = zeros(1,2);

for n = 1:length(N)
    f1.flim = 0;
    L5diff = inf;
    L4diff = inf;
    for k = 1:length(error_lim)
    % Should be revisited, basic idea is that once the inductor current
    % deviates from a linear relationship by a certain degree, the switching
    % frequency is too close to the resonant switching frequency
        % Calculate the switching frequency for which the current deviation
        % is below the limit
        % only set up for 4 and 5 level!!
        f = ResFrequencyLimit(c,N(n),error_lim(k));

        % Calculate for 4 and 5 level operation
        % Because there are two resonant frequencies for the FCML in PSPWM
        % the acceptable allowable error is determined by finding
        % the error limit that makes the crossover of the two resonant
        % limits as close to each other as possible at the duty cycle where
        % circuit transitions from being predominately in operation with
```

```
30          % the first resonant frequency to operation with the second
31
32          if (N(n) == 5)
33              if (abs(f.f_diff) <= L5diff) %
34                  L5diff = abs(f.f_diff);
35                  f1.flim = f.flim;
36                  error(1,n) = error_lim(k);
37              else
38                  f1.flim = f1.flim;
39              end
40          else
41              if (abs(f.f_diff) <= L4diff)
42                  L4diff = abs(f.f_diff);
43                  f1.flim = f.flim;
44                  error(1,n) = error_lim(k);
45              else
46                  f1.flim = f1.flim;
47              end
48          end
49      end
50      % Calculate frequency limits based on inductor saturation
51      % and flying capacitor voltage ripple
52      % outputs the maximum of the two
53      f2 = fnFrequencyLimit(c,N(n)); % only set up for 4 and 5 level
54
55      % find the maximum of the limits
56      flim(n) = max([f1.flim, f2.flim]);
57
58  end

59  t = 0:1e-6:1/(4*f_grid);
```

60 Initialize

```
61  Vout=zeros(2,length(t));
62  D=zeros(2,length(t));
63  Deff=zeros(2,length(t));
64  Iout=zeros(2,length(t));
65  Pout=zeros(2,length(t));
66  Ipk=zeros(2,length(t));
67  dIpp=zeros(2,length(t));
68  fsw=zeros(2,length(t));
```

```
Dtran = 0;
```

Loop statement to calculate transition duty cycle

```
 % for a range of duty cycles, calculate the ZVS switching frequency
for i=1:length(t)
    for j=1:length(N)
        Vout(j,i) = Vout_pk*abs(sin(2*pi*f_grid*t(i)));
        %Vout(j,i)=Vout_pk;
        D(j,i) = Vout(j,i)/c.Vin;
        Deff(j,i) = (N(j)-1)*D(j,i)-floor((N(j)-1)*D(j,i));
        Iout(j,i) = c.Iout_avg;
        Pout(j,i)= Iout(j,i)*Vout(j,i);
        Ipk(j,i) = 2*Iout(j,i)-c.Iv;
        dIpp(j,i) = Ipk(j,i)-c.Iv;
        fsw(j,i) = ((c.Vin*(Deff(j,i)*(1-Deff(j,i))))/
                        (c.L*dIpp(j,i)*(N(j)-1)^2))*10^-3; %in kHz
    end

end
```

Calculate Optimum switching frequency for proposed method

```
f_opt0 = zeros(1,length(t));
f_opt0(1) = fsw(2,1);

for i = 1:length(t)
    if (fsw(1,i) >= flim(1) && fsw(2,i) > flim(2))
        f_opt0(i) = fsw(2,i);

    elseif (fsw(1,i) > flim(1) && fsw(2,i) < flim(2))
        f_opt0(i) = fsw(1,i);
        if (Dtran == 0 && D(1,i) >= .125)
            Dtran = D(1,i+1);
            fsw4 = fsw(1,i+1);
            fsw5 = fsw(2,i+1);
        end
    elseif (fsw(1,i) < flim(1) && fsw(2,i) > flim(2))
        f_opt0(i) = fsw(2,i);
    else
        if (i ~= 1 && (f_opt0(i-1) == fsw(2,i-1)))
            f_opt0(i) = fsw(2,i);
```

```
        elseif (i ~= 1 && (f_opt0(i-1) == fsw(1,i-1)))
            f_opt0(i) = fsw(1,i);
        else
            f_opt0(i) = fsw(2,i);
        end
    end

end
```

Outputs

```
duty.error = error;
duty.flim = flim;
duty.fsw4 = fsw4;
duty.fsw5 = fsw5;
duty.Dtran = Dtran;
```

Plots

```
% f_pk=max(fsw(1,1:length(t)));
% figure1 = figure('Name','ZVS Switching Frequency vs. Duty Cycle',
                                                    'Color',[1 1 1]);
% axes1 = axes('Parent',figure1,'YMinorTick','on',...
%     'YMinorGrid','off',...
%     'XMinorTick','on',...
%     'XMinorGrid','off',...
%     'FontSize',16,...
%     'FontName','Times New Roman');
% box(axes1,'on');
% grid(axes1,'on');
% hold(axes1,'all');
%
% f_lim=ones(1,length(t));
% plot(D(1,1:length(t)),fsw(1,1:length(t)), '--',
                            'color', [1 0.667 0], 'LineWidth',1.5)
% plot(D(2,1:length(t)),fsw(2,1:length(t)),':',
                            'color', [0.57 0 0.713],'LineWidth',1.5)
% plot(D(2,1:length(t)),f_opt0(1:length(t)),
                            'color', [0.134 0.55 0.134],'LineWidth',2)
% plot(D(2,1:length(t)),flim(1)*f_lim, 'y')
% plot(D(2,1:length(t)),flim(2)*f_lim, 'b')
%
```

```
145 % set(gca, 'XTick', [0,0.25,0.5,0.75,1])
146 % xlabel('Duty Cycle','FontSize',16,'FontName','Times New Roman');
147 % ylabel('Switching Frequency [kHz]','FontSize',16,
148                 'FontName','Times New Roman', 'Interpreter', 'tex');
149 % grid on;
150 %
151 % set(gca,'FontSize',16, 'FontName','Times New Roman')
152 % h = legend('4-Level','5-Level','Proposed', 'Location', 'northeast');
153 % set(h,'FontSize',12);
```

Resonant Frequency Limit

```
function f = ResFrequencyLimit(c, N, error_lim)

L = c.L;
Izvs = c.Iv;
Idc = c.Iout_avg;
Vin = c.Vin;
D = linspace(0.0126,1,80);

VDC_max_Cfly = (Vin*(N-2)/(N-1));
VDC_mid_Cfly = (Vin*(N-3)/(N-1));
%'C5750X6S2W225K250KA'

C_int = c.C_int;

% instead of interpolate, look up value from table

    index_Vmax = (round(VDC_max_Cfly*100))+1;
    index_Vmid = (round(VDC_mid_Cfly*100))+1;

    Cfly = c.num_cap*C_int(index_Vmax); % read from look-up table
    Cfly_mid = c.num_cap*C_int(index_Vmid);

% C_data = [2.20E+00; 2.23E+00; 2.23E+00; 2.23E+00; 2.23E+00; 2.22E+00;
%           2.18E+00; 2.10E+00; 2.05E+00; 1.93E+00; 1.81E+00; 1.63E+00;
%           1.47E+00; 1.29E+00; 1.10E+00; 9.63E-01; 7.69E-01; 6.39E-01;
%           5.48E-01; 4.31E-01; 3.89E-01; 0.347] * 1e-6;% * 2.014/2.2;
% VDC = [0.00E+00; 1.00E+00; 2.00E+00; 4.00E+00; 6.30E+00; 1.00E+01;
%        1.60E+01; 2.50E+01; 3.00E+01; 4.00E+01; 5.00E+01; 6.50E+01;
%        8.00E+01; 1.00E+02; 1.25E+02; 1.50E+02; 2.00E+02; 2.50E+02;
%        3.00E+02; 4.00E+02; 4.50E+02; 500];

% Capacitor de-rating
% Cfly = c.num_cap*interp1(VDC,C_data,VDC_max_Cfly);
% Cfly_mid = c.num_cap*interp1(VDC,C_data,VDC_mid_Cfly);
C = Cfly;
Ca = Cfly_mid;
Cb = C;

Cx = ((1/Ca)+(1/Cb))^-1; % effective capacitance for two
                         % capacitors in series
```

```
% Calculate resonant frequency
w1 = 1/sqrt(L*C);
wx = 1/sqrt(L*Cx);
iL0 = Izvs; % initial condition for on-time
ipk = 2*Idc-Izvs; % initial condition for off-time
Tmax = 2*pi/(2*wx); % used to scale window of time, set window
                    % to be half the period of the faster current

t = linspace(0,Tmax,1000);

% Initialize
Deff = zeros(1,length(D));
iL1 = zeros(1,length(t));
iL2 = zeros(1,length(t));
iL1off = zeros(1,length(t));
iL2off = zeros(1,length(t));
ton_max = zeros(1,length(D));
toff_max = zeros(1,length(D));
fsw = zeros(1,length(D));
fsw_off = zeros(1,length(D));
Dregion = zeros(1,N-1);

% set boundaries of duty cycle regions
for k = 1:N-1
    Dregion(k) = k/(N-1);
end

% calculate i_1cap and i_2cap for every duty cycle
% resonant current based on two resonant frequencies
% (one cap or two caps in series)
% dependent on duty cycle region
for i = 1:length(D)

    % calculate effective duty cycle
    Deff(i) = D(i)*(N-1) - floor(D(i)*(N-1));

    if (N == 5)
        if (D(i) <= Dregion(1))
            iL1 = iL0*cos(w1*t) + (Vin*(Dregion(1) -
                                    D(i)))*sqrt(C/L)*sin(w1*t);
            iL2 = iL0*cos(wx*t) + (Vin*(Dregion(1) -
                                    D(i)))*sqrt(Cx/L)*sin(wx*t);
```

```
81
82          elseif (D(i) > Dregion(1) && D(i) <= Dregion(2))
83              iL1 = iL0*cos(w1*t) + (Vin*(Dregion(2) -
84                                      D(i)))*sqrt(C/L)*sin(w1*t);
85              iL2 = iL0*cos(wx*t) + (Vin*(Dregion(2) -
86                                      D(i)))*sqrt(Cx/L)*sin(wx*t);
87
88              % off time with different initial conditions
89              iL1off = ipk*cos(w1*t) + (Vin*(Dregion(1) -
90                                      D(i)))*sqrt(C/L)*sin(w1*t);
91              % off time with different initial conditions
92              iL2off = ipk*cos(wx*t) + (Vin*(Dregion(1) -
93                                      D(i)))*sqrt(Cx/L)*sin(wx*t);
94
95          elseif (D(i) > Dregion(2) && D(i) <= Dregion(3))
96              iL1 = iL0*cos(w1*t) + (Vin*(Dregion(3) -
97                                      D(i)))*sqrt(C/L)*sin(w1*t);
98              iL2 = iL0*cos(wx*t) + (Vin*(Dregion(3) -
99                                      D(i)))*sqrt(Cx/L)*sin(wx*t);
100             iL1off = ipk*cos(w1*t) + (Vin*(Dregion(2) -
101                                     D(i)))*sqrt(C/L)*sin(w1*t);
102             iL2off = ipk*cos(wx*t) + (Vin*(Dregion(2) -
103                                     D(i)))*sqrt(Cx/L)*sin(wx*t);
104
105         elseif (D(i) > Dregion(3) && D(i) <= Dregion(4))
106             iL1 = iL2;
107             iL1off = ipk*cos(w1*t) + (Vin*(Dregion(3) -
108                                     D(i)))*sqrt(C/L)*sin(w1*t);
109             iL2off = ipk*cos(wx*t) + (Vin*(Dregion(3) -
110                                     D(i)))*sqrt(Cx/L)*sin(wx*t);
111
112         end
113
114     elseif (N == 4)
115         if (D(i) <= Dregion(1))
116             iL1 = iL0*cos(w1*t) + (Vin*(Dregion(1) -
117                                     D(i)))*sqrt(C/L)*sin(w1*t);
118             iL2 = iL0*cos(wx*t) + (Vin*(Dregion(1) -
119                                     D(i)))*sqrt(Cx/L)*sin(wx*t);
120
121         elseif (D(i) > Dregion(1) && D(i) <= Dregion(2))
122             iL1 = iL0*cos(w1*t) + (Vin*(Dregion(2) -
```

```
123                                     D(i)))*sqrt(C/L)*sin(w1*t);
124             iL2 = iL0*cos(wx*t) + (Vin*(Dregion(2) -
125                                     D(i)))*sqrt(Cx/L)*sin(wx*t);
126             iL1off = ipk*cos(w1*t) + (Vin*(Dregion(1) -
127                                     D(i)))*sqrt(C/L)*sin(w1*t);
128             iL2off = ipk*cos(wx*t) + (Vin*(Dregion(1) -
129                                     D(i)))*sqrt(Cx/L)*sin(wx*t);
130
131         elseif (D(i) > Dregion(2) && D(i) <= Dregion(3))
132             iL1 = iL2;
133             iL1off = ipk*cos(w1*t) + (Vin*(Dregion(2) -
134                                     D(i)))*sqrt(C/L)*sin(w1*t);
135             iL2off = ipk*cos(wx*t) + (Vin*(Dregion(2) -
136                                     D(i)))*sqrt(Cx/L)*sin(wx*t);
137
138         end
139      end
140
141
142      % error_lim from Operation Points
143     error_lim_off = error_lim;
144
145      error = zeros(1,length(t));
146      error_off = zeros(1,length(t));
147
148      tmax = 0;
149      tmax_off = 0;
150
151      for k = 2:length(t)
152
153      % calculate difference in currents (1cap vs 2cap) for on-time current
154          error(k) = (iL1(k) - iL2(k));
155
156     % calculate difference in currents (1cap vs 2cap) for off-time current
157          error_off(k) = iL1off(k) - iL2off(k);
158
159          if ((error(k) >= error_lim) && (error(k-1) < error_lim))
160              % after the error limit is reached set tmax to that time
161              tmax = t(k);
162          end
163          if ((error_off(k) >= error_lim_off) &&
164                                      (error_off(k-1) < error_lim_off))
```

```
165             % after the error limit is reached set tmax to that time
166             tmax_off = t(k);
167         end
168     end
169
170     ton_max(i) = tmax; % each duty cycle has own ton_max
171     toff_max(i) = tmax_off; % each duty cycle has own toff_max
172
173     % calculate switching frequency minimum to satisfy
174     % on-time less that ton_max
175     fsw(i) = Deff(i)/(ton_max(i)*(N-1));
176
177     % calculate switching frequency minimum to satisfy
178     % off-time less that toff_max
179     fsw_off(i) = (1-Deff(i))/(toff_max(i)*(N-1));
180
181 end
182
183  D_off = D;
184  for i = 1:length(fsw)
185
186      if (isfinite(fsw(i))) % don't plot fsw infinite
187          fsw(i) = fsw(i);
188      else
189          fsw(i) = 0;
190          D(i) = 0;
191      end
192  end
193  if N == 4
194      f_diff = fsw_off(40) - fsw(40);
195  else
196      f_diff = fsw_off(30) - fsw(30);
197  end
198 % find maximum of the frequency for on-time calculations
199  [maxf, Dmaxf] = max(fsw);
200  %maxf/1000 % print maximum fsw_on
201  %D(Dmaxf) %print duty cycle for maximum fsw_on
202
203 % figure % plot 2 currents for on and off-times
204 % plot(t,iL1)
205 % hold on
206 % plot(t,iL2)
```

```
% plot(t,iL1off)
% plot(t,iL2off)
% legend('iL1', 'iL2', 'iL1off', 'iL2off');

%%%%%
% figure
% scatter(D,fsw/1000)
% hold on
% scatter(D_off,fsw_off/1000)
% grid on
% grid minor
% xlim([0 1])
% legend('fsw-on', 'fsw-off');

f.f_diff = f_diff;
f.flim = maxf/1000;
```

Component Frequency Limit

```
function f2 = fnFrequencyLimit(c,N)

Vin = c.Vin;
Vout_rms = Vin/sqrt(2);
Iout_dc = c.Iout_avg;

Izvs = c.Iv;

L = c.L;

f_grid = 60;
t = 0:1e-6:1/f_grid;
for i=1:length(t)
        Vout(i) = (Vout_rms*(sqrt(2)))*abs(sin(2*pi*f_grid*t(i)));
        Iout(i) = Iout_dc;
end



Isat = c.Isat;
ripple = 0.1; %percent ripple on Caps


VDC_max_Cfly = max(Vin*(N-2)/(N-1));
%'C5750X6S2W225K250KA'
C.C = [2.20E+00; 2.23E+00; 2.23E+00; 2.23E+00; 2.23E+00; 2.22E+00; 2.18E+00;
        2.10E+00; 2.05E+00; 1.93E+00; 1.81E+00; 1.63E+00; 1.47E+00;
        1.29E+00; 1.10E+00; 9.63E-01; 7.69E-01; 6.39E-01; 5.48E-01;
        4.31E-01; 3.89E-01; 0.347] * 1e-6;% * 2.014/2.2;
C.VDC = [0.00E+00; 1.00E+00; 2.00E+00; 4.00E+00; 6.30E+00; 1.00E+01;
        1.60E+01;  2.50E+01; 3.00E+01; 4.00E+01; 5.00E+01; 6.50E+01;
        8.00E+01; 1.00E+02; 1.25E+02; 1.50E+02; 2.00E+02; 2.50E+02;
        3.00E+02; 4.00E+02; 4.50E+02; 500];
Cfly = c.num_cap*interp1(C.VDC,C.C,VDC_max_Cfly);


Dregion = zeros(1,N-1);
D = zeros(1,length(Vout));
Deff = zeros(1,length(Vout));
```

```
41 Pout = zeros(1,length(Vout));
42 Iin = zeros(1,length(Vout));
43 IL = zeros(1,length(Vout));
44
45 fswCfly = zeros(1,length(Vout));
46 fswIsat = zeros(1,length(Vout));
47 fswZVS = zeros(1,length(Vout));
48
49 for k = 1:N-1
50     Dregion(k) = k/(N-1);
51 end
52
53
54 for i = 1:length(Vout)
55
56     D(i) = Vout(i)/Vin;
57     Deff(i) = (N-1)*D(i)-floor((N-1)*D(i));
58     Pout(i) = Iout(i)*Vout(i);
59     Iin(i) = Pout(i)/Vin;
60     IL(i) = Iout(i);
61
62
63     if (N == 5)
64         if (D(i) <= Dregion(1))
65             fswCfly(i) = IL(i)*Deff(i)/(2*Cfly*ripple*Vin);
66             fswIsat(i) = (Dregion(1)-D(i))*Vin*Deff(i)/
67                                         (2*L*(N-1)*(Isat-IL(i)));
68             fswZVS(i) = (Dregion(1)-D(i))*Vin*Deff(i)/
69                                         (2*L*(N-1)*(IL(i)-Izvs));
70
71         elseif (D(i) > Dregion(1) && D(i) <= Dregion(2))
72             fswCfly(i) = IL(i)/(2*Cfly*ripple*Vin);
73             fswIsat(i) = (Dregion(2)-D(i))*Vin*Deff(i)/
74                                         (2*L*(N-1)*(Isat-IL(i)));
75             fswZVS(i) = (Dregion(2)-D(i))*Vin*Deff(i)/
76                                         (2*L*(N-1)*(IL(i)-Izvs));
77
78         elseif (D(i) > Dregion(2) && D(i) <= Dregion(3))
79             fswCfly(i) = IL(i)/(2*Cfly*ripple*Vin);
80             fswIsat(i) = (Dregion(3)-D(i))*Vin*Deff(i)/
81                                         (2*L*(N-1)*(Isat-IL(i)));
82             fswZVS(i) = (Dregion(3)-D(i))*Vin*Deff(i)/
```

```
                                        (2*L*(N-1)*(IL(i)-Izvs));

        elseif (D(i) > Dregion(3) && D(i) <= Dregion(4))
            fswCfly(i) = IL(i)*(1-Deff(i))/(2*Cfly*ripple*Vin);
            fswIsat(i) = (Dregion(4)-D(i))*Vin*Deff(i)/
                                        (2*L*(N-1)*(Isat-IL(i)));
            fswZVS(i) = (Dregion(4)-D(i))*Vin*Deff(i)/
                                        (2*L*(N-1)*(IL(i)-Izvs));

        end
    elseif (N == 4)
        if (D(i) <= Dregion(1))
            fswCfly(i) = IL(i)*Deff(i)/(2*Cfly*ripple*Vin);
            fswIsat(i) = (Dregion(1)-D(i))*Vin*Deff(i)/
                                        (2*L*(N-1)*(Isat-IL(i)));
            fswZVS(i) = (Dregion(1)-D(i))*Vin*Deff(i)/
                                        (2*L*(N-1)*(IL(i)-Izvs));

        elseif (D(i) > Dregion(1) && D(i) <= Dregion(2))
            fswCfly(i) = IL(i)/(2*Cfly*ripple*Vin);
            fswIsat(i) = (Dregion(2)-D(i))*Vin*Deff(i)/
                                        (2*L*(N-1)*(Isat-IL(i)));
            fswZVS(i) = (Dregion(2)-D(i))*Vin*Deff(i)/
                                        (2*L*(N-1)*(IL(i)-Izvs));

        elseif (D(i) > Dregion(2) && D(i) <= Dregion(3))
            fswCfly(i) = IL(i)*(1-Deff(i))/(2*Cfly*ripple*Vin);
            fswIsat(i) = (Dregion(3)-D(i))*Vin*Deff(i)/
                                        (2*L*(N-1)*(Isat-IL(i)));
            fswZVS(i) = (Dregion(3)-D(i))*Vin*Deff(i)/
                                        (2*L*(N-1)*(IL(i)-Izvs));

        end
    end
end
Cfly_lim = max(fswCfly)/1000;
Isat_lim = max(fswIsat)/1000;
ZVS_lim = max(fswZVS)/1000;
f_lim = max([Cfly_lim, Isat_lim]);
f2.flim = f_lim;
```

Main Active Balancing

Contents


```
% Active Balancing Calculations, specific to 4 level AB
% Also currently specific to lowest duty cycle range
% Assume capacitor voltage is constant in each subperiod ty to tz

function abmain = ABmain(D, fsw4, fsw5, N, c)
```

values

```
    c.D0 = D;
    c.T0 = 1/(fsw5*10^3);
    c.N0 = N(2);
    c.N1 = N(1);
    c.Def0 = (c.N0-1)*c.D0-floor((c.N0-1)*c.D0);
    c.T = 1/(fsw4*10^3);
    c.Tef4 = c.T/(c.N1-1);


    c.R = c.D0*c.Vin/c.Iout_avg;

    c.Def4 = (c.N1-1)*c.D0-floor((c.N1-1)*c.D0);
    c.kx = [1 2 3];
    c.div = 75;
```

design parameters

```
    Iout = c.Iout_avg;
    a = c.a; % factor of duty cycle adjustment on d2
    abcount = c.g; % number of cycles in Active Balancing
    endcount = round(1.2.*abcount); % total number of cycles to calculate
    %waveforms = zeros(1,2);
```

initialize

```
    c.Vc = zeros(3, 7); % capacitor voltages at times 1-6 (cap x time)
    c.dVc = zeros(3,3,2); % change in capacitor voltage specific for 4 case
    c.VL = zeros(2, 3); % inductor voltage
    c.ix = zeros(1, 7);        % inductor current
    c.di = zeros(2, 3); % change in inductor current

    % Areas used for capacitor charge eqn Ax_y is area for cap x in region y
    c.Area = zeros(3, 3);

    diff = zeros(length(Iout),length(a));
    alpha_index = 1;
    gamma_index = 1;
    t = 2; % start time required for matlab indexing
```

current loop

```
        opt(1,:) = [inf 0 0];

            cycle.abcount = abcount;
            cycle.endcount = endcount;


                cycle.a = a;
```

Initialize function

```
                %initializations depend on parameters being swept
                init = ABinit(c,cycle,t,Iout);
                c = init.c;
```

Active Balancing Function

```
                abcalc = ABCalc(c,cycle,t);
                c = abcalc.c;
```

error calculation

```
                norm = sqrt(abcalc.diff.Vc1_diff^2 +
                                abcalc.diff.Vc3_diff^2);
                diff_Vc1 = abcalc.diff.Vc1_diff;
```

```
60                  diff_Vc3 = abcalc.diff.Vc3_diff;
61  %                 if diff(j,k) < opt(1,1)
62  %                     waveforms(1) = c;
63  %                     cycle_plot = cycle;
64  %                     opt = [diff(j,k) j k];
65  %                     alpha_index = k;
66  %                     gamma_index = j;
67  %                 elseif diff(j,k) == opt(1,1)
68  %                     waveforms(2) = c;
69  %                     opt(2,:) = [diff(j,k) j k];
70  %                 end
71
72
73
74      % end
```

75 display waveforms

```
76      % ABplot(waveforms(1),cycle_plot);
77           %Current = Iout(opt(1,2))
```

78 outputs

```
79      %%output
80      abmain.norm = norm;
81      abmain.diff_Vc1 = diff_Vc1;
82      abmain.diff_Vc3 = diff_Vc3;
83      abmain.curr_max = max(c.curr);
84      abmain.c = c;
85  %     abmain.alpha = a(alpha_index);
86  %     abmain.gamma = abcount(gamma_index);
87  %     abmain.opt = opt;
```

Active Balancing Initialization

Contents

- Set duty cycles and subperiods to active balancing
- initial voltages and currents, t0
- Output

```
function init = ABinit(c, cycle,t,Iout)
```

Set duty cycles and subperiods to active balancing

```
% adjust subperiod of q2, using effective period
% ALWAYS use 4 level N and 4 level T
c.Tab(2) = cycle.a/(c.N1-1);
c.Tab(1) = 0.5*(1-c.Tab(2)); % T1+T2+T4=1
c.Tab(3) = c.Tab(1);            % Keep T1=T4

% D2 = Def4*a*Tef1,
% where Tef1 is the fraction of the main period that is
% the effective period at Vsw
c.D(2) = c.Def4*c.Tab(2);
c.D(1) = c.Def4*c.Tab(1); % D1 = Def4*Tab1
c.D(3) = c.Def4*c.Tab(3); % Keep D1=D4
```

initial voltages and currents, t0

```
c.Vout(t-1) = c.D0*c.Vin; % Buck conversion ratio

% capacitor voltages as fraction of input voltage during previous N
Vc0(1) = c.kx(1)*c.Vin/(c.N0-1); % k1 = 1
Vc0(2) = c.kx(2)*c.Vin/(c.N0-1); % k2 = 2
Vc0(3) = c.kx(3)*c.Vin/(c.N0-1); % k3 = 3

c.Vc(:,1) = Vc0; % concatenate flying cap voltages and initial conditions
% Capacitor voltage goals
c.Vc_4L(1) = c.kx(1)*c.Vin/(c.N1-1); % k1 = 1
c.Vc_4L(2) = c.kx(2)*c.Vin/(c.N0-1); % k2 = 2
c.Vc_4L(3) = c.kx(2)*c.Vin/(c.N1-1); % k2 = 2

%Capacitor derating
c.CapC = [2.20E+00; 2.23E+00; 2.23E+00; 2.23E+00; 2.23E+00; 2.22E+00;
```

```
            2.18E+00; 2.10E+00; 2.05E+00; 1.93E+00; 1.81E+00; 1.63E+00;
            1.47E+00; 1.29E+00; 1.10E+00; 9.63E-01; 7.69E-01; 6.39E-01;
            5.48E-01; 4.31E-01; 3.89E-01; 0.347] * 1e-6;% * 2.014/2.2;
c.CapVDC = [0.00E+00; 1.00E+00; 2.00E+00; 4.00E+00; 6.30E+00; 1.00E+01;
            1.60E+01; 2.50E+01; 3.00E+01; 4.00E+01; 5.00E+01; 6.50E+01;
            8.00E+01; 1.00E+02; 1.25E+02; 1.50E+02; 2.00E+02; 2.50E+02;
            3.00E+02; 4.00E+02; 4.50E+02; 500];

% inductor current at time t0 is the
% minimum iL from the previous N operation
% <iL>-dipp/2 with <iL> = Iout
% and dipp = Vin*T0*Def0*(1-Def0)/(L*(N0-1)^2)
i0 = (Iout - c.Vin*c.T0*c.Def0*(1-c.Def0)/(2*c.L*(c.N0-1)^2));
c.curr(:,1) = i0;

% Calc Max current
c.Imax = c.N1*c.Vripple*c.Vin*8*(1/c.T)*c.num_capOut*c.cap + c.Iv;
```

Output

```
init.c = c;
init.Iout = Iout;
```

Set-up for Active Balancing Loop

Contents


```
function abcalc = ABCalc(c, cycle, t)
```

Outer Loop (time Block)

```
for n = 1:cycle.endcount % for n cycles do calculations
    % only 5 to 4 level case right now
    % if (c.N1 == c.N1 && (c.Vc(3,end) > k(2)*c.Vin/(c.N1-1) ||
            c.Vc(1,end) < k(1)*c.Vin/(c.N1-1)))
    % if the cycle index is within AB range, do AB calcs
    if(n <= cycle.abcount)

        tend = t + 2*(4-1)-1; % for indexing
                        % and saving calculated values for next cycle
        ab = ActiveBalancing(c, n, t);
        c.Vc(:,t:tend) = ab.Vc; % save calculated values
        c.curr(:,t:tend) = ab.curr; % save calculated values
        c.Vout(:,t:tend) = ab.Vout;
        t = tend + 1; %increase index


        elseif (n > cycle.abcount)
            cycle.a = 1;

            % reset subperiod to normal 4 level operation after AB
            c.Tab(2) = cycle.a/(c.N1-1);
            c.Tab(1) = cycle.a/(c.N1-1);
            c.Tab(3) = cycle.a/(c.N1-1);

            % reset duty cycle to normal 4 level operation after AB
            c.D(2) = c.D0;
            c.D(1) = c.D0;
            c.D(3) = c.D0;

```

```
            tend = t + 2*(4-1)-1; % for indexing
                                  % and saving calculated values for next cycle
            ab = ActiveBalancing(c, n, t);

            c.Vc(:,t:tend) = ab.Vc; % save calculated values
            c.curr(:,t:tend) = ab.curr; % save calculated values
            c.Vout(:,t:tend) = ab.Vout;
            t = tend + 1; %increase index

    end


end
```

calculate error on cap voltages

```
diff.Vc1_diff = abs(c.Vc_4L(1)-c.Vc(1,end));
diff.Vc3_diff = abs(c.Vc_4L(3)-c.Vc(3,end));
```

Output

```
abcalc.c = c;
abcalc.diff = diff;
```

Display

```
% %% Display
% figure
% timesteps1 = [c.D(1)*c.T
                c.Tab(1)*c.T
                c.Tab(1)*c.T+c.D(2)*c.T
                c.Tab(1)*c.T+c.Tab(2)*c.T
                c.Tab(1)*c.T+c.Tab(2)*c.T+c.D(3)*c.T
                c.Tab(1)*c.T+c.Tab(2)*c.T+c.Tab(3)*c.T];
% timeshift = c.T*ones(1,length(timesteps1));
% timesteps(1) = 0;
% index = 2;
%
% timeshift4 = c.T*ones(1,length(timesteps1));
%
% for m = 1:cycle.endcount
%
```

```
%         timesteps(index:index + (2*c.N1-3)) =
                    timesteps1 + (m-1)*timeshift4;
%         index = index + (2*c.N1-3) + 1;
%
% end
%
% Vout = c.Vout*ones(length(timesteps));
% % current = eval(c.curr(1:end));
% % voltage1 = eval(c.Vc(1,1:end));
% % voltage3 = eval(c.Vc(3,1:end));
% current = c.curr(1:end);
% voltage1 = c.Vc(1,1:end);
% voltage3 = c.Vc(3,1:end);
% outVoltage = c.Vout(1:end);
%
% hold on
%
% ylim([-40 40])
% plot(timesteps, voltage1, 'color', [0.953 0.918 0.257], 'linewidth', 2)
% plot(timesteps, voltage3,'color', [0.953 0.257 0.918], 'linewidth', 2)
% plot(timesteps, outVoltage, 'color', [0.257 0.953 0.894], 'linewidth', 2)
%
% volt4L = c.Vc_4L'*ones(1,length(timesteps));
% plot(timesteps, volt4L(1,:), 'color', [0 0 0])
% plot(timesteps, volt4L(3,:), 'color', [0 0 0])
%
% yyaxis right
% ylim([-5 35])
% plot(timesteps, current, 'color', [0.324 0.953 0.257], 'linewidth', 2)
%
%
% resize_figure(6, 1.2)
```

Sub-period Active Balancing Calculations

Contents

Active Balancing Function

```
function ab = ActiveBalancing(c,n,t)

    Vc(1,t-1) = c.Vc(1,t-1);
    Vc(2,t-1) = c.Vc(2,t-1);
    Vc(3,t-1) = c.Vc(3,t-1);
    curr(t-1) = c.curr(t-1);
    Vout(t-1) = c.Vout(t-1);

    Vc_loop = zeros(3,c.div);
    curr_loop = zeros(1,c.div);
    Vout_loop = zeros(1,c.div);
    iload = zeros(1,c.div);
    ic = zeros(1,c.div);
    dVout = zeros(1,c.div);

    % de-rated capacitor look-up table
    V_base = c.V_base;
    C_int = c.C_int;

```

On time of q4 from t0 to t1, region 1

```
    Vc_loop(:,1) = Vc(:,t-1);
    curr_loop(1) = curr(t-1);
    Vout_loop(1) = Vout(t-1);

    for j = 1:c.div
        % during on time, voltage across inductor is sum of series
```

```
27          % cap voltages - Vout (this one is specific to 4 level)
28          VL(1,1) = c.Vin-Vc_loop(3,j)-Vout_loop(j);
29
30          % change in inductor current in on time (D4*T) is
31          % di=dt*VL/L divided into smalled div
32          di(1,1) = VL(1,1)*c.D(3)*c.T/(c.div*c.L);
33
34          % update current at end of subperiod, old value + di
35          curr_loop(j+1) = curr_loop(j)+di(1,1);
36
37          % c.C1 = 3*(interp1(c.CapVDC,c.CapC,Vc_loop(1,j)));
38          % c.C3 = 3*(interp1(c.CapVDC,c.CapC,Vc_loop(3,j)));
39          % c.Cout = 4*(interp1(c.CapVDC,c.CapC,Vout_loop(j)));
40
41          % instead of interpolate, look up value from table
42
43          indexC1 = ((round(Vc_loop(1,j)*100)))+1;
44          indexC3 = ((round(Vc_loop(3,j)*100)))+1;
45          indexCout = ((round(Vout_loop(j)*100)))+1;
46
47          if indexC1 <= 0
48              indexC1 = 1;
49          end
50          if indexC3 <= 0
51              indexC3 = 1;
52          end
53          if indexCout <= 0
54              indexCout = 1;
55          end
56
57          c.C1 = c.num_cap*C_int(indexC1); % read from look-up table
58          c.C3 = c.num_cap*C_int(indexC3);
59          c.Cout = c.num_capOut*C_int(indexCout);
60
61
62          % Charge area of cap C1, C1 not charged/discharged in this region
63          Area(1,1) = 0;
64
65          dVc(1,1,1) = Area(1,1)/c.C1; % change in cap voltage dV = Q/C
66
67          % update cap voltage at end of subperiod, old value + dV
68          Vc_loop(1,j+1) = Vc_loop(1,j)+dVc(1,1,1);
```

```
69
70          % Voltage on cap C2 does not change because no current into C2
71
72          % Charge area of cap C3, triangle (0.5*b*h = 0.5*D4*T*di)
73          % and rectangle (b*h=D4*T*i_start) where i_start is current
74          % value from end of last subperiod
75          Area(3,1) = ((0.5*c.D(3)*c.T*di(1,1)+curr_loop(j)*c.D(3)*c.T))/
76                                                              c.div;
77
78          % change in cap voltage dV = Q/C
79          dVc(3,1,1) = Area(3,1)/c.C3;
80
81          % update cap voltage at end of subperiod, old value + dV
82          Vc_loop(3,j+1) = Vc_loop(3,j)+dVc(3,1,1);
83
84          iload(j) = Vout_loop(j)/c.R;
85          ic(j) = curr_loop(j)-iload(j);
86          dVout(j) = ic(j)*c.D(3)*c.T/(c.div*c.Cout);
87          Vout_loop(j+1) = Vout_loop(j) + dVout(j); % ?
88      end
89
90      % update current at end of subperiod, old value + di
91      curr(t) = curr_loop(end);
92
93      % update cap voltage at end of subperiod, old value + dV
94      Vc(1,t) = Vc_loop(1,end);
95
96      % Voltage on cap C2 does not change because no current into C2
97
98      % update cap voltage at end of subperiod, old value + dV
99      Vc(3,t) = Vc_loop(3,end);
100
101     Vout(t) = Vout_loop(end); % update
```

102 Off time of q1 t1 to t2, region 1

```
103     Vc_loop(:,1) = Vc(:,t);
104     curr_loop(1) = curr(t);
105     Vout_loop(1) = Vout(t);
106
107     for j = 1:c.div
108         % during off time, voltage across inductor is sum of
```

```
109         % series cap voltages - Vout (this one is specific to 4 level)
110         VL(2,1) = -Vout_loop(j);
111 
112         % change in inductor current in on time (Tab3-D3*T) is di=dt*VL/L
113         di(2,1) = VL(2,1)*((c.Tab(3)-c.D(3))*c.T)/(c.div*c.L);
114 
115         % update current at end of subperiod, old value + di
116         curr_loop(j+1) = curr_loop(j)+di(2,1);
117 
118         % c.Cout = 4*(interp1(c.CapVDC,c.CapC,Vout_loop(j)));
119         % instead of interpolate, look up value from table
120         indexCout =((round(Vout_loop(j)*100)))+1;
121 
122         if indexCout <= 0
123             indexCout = 1;
124         end
125         c.Cout = c.num_capOut*C_int(indexCout);
126 
127 
128         iload(j) = Vout_loop(j)/c.R;
129         ic(j) = curr_loop(j)-iload(j);
130         dVout(j) = (ic(j)*(c.Tab(3)-c.D(3))*c.T)/(c.div*c.Cout);
131         Vout_loop(j+1) = Vout_loop(j) + dVout(j); % ?
132 
133     end
134 
135     % update current at end of subperiod, old value + di
136     curr(t+1) = curr_loop(end);
137 
138     % cap voltage does not change in off time (specific to lowest D range)
139     Vc(1,t+1) = Vc_loop(1,end);
140 
141     % cap voltage does not change in off time (specific to lowest D range)
142     Vc(3,t+1) = Vc_loop(3,end);
143 
144     Vout(t+1) = Vout_loop(end);
```

145 **On time of q2 from t2 to t3, region 2

```
146     % doing a half step here, update voltage, do other half
147     Vc_loop(:,1) = Vc(:,t+1);
148     curr_loop(1) = curr(t+1);
```

```
149     Vout_loop(1) = Vout(t+1);
150
151     for j = 1:c.div
152
153          % during on time, voltage across inductor is sum of series
154          % cap voltages - Vout (this one is specific to 4 level)
155         VL(1,2) = Vc_loop(3,j)-Vc_loop(1,j)-Vout_loop(j);
156
157         % change in inductor current in on time (D2*T) is di=dt*VL/L
158         di(1,2) = VL(1,2)*c.D(2)*c.T/(c.div*c.L);
159
160         % update current at end of subperiod, old value + di
161         curr_loop(j+1) = curr_loop(j)+di(1,2);
162
163         % c.C1 = 3*(interp1(c.CapVDC,c.CapC,Vc_loop(1,j)));
164         % c.C3 = 3*(interp1(c.CapVDC,c.CapC,Vc_loop(3,j)));
165         % c.Cout = 4*(interp1(c.CapVDC,c.CapC,Vout_loop(j)));
166
167         % instead of interpolate, look up value from table
168
169         indexC1 = ((round(Vc_loop(1,j)*100)))+1;
170         indexC3 = ((round(Vc_loop(3,j)*100)))+1;
171         indexCout = ((round(Vout_loop(j)*100)))+1;
172         if indexC1 <= 0
173             indexC1 = 1;
174         end
175         if indexC3 <= 0
176             indexC3 = 1;
177         end
178         if indexCout <= 0
179             indexCout = 1;
180         end
181
182         c.C1 = c.num_cap*C_int(indexC1); % read from look-up table
183         c.C3 = c.num_cap*C_int(indexC3);
184         c.Cout = c.num_capOut*C_int(indexCout);
185
186         % charge area of cap C1, triangle (0.5*b*h = 0.5*D2*T*di)
187         % and rectangle (b*h=D2*T*i_start) where i_start is current
188         % value from end of last subperiod
189         Area(1,2) = (0.5*c.D(2)*c.T*di(1,2)+curr_loop(j)*c.D(2)*c.T)/c.div;
190
```

```
191         % change in cap voltage dV = Q/C
192         dVc(1,2,1) = Area(1,2)/c.C1;
193
194         % update cap voltage at end of subperiod, old value + dV
195         Vc_loop(1,j+1) = Vc_loop(1,j)+dVc(1,2,1);
196
197         % Voltage on cap C2 does not change because no current into C2
198
199         % discharge area of cap C3, triangle (0.5*b*h = 0.5*D2*T*di)
200         % and rectangle (b*h=D2*T*i_start) where i_start is current
201         % value from end of last subperiod
202         Area(3,2) = -((0.5*c.D(2)*c.T*di(1,2)+curr_loop(j)*c.D(2)*c.T))/
203                                                                 c.div;
204
205         % change in cap voltage dV = Q/C
206         dVc(3,2,1) = Area(3,2)/c.C3;
207
208         % update cap voltage at end of subperiod, old value + dV
209         Vc_loop(3,j+1) = Vc_loop(3,j)+dVc(3,2,1);
210
211         iload(j) = Vout_loop(j)/c.R;
212         ic(j) = curr_loop(j)-iload(j);
213         dVout(j) = ic(j)*c.D(2)*c.T/(c.div*c.Cout);
214         Vout_loop(j+1) = Vout_loop(j) + dVout(j); %
215     end
216
217     % update current at end of subperiod, old value + di
218     curr(t+2) = curr_loop(end);
219
220     % update cap voltage at end of subperiod, old value + dV
221     Vc(1,t+2) = Vc_loop(1,end);
222
223     % Voltage on cap C2 does not change because no current into C2
224
225     % update cap voltage at end of subperiod, old value + dV
226     Vc(3,t+2) = Vc_loop(3,end);
227
228     Vout(t+2) = Vout_loop(end); % update
```

229 Off time of q2 t3 to t4, region 2

```
230     Vc_loop(:,1) = Vc(:,t+2);
```

```
231     curr_loop(1) = curr(t+2);
232     Vout_loop(1) = Vout(t+2);
233
234     for j = 1:c.div
235
236         % during off time, voltage across inductor is sum of series
237         % cap voltages - Vout (this one is specific to 4 level)
238         VL(2,2) = -Vout_loop(j);
239
240         % change in inductor current in off time (Tab2-D2*T) is di=dt*VL/L
241         di(2,2) = VL(2,2)*((c.Tab(2)-c.D(2))*c.T)/(c.div*c.L);
242
243          % update current at end of subperiod, old value + di
244         curr_loop(j+1) = curr_loop(j)+di(2,2);
245
246         % c.Cout = 4*(interp1(c.CapVDC,c.CapC,Vout_loop(j)));
247         % instead of interpolate, look up value from table
248         indexCout = ((round(Vout_loop(j)*100)))+1;
249
250         if indexCout <= 0
251             indexCout = 1;
252         end
253         c.Cout = c.num_capOut*C_int(indexCout);
254
255
256         iload(j) = Vout_loop(j)/c.R;
257         ic(j) = curr_loop(j)-iload(j);
258         dVout(j) = (ic(j)*(c.Tab(2)-c.D(2))*c.T)/(c.div*c.Cout);
259         Vout_loop(j+1) = Vout_loop(j) + dVout(j); %
260
261     end
262
263     curr(t+3) = curr_loop(end);
264
265     % cap voltage does not change in off time (specific to lowest D range)
266     Vc(1,t+3) = Vc_loop(1,end);
267
268     % cap voltage does not change in off time (specific to lowest D range)
269     Vc(3,t+3) = Vc_loop(3,end);
270
271     Vout(t+3) = Vout_loop(end);
```

On time of q3 from t4 to t5, region 3

```
    Vc_loop(:,1) = Vc(:,t+3);
    curr_loop(1) = curr(t+3);
    Vout_loop(1) = Vout(t+3);

    for j = 1:c.div

        % during on time, voltage across inductor is sum of series
        % cap voltages - Vout (this one is specific to 4 level)
        VL(1,3) = Vc_loop(1,j)-Vout_loop(j);

        % change in inductor current in on time (D1*T) is di=dt*VL/L
        di(1,3) = VL(1,3)*c.D(1)*c.T/(c.div*c.L);

        % update current at end of subperiod, old value + di
        curr_loop(j+1) = curr_loop(j)+di(1,3);

        % c.C1 = 3*(interp1(c.CapVDC,c.CapC,Vc_loop(1,j)));
        % c.C3 = 3*(interp1(c.CapVDC,c.CapC,Vc_loop(3,j)));
        % c.Cout = 4*(interp1(c.CapVDC,c.CapC,Vout_loop(j)));

        % instead of interpolate, look up value from table

        indexC1 = ((round(Vc_loop(1,j)*100)))+1;
        indexC3 = ((round(Vc_loop(3,j)*100)))+1;
        indexCout = ((round(Vout_loop(j)*100)))+1;
        if indexC1 <= 0
            indexC1 = 1;
        end
        if indexC3 <= 0
            indexC3 = 1;
        end
        if indexCout <= 0
            indexCout = 1;
        end

        c.C1 = c.num_cap*C_int(indexC1); % read from look-up table
        c.C3 = c.num_cap*C_int(indexC3);
        c.Cout = c.num_capOut*C_int(indexCout);
```

```
313         % discharge area of cap C1, triangle (0.5*b*h = 0.5*D1*T*di)
314         % and rectangle (b*h=D1*T*i_start) where i_start is current
315         % value from end of last subperiod
316         Area(1,3) = -((0.5*c.D(1)*c.T*di(1,3)+curr_loop(j)*c.D(1)*c.T))/
317                                                                 c.div;
318
319         % change in cap voltage dV = Q/C
320         dVc(1,3,1) = Area(1,3)/c.C1;
321
322         % update cap voltage at end of subperiod, old value + dV
323         Vc_loop(1,j+1) = Vc_loop(1,j)+dVc(1,3,1);
324
325         % Voltage on cap C2 does not change because no current into C2
326
327         % Charge area of cap C3, C3 not charged/discharged in this region
328         Area(3,3) = 0;
329
330         % change in cap voltage dV = Q/C
331         dVc(3,3,1) = Area(3,3)/c.C3;
332
333         % update cap voltage at end of subperiod, old value + dV
334         Vc_loop(3,j+1) = Vc_loop(3,j)+dVc(3,3,1);
335
336         iload(j) = Vout_loop(j)/c.R;
337         ic(j) = curr_loop(j)-iload(j);
338         dVout(j) = ic(j)*c.D(1)*c.T/(c.div*c.Cout);
339         Vout_loop(j+1) = Vout_loop(j) + dVout(j); % ?
340     end
341
342     % update current at end of subperiod, old value + di
343     curr(t+4) = curr_loop(end);
344
345     % update cap voltage at end of subperiod, old value + dV
346     Vc(1,t+4) = Vc_loop(1,end);
347
348     % Voltage on cap C2 does not change because no current into C2
349
350      % update cap voltage at end of subperiod, old value + dV
351     Vc(3,t+4) = Vc_loop(3,end);
352
353     Vout(t+4) = Vout_loop(end); % update
```

Off time of q3 t5 to t6, region 3

```
    Vc_loop(:,1) = Vc(:,t+4);
    curr_loop(1) = curr(t+4);
    Vout_loop(1) = Vout(t+4);

    for j = 1:c.div

        % during off time, voltage across inductor is sum of series
        % cap voltages - Vout (this one is specific to 4 level)
        VL(2,3) = -Vout_loop(j);

        % change in inductor current in on time (Tab1-D1*T) is di=dt*VL/L
        di(2,3) = VL(2,3)*((c.Tab(1)-c.D(1))*c.T)/(c.div*c.L);

        % update current at end of subperiod, old value + di
        curr_loop(j+1) = curr_loop(j)+di(2,3);

        % c.Cout = 4*(interp1(c.CapVDC,c.CapC,Vout_loop(j)));

        % instead of interpolate, look up value from table
        indexCout = ((round(Vout_loop(j)*100)))+1;

        if indexCout <= 0
            indexCout = 1;
        end
        c.Cout = c.num_capOut*C_int(indexCout);

        iload(j) = Vout_loop(j)/c.R;
        ic(j) = curr_loop(j)-iload(j);
        dVout(j) = (ic(j)*(c.Tab(1)-c.D(1))*c.T)/(c.div*c.Cout);
        Vout_loop(j+1) = Vout_loop(j) + dVout(j); % ?

    end

    % update current at end of subperiod, old value + di
    curr(t+5) = curr_loop(end);

     % cap voltage does not change in off time (specific to lowest D range)
    Vc(1,t+5) = Vc_loop(1,end);

    % cap voltage does not change in off time (specific to lowest D range)
```

```
395     Vc(3,t+5) = Vc_loop(3,end);
396
397     Vout(t+5) = Vout_loop(end);
```

398 Output

```
399 ab.Vc = Vc(:,t:t+5);
400 ab.curr = curr(:,t:t+5);
401 ab.Vout = Vout(:,t:t+5);
```

Plot Active Balancing Waveforms

Display

```
function display = ABplot(c,cycle)
    figure
    timesteps1 = [c.D(1)*c.T
                    c.Tab(1)*c.T
                    c.Tab(1)*c.T+c.D(2)*c.T
                    c.Tab(1)*c.T+c.Tab(2)*c.T
                    c.Tab(1)*c.T+c.Tab(2)*c.T+c.D(3)*c.T
                    c.Tab(1)*c.T+c.Tab(2)*c.T+c.Tab(3)*c.T];
    timeshift = c.T*ones(1,length(timesteps1));
    timesteps(1) = 0;
    index = 2;

    timeshift4 = c.T*ones(1,length(timesteps1));

    for m = 1:cycle.endcount

        timesteps(index:index + (2*c.N1-3))= timesteps1 + (m-1)*timeshift4;
        index = index + (2*c.N1-3) + 1;

    end

    Vout = c.Vout*ones(length(timesteps));
    % current = eval(c.curr(1:end));
    % voltage1 = eval(c.Vc(1,1:end));
    % voltage3 = eval(c.Vc(3,1:end));
    current = c.curr(1:end);
    voltage1 = c.Vc(1,1:end);
    voltage3 = c.Vc(3,1:end);
    outVoltage = c.Vout(1:end);

    hold on

    ylim([-40 100])
    plot(timesteps, voltage1, 'color', [0.953 0.918 0.257], 'linewidth', 2)
    plot(timesteps, voltage3,'color', [0.953 0.257 0.918], 'linewidth', 2)
    plot(timesteps, outVoltage, 'color', [0.257 0.953 0.894],
                                                        'linewidth', 2)
```

```
39
40      volt4L = c.Vc_4L'*ones(1,length(timesteps));
41      plot(timesteps, volt4L(1,:), 'color', [0 0 0])
42      plot(timesteps, volt4L(3,:), 'color', [0 0 0])
43
44      yyaxis right
45      ylim([-10 35])
46      plot(timesteps, current, 'color', [0.324 0.953 0.257], 'linewidth', 2)
47
48
49      resize_figure(6, 1.2)
```

Plot Active Balancing Parameters

Display

```
2  function disp = sys_plot(opt)
3      figure
4      plot(opt(:,1), opt(:,6), '-*')
5      title('Alpha','FontSize',16,'FontName','Times New Roman');
6      grid on
7      xlabel('Average Output Current [A]','FontSize',16,
8              'FontName','Times New Roman');
9      ylabel('\alpha','FontSize',16,'FontName','Times New Roman');
10
11     figure
12     plot(opt(:,1), opt(:,7), '-*')
13     title('Gamma','FontSize',16,'FontName','Times New Roman');
14     grid on
15     xlabel('Average Output Current [A]','FontSize',16,
16             'FontName','Times New Roman');
17     ylabel('\gamma [cycles] ','FontSize',16,'FontName','Times New Roman');
```

Appendix B

Five-level FCML Hardware Prototype Circuit Schematic and PCB Layout

Included here for reference are the circuit schematic and PCB layout for the 5-level FCML prototype that was built for testing dynamic level selection to maintain wide-range ZVS.

Schematic

Below, are the circuit schematics for the 5-level FCML prototype.

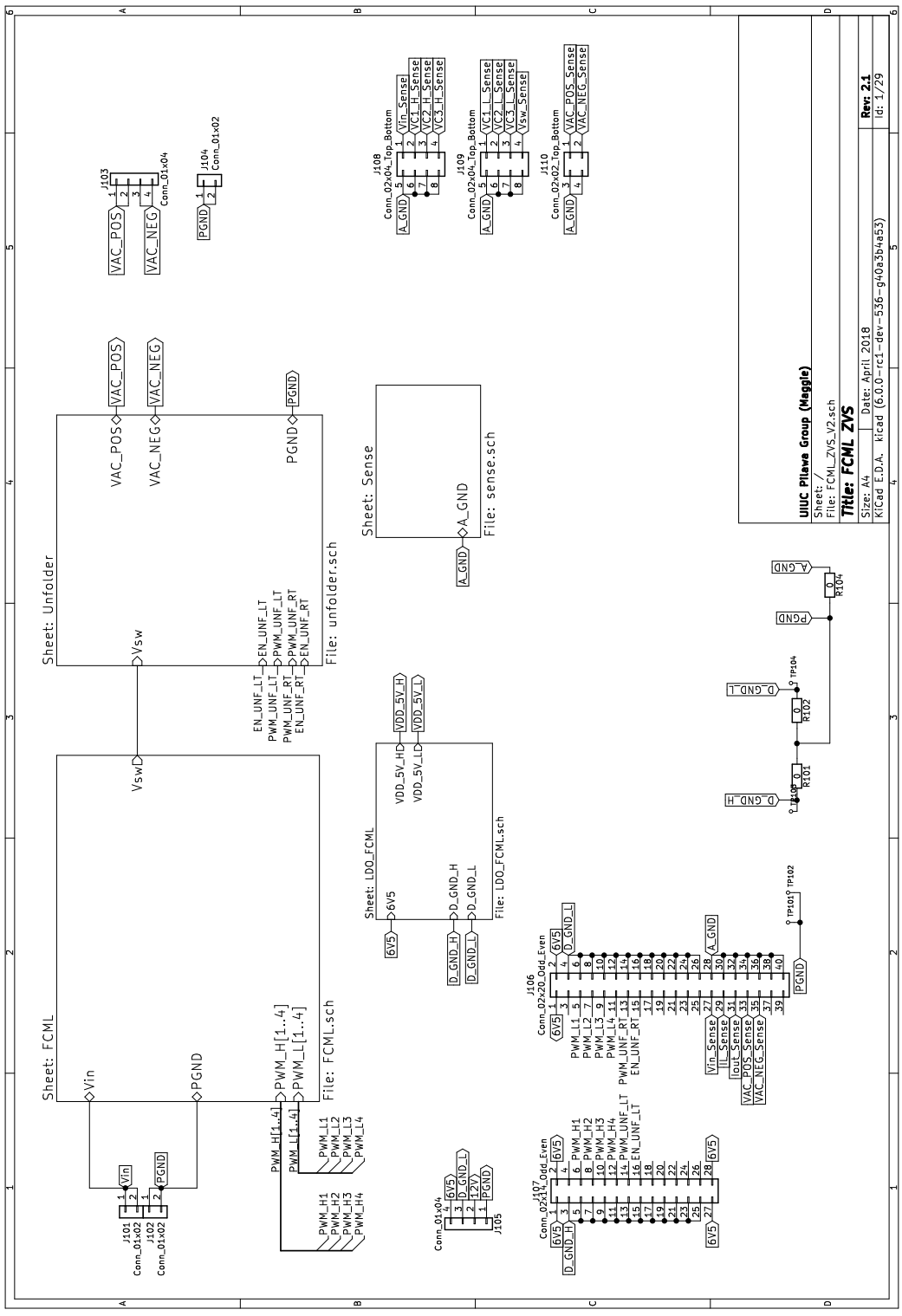


Figure B.1: Top level circuit schematic for the 5-level FCML prototype.

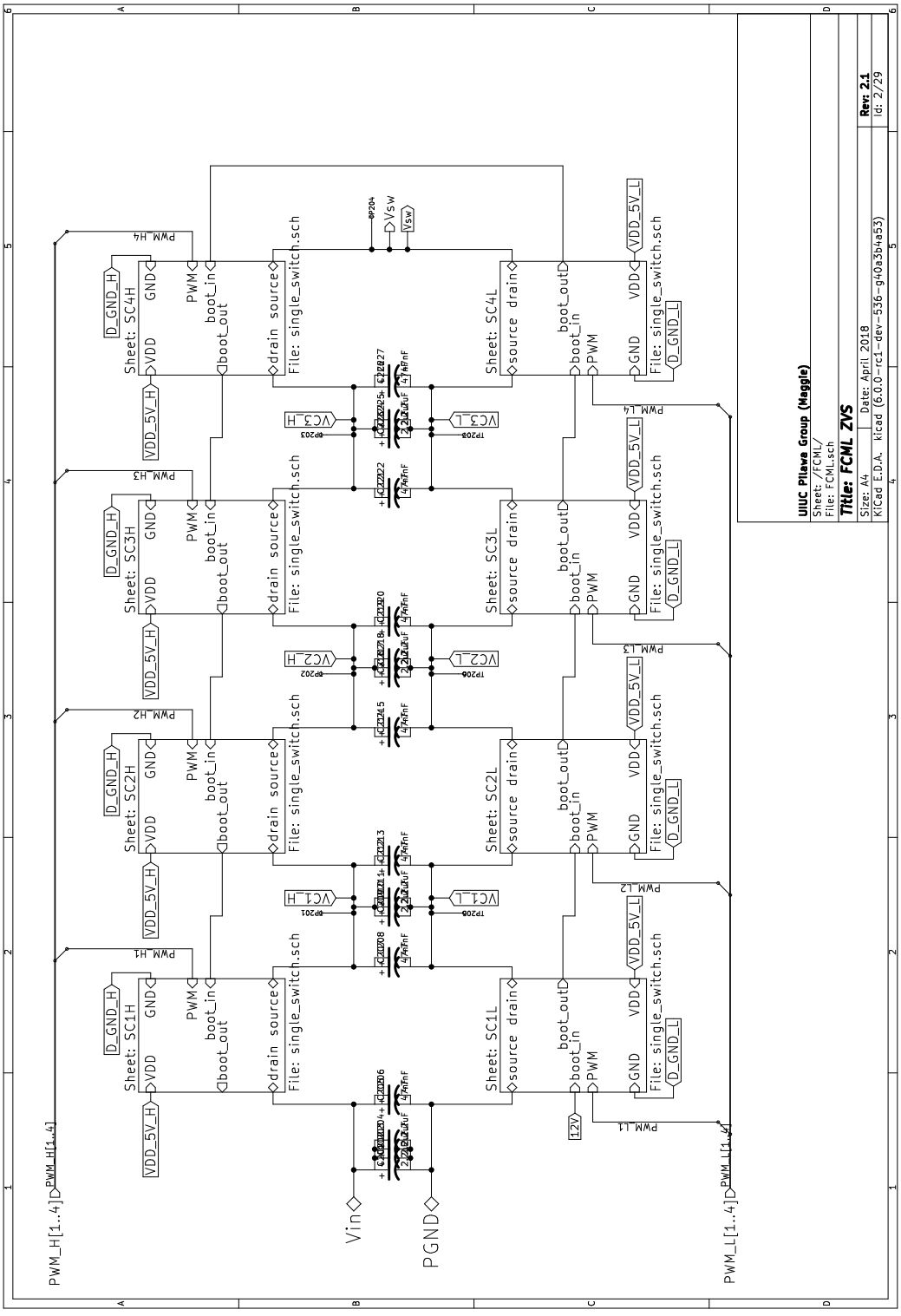


Figure B.2: Circuit schematic for the 5-level FCML power stage.

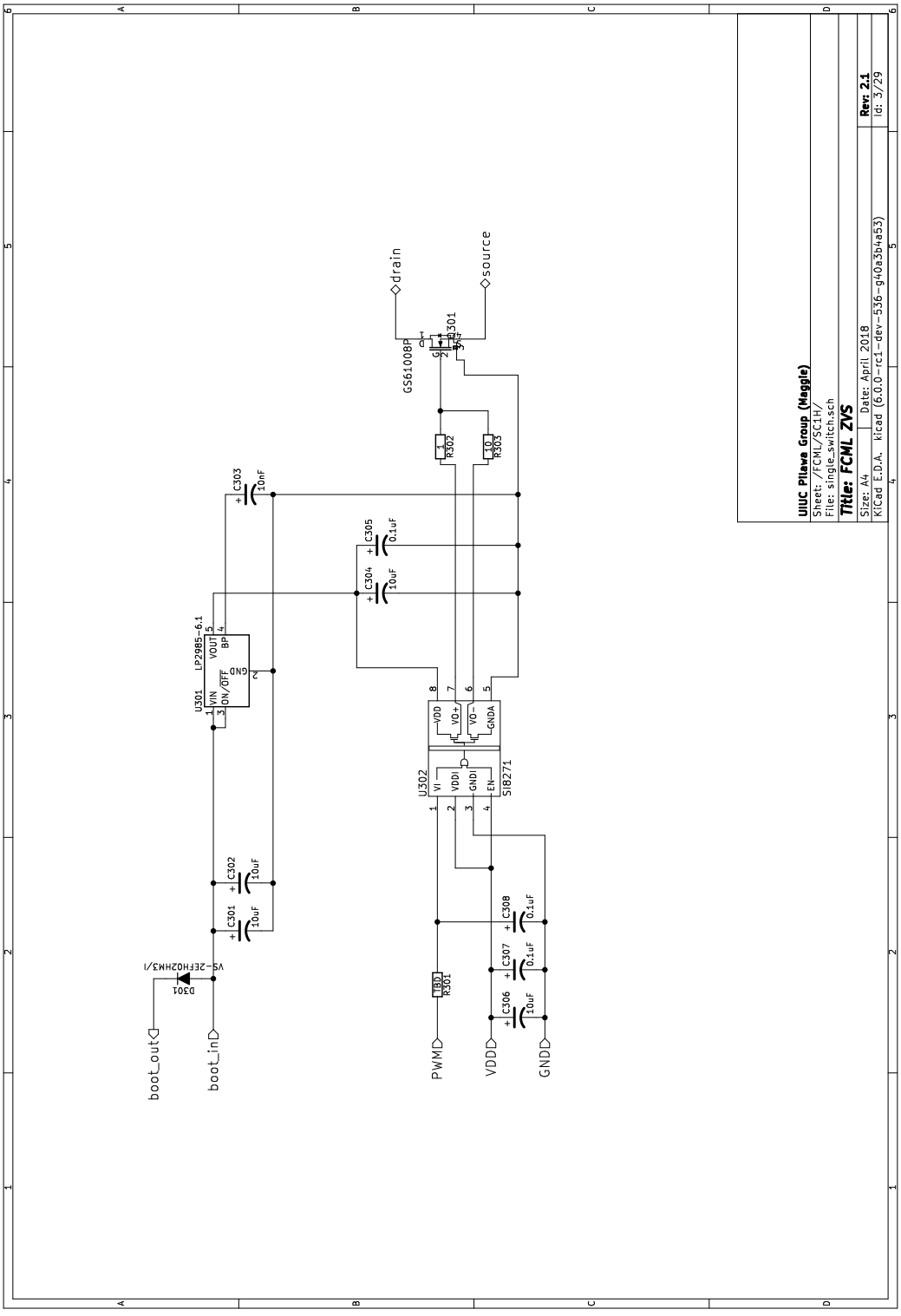


Figure B.3: Circuit schematic for a high-side switch including gate driver.

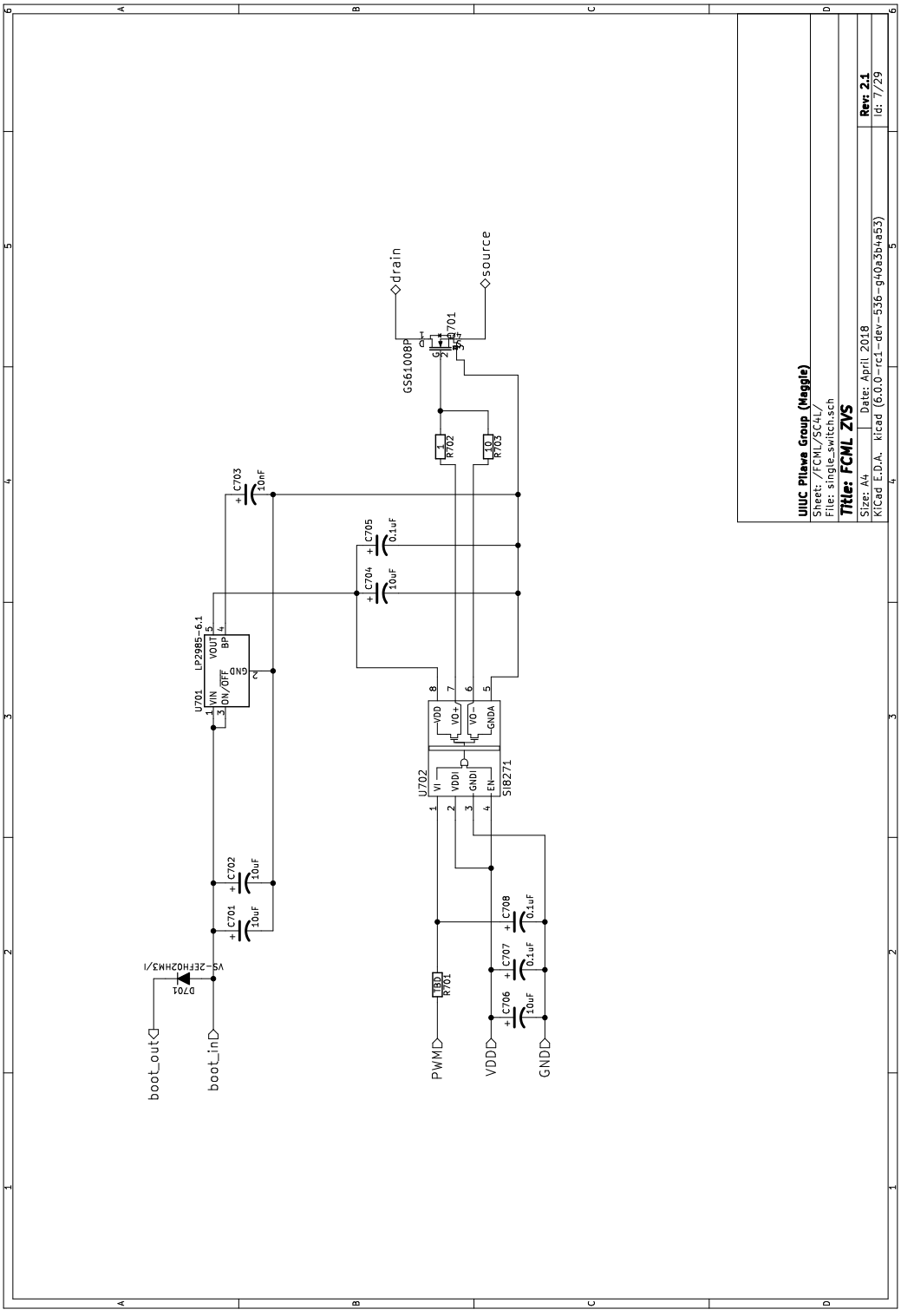


Figure B.4: Circuit schematic for a low-side switch including gate driver.

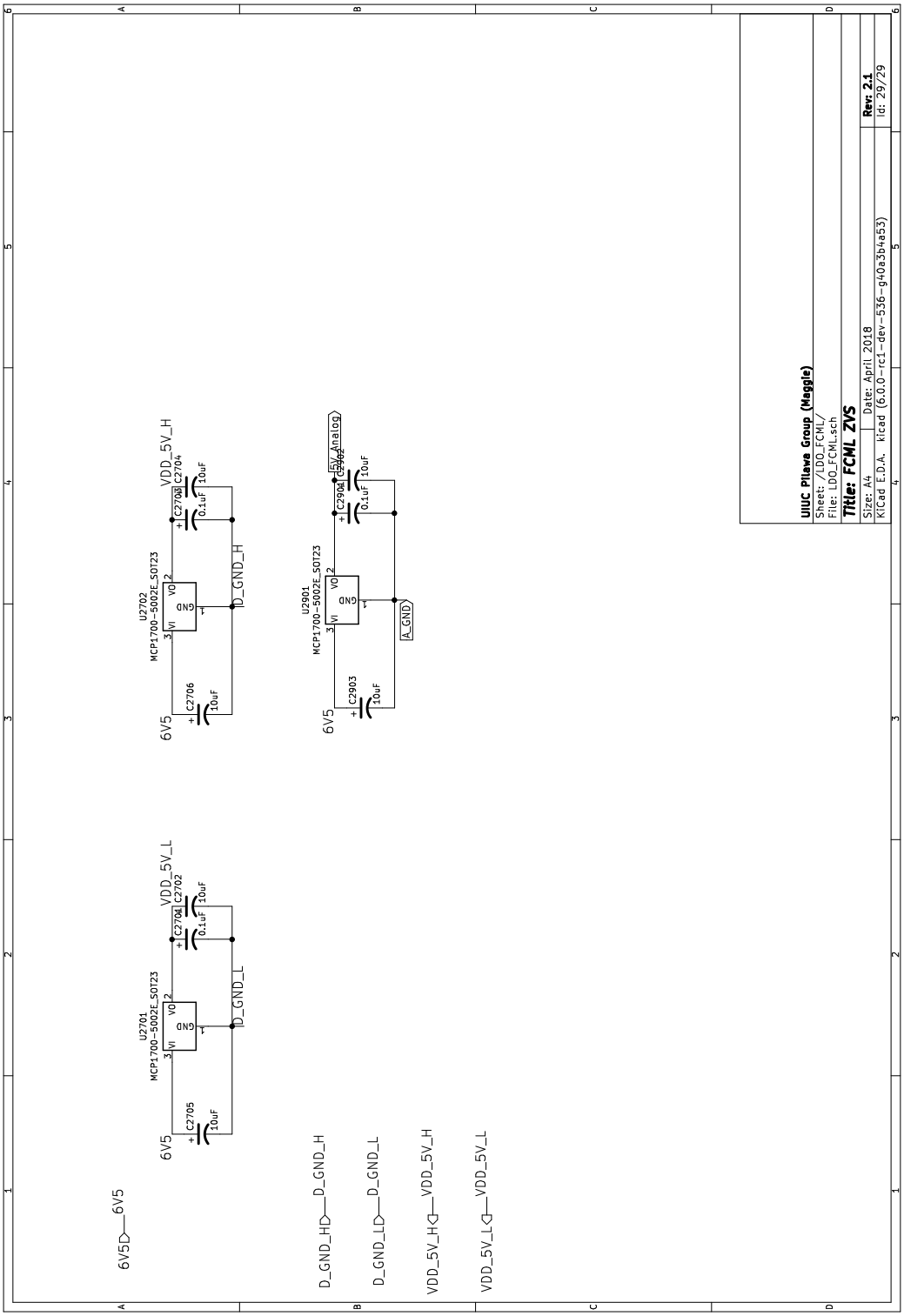


Figure B.5: Circuit schematic for the LDOs.

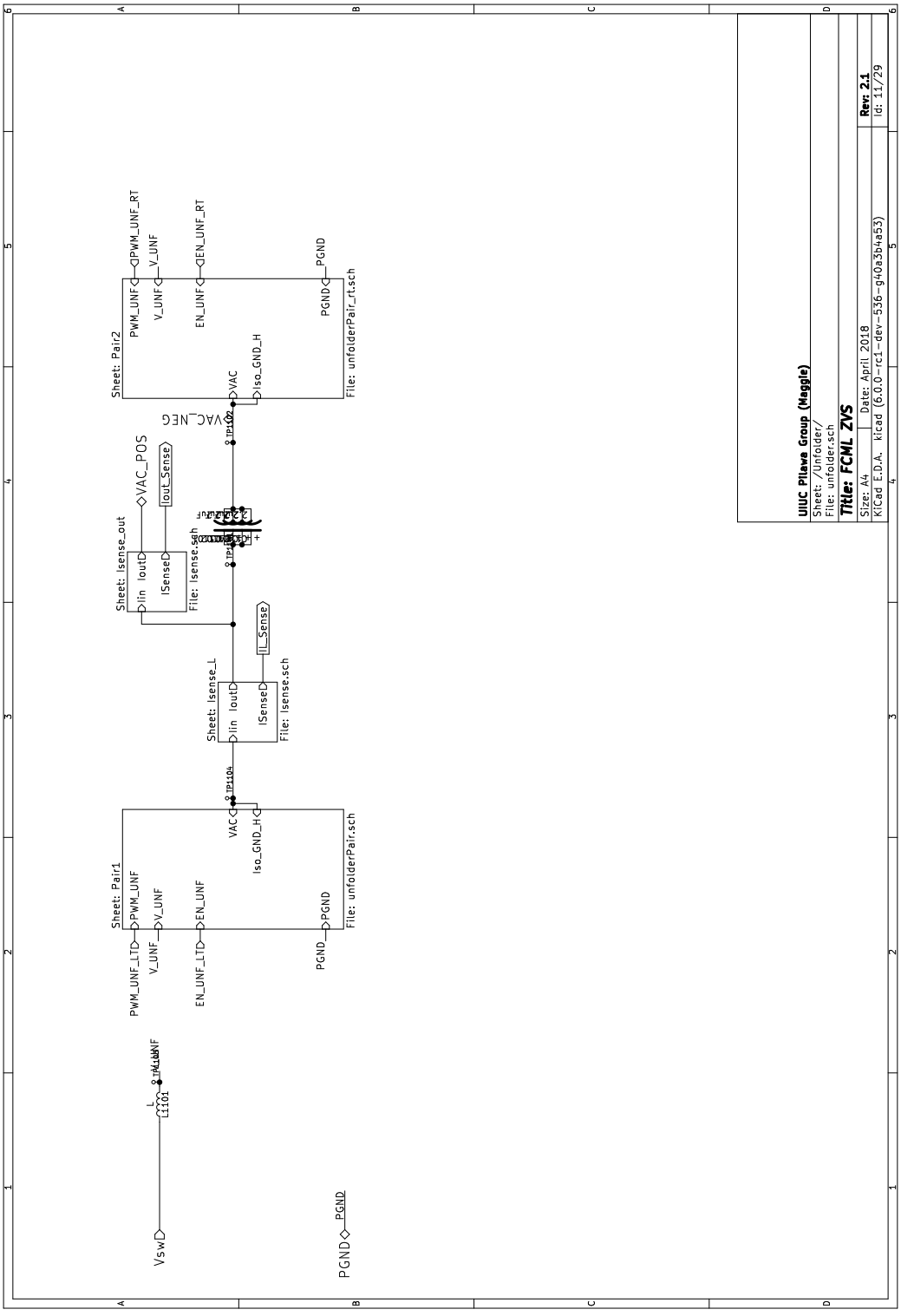


Figure B.6: Circuit schematic for the (unused) unfolder stage.

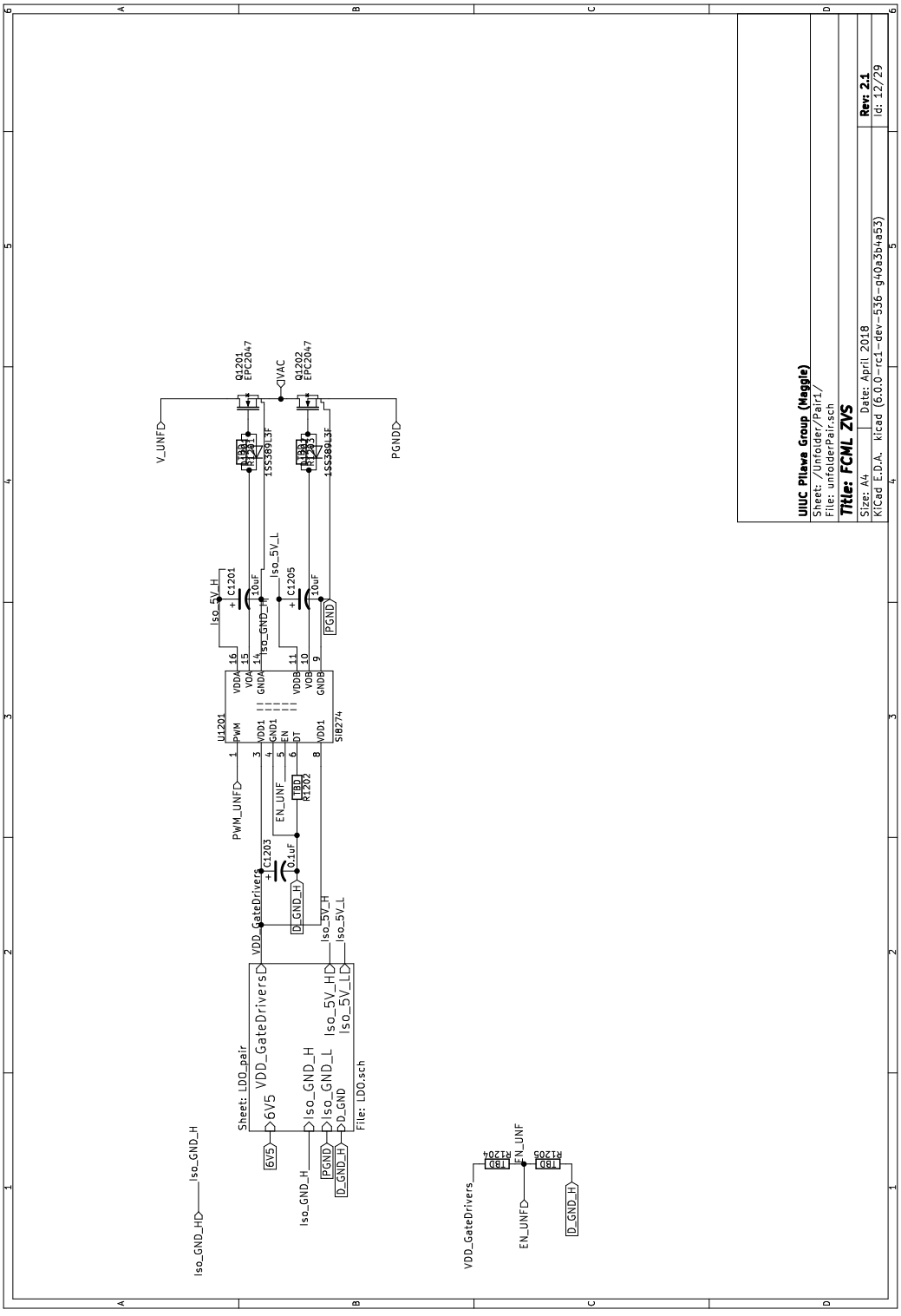


Figure B.7: Circuit schematic for the switch pairs of the unfolder stage.

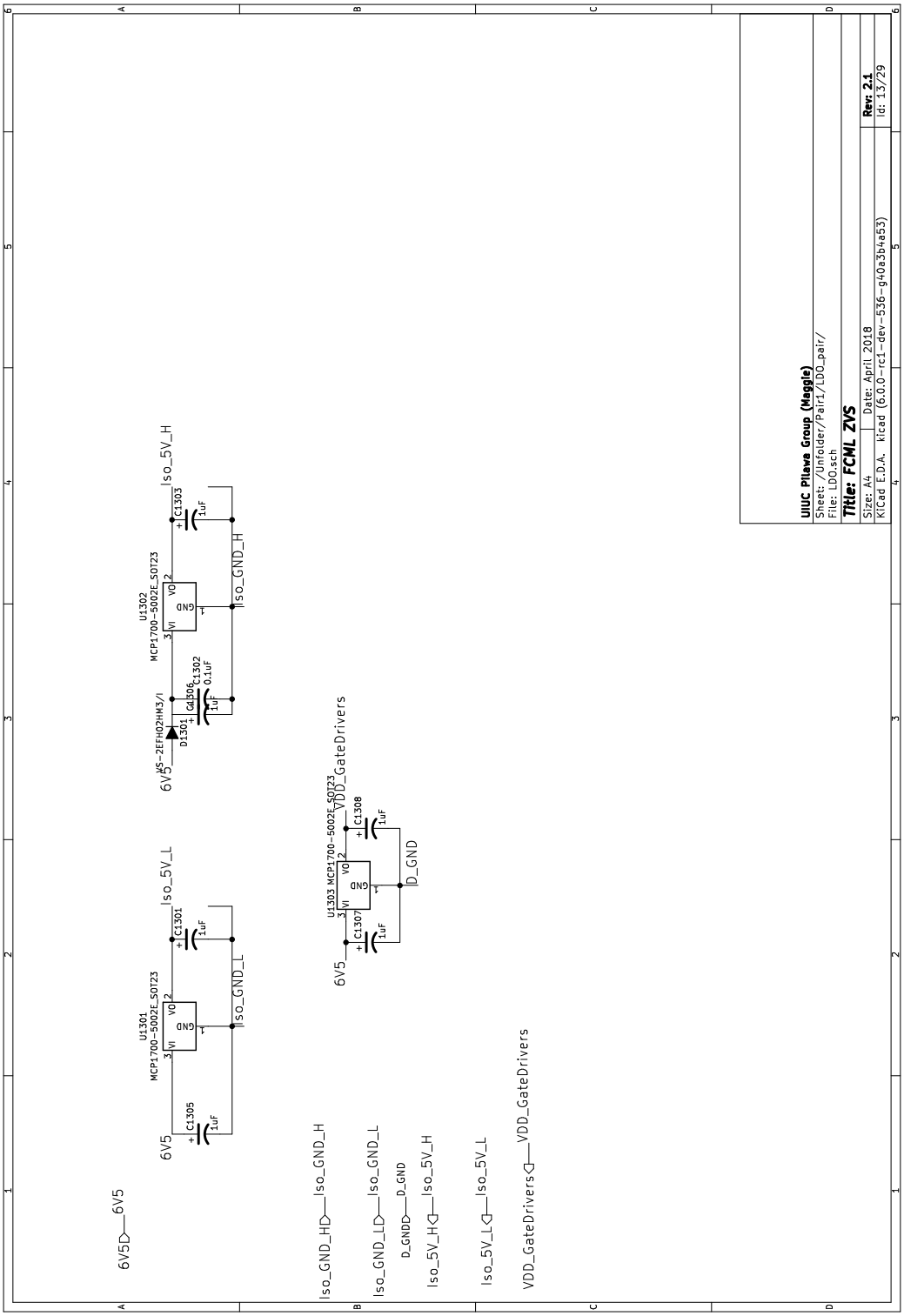


Figure B.8: Circuit schematic for the LDOs used for the unfolder stage.

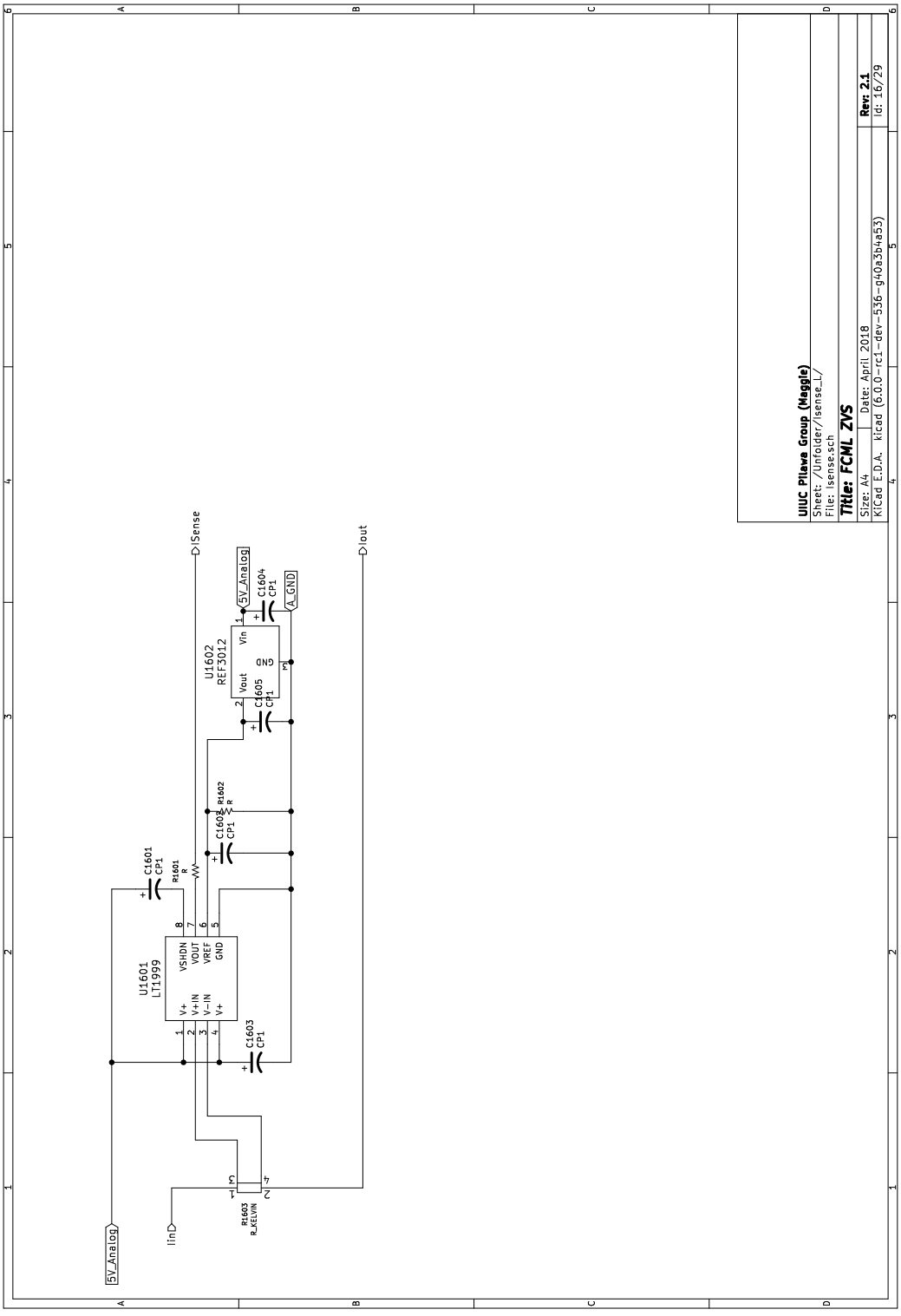


Figure B.9: Circuit schematic for current sensing.

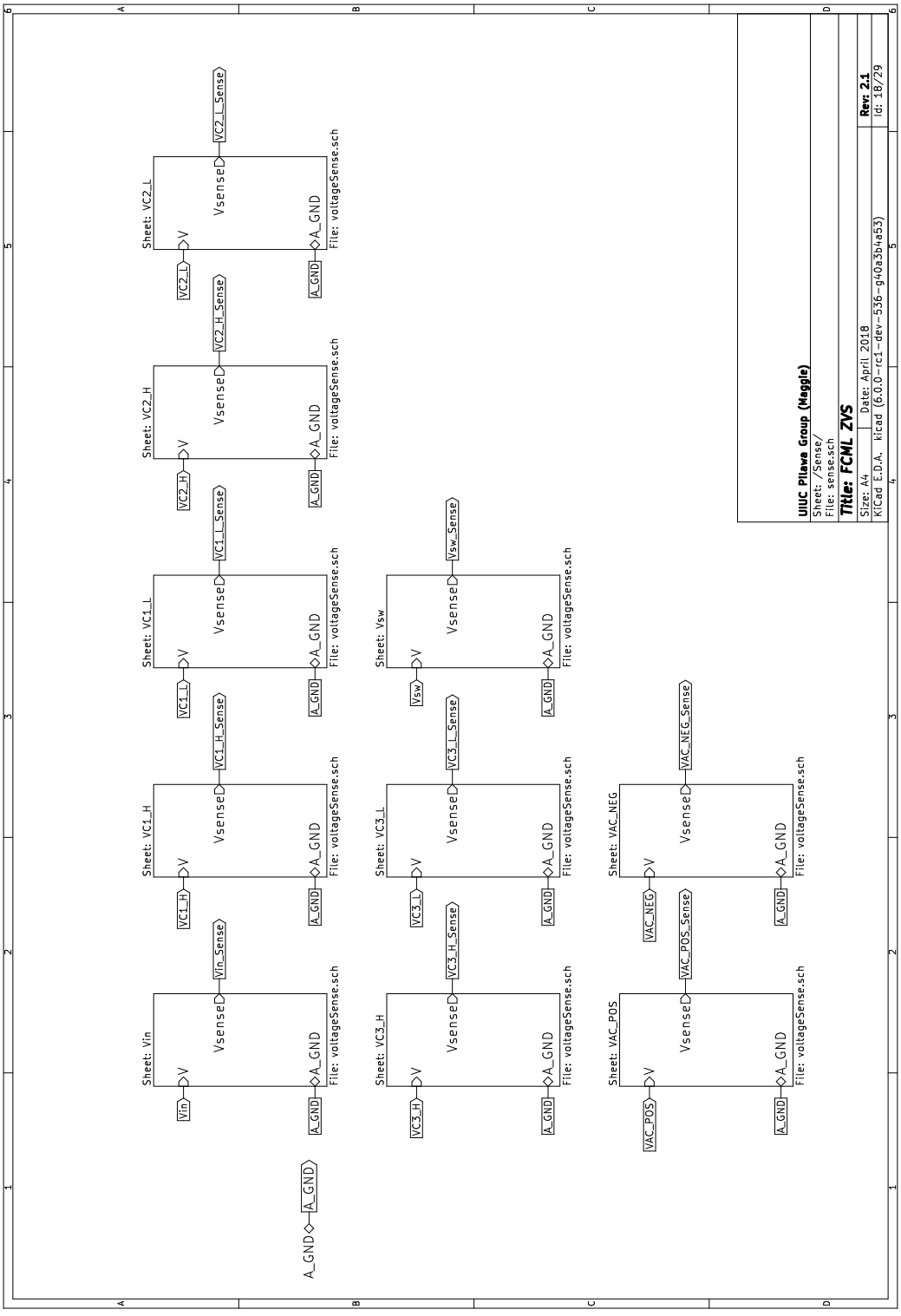


Figure B.10: Circuit schematic for voltage sensing.

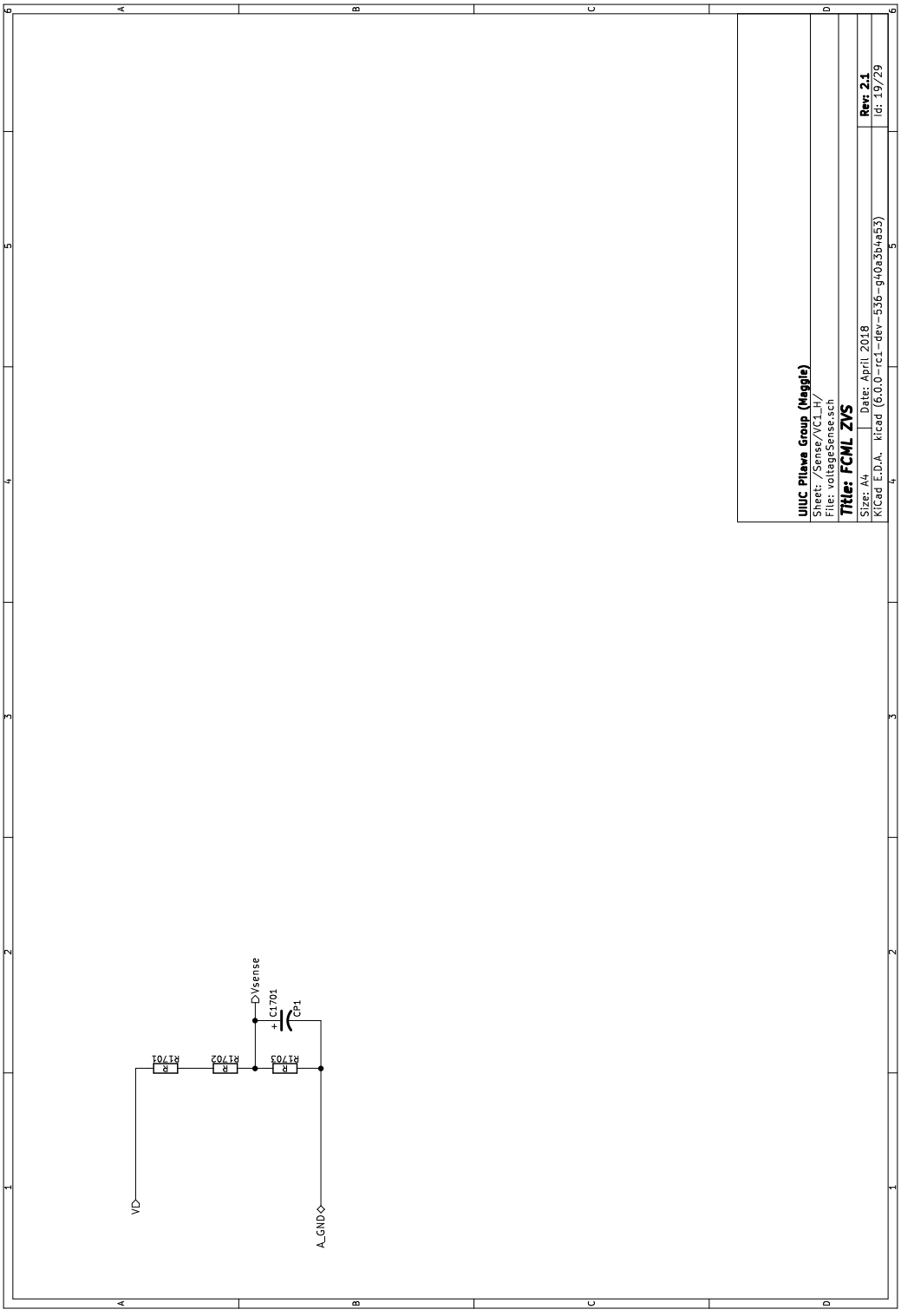


Figure B.11: Circuit schematic for a voltage sensing network.

PCB Layout

Below, are the PCB layers for the 5-level FCML prototype.

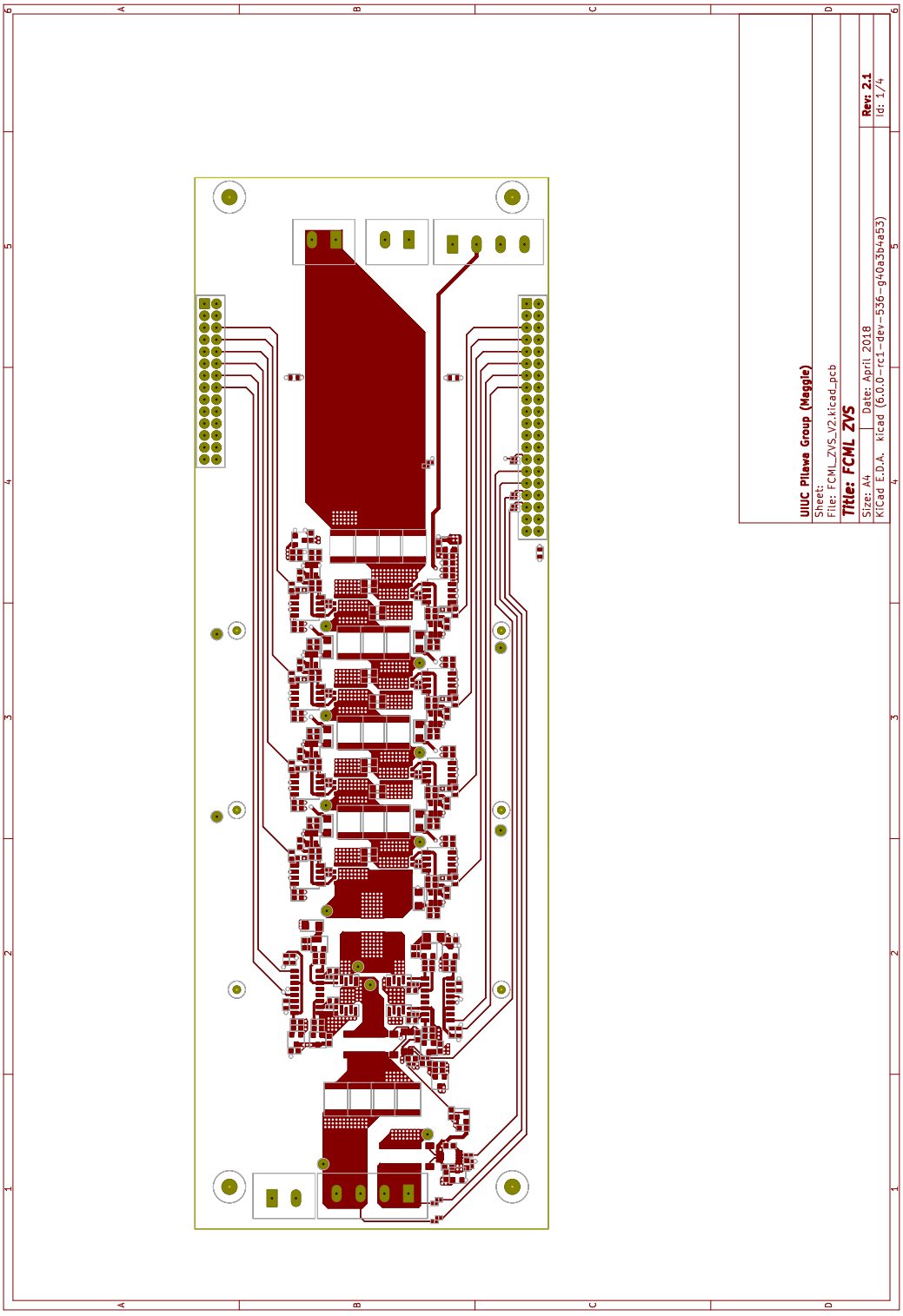


Figure B.12: Top layer of PCB.

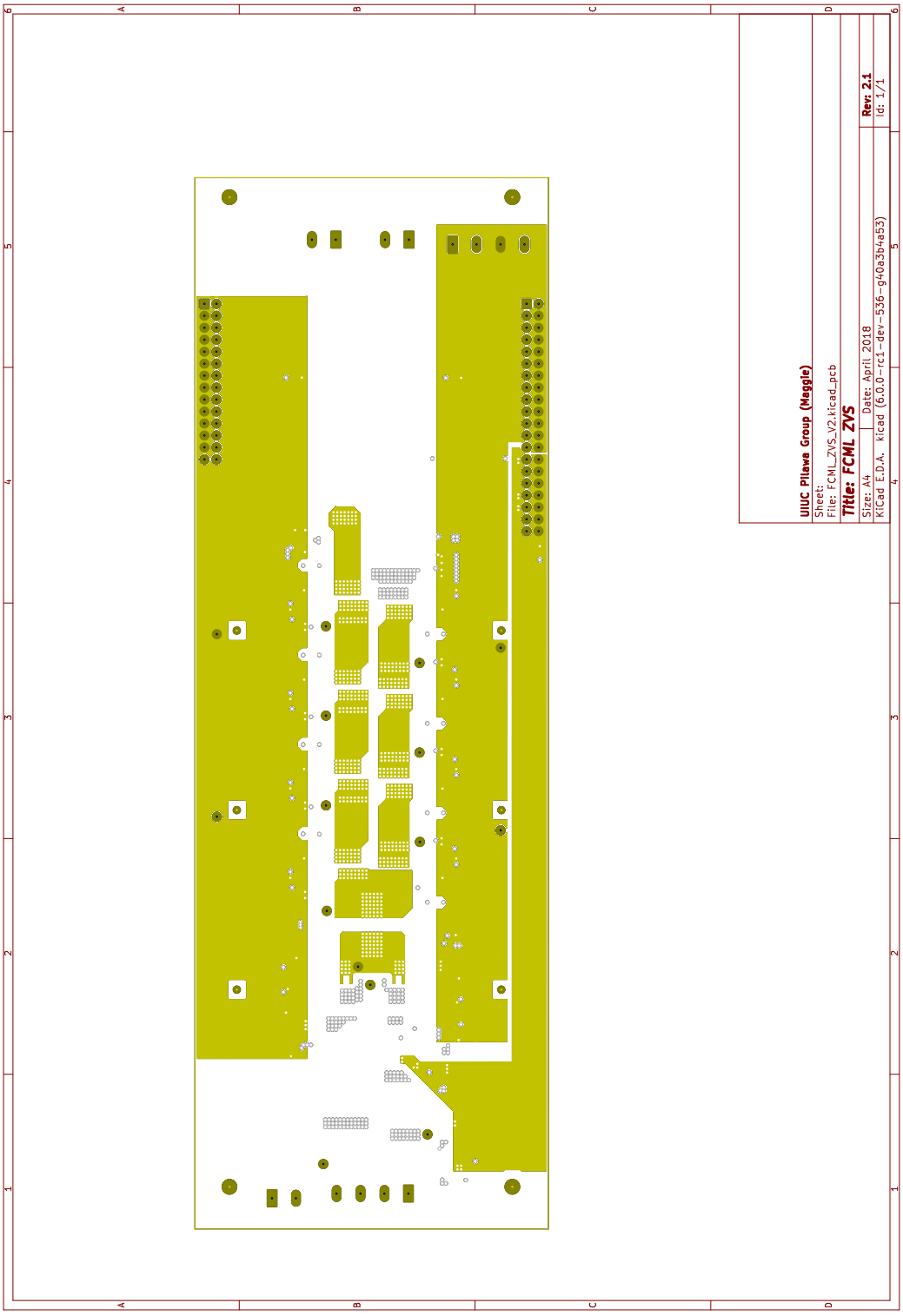


Figure B.13: First inner layer of PCB.

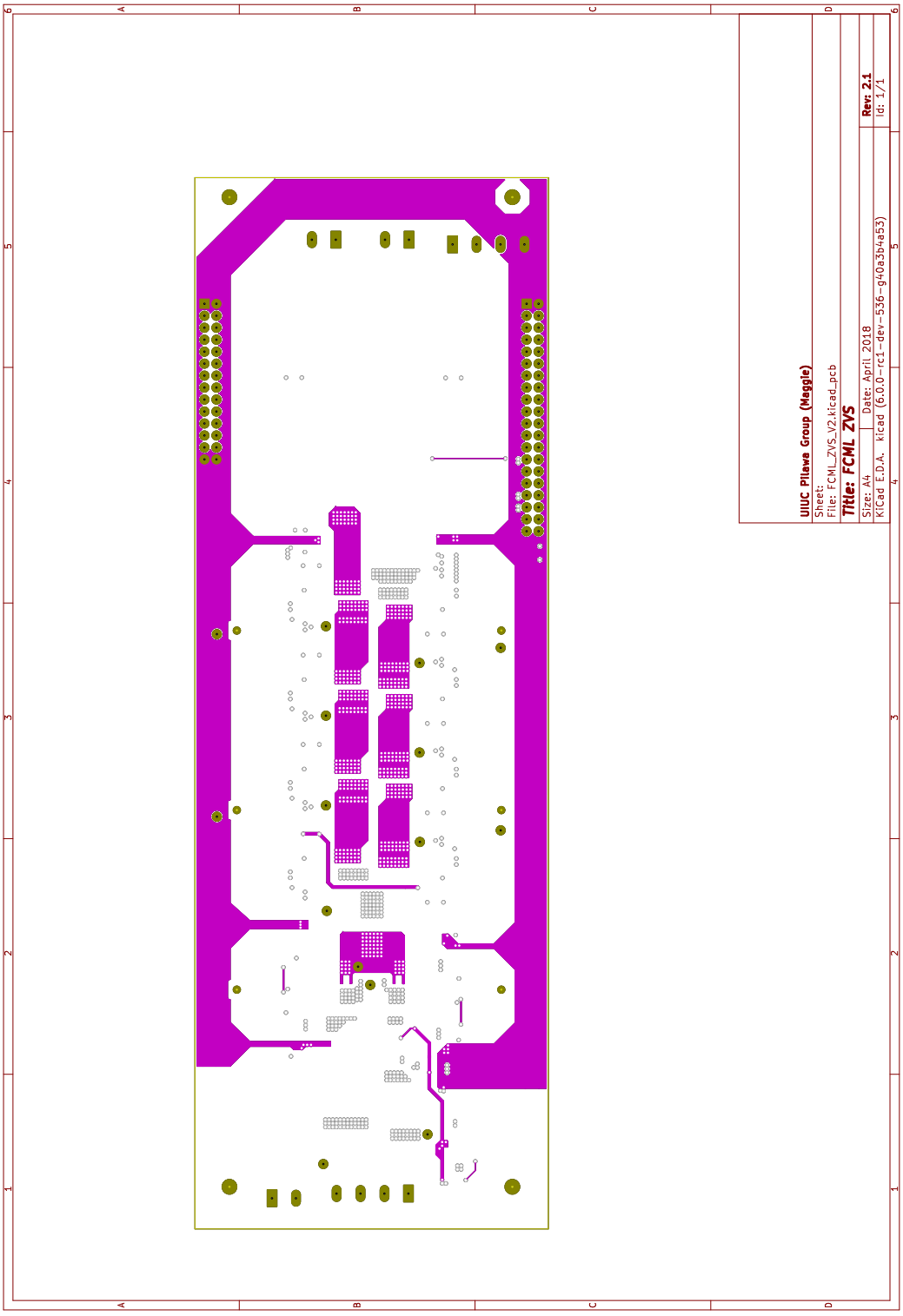


Figure B.14: Second inner layer of PCB.

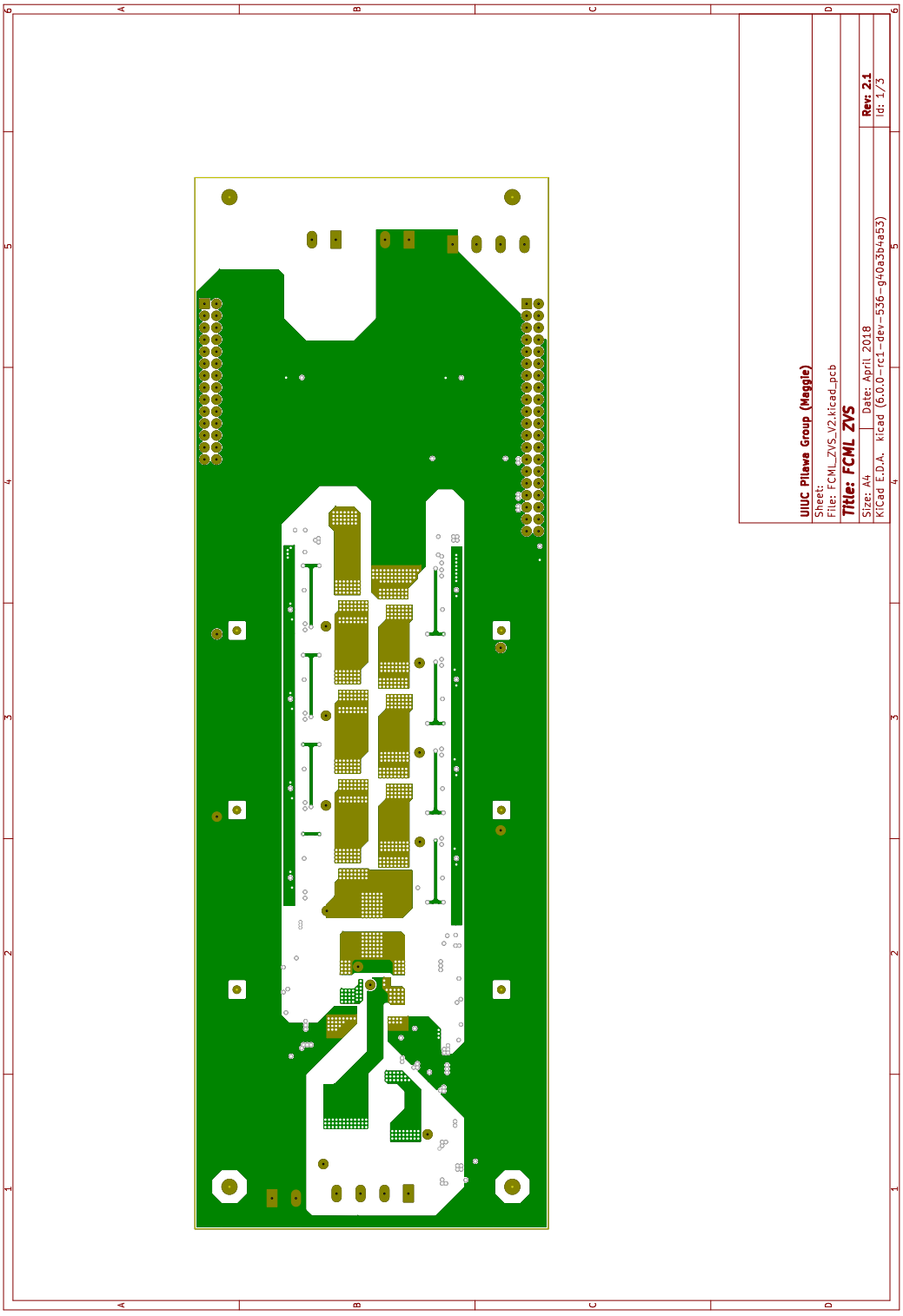


Figure B.15: Bottom layer of PCB.

Appendix C

Microcontroller Code for Dynamic Level Transitioning with Active Balancing

Included here are the program files for operating dynamic level selection for a 4/5-level FCML converter using a TI C2000 microcontroller for control.

Global Variables Header File

```
/*
 * global_variables.h
 */

#ifndef ZVS_FCML_28377D_GLOBAL_VARIABLES_H_
#define ZVS_FCML_28377D_GLOBAL_VARIABLES_H_

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

#define fundamental_frequency 60 // fundamental frequency in Hz

// Global variable declarations

extern int32 enable;
//extern __interrupt void cpu_timer0_isr(void);
extern __interrupt void ePWMInterrupt(void);
extern __interrupt void ADCInterrupt(void);

extern int32 sysclk;
```

```
20 extern float sysclk_inv;
21 extern float main_duty; // initial duty cycle
22 extern int32 period; // switching period
23 extern int32 period1; // adjusted switching period due to mcu timing issues
24
25 // Switching periods for active balancing
26 extern int32 periodAB451; // adjusted due to mcu timing issues
27 extern int32 periodAB45;
28
29
30 // Global variable definitions
31
32 extern int32 enable;
33 extern int32 state;
34 extern int32 N;
35 extern int fs; // switching frequency
36 extern int feff; // effective switching frequency at inductor
37 extern int freq;
38
39 // switching frequency lower limit (when to switch levels)
40 extern int32 f_lim;
41
42 extern int32 f_high; // set a switching frequency maximum
43 extern int f_set;
44 extern float L; // inductor value
45 extern float main_duty; // initial duty cycle
46 extern float duty;
47 extern float deff_r; // effective duty cycle
48 extern float deff;
49 extern int abcount; // number of active balancing cycles (gamma)
50
51 // (alpha) duty cycle adjustment factor from 5 to 4 levels
52 extern float abfactor54;
53 // (alpha) duty cycle adjustment factor from 4 to 5 levels
54 extern float abfactor45;
55
56
57 // Local variable definitions
58
59
60 extern int32 period;       // period of the ePWM counter
61 extern int32 periodp; // period for 1st AB due to mcu timing issues
```

```
62 extern int32 periodZVS; // period needed for ZVS
63 extern int32 deadtime_r;     // deadtime, constant
64 extern int32 deadtime_f;     // deadtime, constant
65 extern int32 phase;       // phase shift of each ePWM, in degrees
66 extern int32 sysclk;  // system clock, in kHz
67
68 // phase shift factor for each switch pair
69 extern float ps2_float;
70 extern float ps3_float;
71 extern float ps4_float;
72 extern float ps5_float;
73 extern float ps6_float;
74 extern float ps7_float;
75
76 // duty cycles for each subperiod
77 // factors
78 extern float d1;
79 extern float d2;
80 extern float d4;
81 extern float d8;
82 // total duty cycle in system clock ticks
83 extern float d1p;
84 extern float d2p;
85 extern float d4p;
86 extern float d8p;
87
88 // factor for length of sub-periods
89 extern float Tx; // inital subperiod
90 extern float T1;
91 extern float T2;
92 extern float T4;
93
94 // total duty cycle in system clock ticks
95 extern int32 D_ePWM2;
96 extern int32 D_ePWM3;
97 extern int32 D_ePWM4;
98 extern int32 D_ePWM5;
99
100 // phase shift for ePWM2, etc
101 extern int32 ps2;
102 extern int32 ps3;
103 extern int32 ps4;
```

```
104 extern int32 ps5;
105 extern int32 ps6;
106 extern int32 ps7;
107 extern float phaseshift; // initial phase shift
108
109
110 extern int32 index;
111 extern int32 currentRead_period;
112 extern int32 i;  //counter to control when level transition
113
114 extern float pre54;
115 extern float post54;
116 extern float pre45;
117 extern float post45;
118 extern int32 period4; // 4 level switching period
119 extern int32 period5; // 5 level switching period
120 extern int32 periodbase; // initial period
121 extern float pfactor; // factor of period adjustment between 4 and 5 levels
122 extern float pshift; // shift in period due to mcu timing issues
123
124 // current sense
125 extern Uint16 dummy_read;
126 extern Uint16 Iout_bias_count;
127 extern int16 Iout_count;
128 extern float Iout;
129 extern float Iout_sample_array[];
130 extern Uint16 Iout_pointer;
131 extern float Iout_sum;
132 extern float Iout_avg;
133 extern float Iout_adc_range_count;
134 extern float Iout_adc_range_count_div;
135 extern float Iout_ADC_Max_Amp;
136 extern float Iout_ADC_Min_Amp;
137 extern float Iout_adc_range_fullamp;
138 extern float Iout_adc_range_fullamp_div;
139 extern float Iout_adc_fullamp_to_count_ratio;
140 extern float Iout_adc_count_to_fullamp_ratio;
141 extern float mov_avg_size;
142 extern float mov_avg_size_div;
143
144 // duty cycle regions for 4 and 5 level
145 extern Uint16 N_minus1;
```

```
146 extern float L4region1;
147 extern float L4region2;
148 extern float L4region3;
149 extern float L5region1;
150 extern float L5region2;
151 extern float L5region3;
152 extern float L5region4;
153
154
155 extern int32 Vin; // input voltage
156 extern float Izvs; // negative peak of inductor current for ZVS
157 extern float f_ZVS; // switching frequency needed for ZVS
158 extern float T_ZVS; // switching period for ZVS
159 extern float T_const;
160
161 // debug variable
162 extern float bug;
163 extern float bug2;
164 extern float shiftx;
165 extern float shifty;
166 extern int16 lim;
167 #endif /* ZVS_FCML_28377D_GLOBAL_VARIABLES_H_ */
```

Global Variables Definition File

```
1
2  /*
3   * global_variables.c
4    */
5
6  #include "F28x_Project.h"     // Device Headerfile and Examples Include File
7  #include "global_variables.h"
8
9  // Global variable definitions
10
11 int32 enable = 0;
12 int32 N = 5;
13 int fs = 75;
14 int feff;
15 int freq = 0;
16 int32 f_lim = 80;
17 int32 f_high = 135;
18 int f_set = 96;
19 float main_duty = 0.25;
20 int32 Vin = 150;
21 float L = 0.0000056;
22 float Izvs = -0.9;
23 float duty;
24 float deff_r;
25 float deff;
26 int abcount = 7;
27 float abfactor54 = 1.0;
28 float abfactor45 = 1;
29 float pfactor = 0.375;
30
31
32 // Local variable definitions
33
34 //int32 num_levels;
35 int32 period = 4000;       // period of the ePWM counter
36 int32 period1;
37 int32 periodp = 4000; //period for 1st AB
38 int32 period4;
39 int32 periodbase;
40 int32 periodZVS = 4000;
```

```
41 int32 deadtime_r = 10;     // deadtime
42 int32 deadtime_f = 10;     // deadtime
43 int32 phase;       // phase shift of each ePWM, in degrees
44 int32 sysclk = 200000;  // system clock, in kHz
45 float sysclk_inv = 0.000005; // system clock in ms
46
47 // period of cpu timer, to trigger current sense read
48 int32 currentRead_period = 200000;
49
50 int32 periodAB451;
51 int32 periodAB45;
52
53 float ps2_float;
54 float ps3_float;
55 float ps4_float;
56 float ps5_float;
57 float ps6_float;
58 float ps7_float;
59
60 float d1;
61 float d2;
62 float d4;
63 float d8;
64 float d1p;
65 float d2p;
66 float d4p;
67 float d8p;
68
69 float Tx = 0.333333;
70 float T1;
71 float T2;
72 float T4;
73
74 int32 D_ePWM2;
75 int32 D_ePWM3;
76 int32 D_ePWM4;
77 int32 D_ePWM5;
78
79
80 int32 ps2;      // phase shift for ePWM2
81 int32 ps3;
82 int32 ps4;
```

```
int32 ps5;
int32 ps6;
int32 ps7;


int32 i = 1;  //counter to control when level transition
int32 index = 1;

int32 state = 5; // initial state (number of levels)

// Dummy variable for ADC measurement
Uint16 dummy_read = 0;

// ADC current measurements (in counts)
Uint16 Iout_bias_count = 0;
int16 Iout_count = 0;
float Iout_adc_range_count = 0;
float Iout_adc_range_count_div;
float Iout_ADC_Max_Amp = 3.5;
float Iout_ADC_Min_Amp = 0;
float Iout_adc_range_fullamp;
float Iout_adc_range_fullamp_div;
float Iout_adc_fullamp_to_count_ratio;
float Iout_adc_count_to_fullamp_ratio;


// ADC current measurements (in amps)
float Iout = 0;

// Moving average variables
float Iout_sample_array[200];
Uint16 Iout_pointer = 0;
float Iout_sum = 0;
float Iout_avg = 0;
float mov_avg_size = 200;
float mov_avg_size_div;


float pshift;
int32 period5;
float phaseshift;

```

```
125 // debug variables
126 float shiftx = 0;
127 float shifty = 0;
128 int16 lim = 0;
129 float bug = 0.0;
130 float bug2 = 0;
131
132 // ZVS frequency calculation variables
133 float deff;
134 Uint16 N_minus1;
135 float L4region1 = 1.0/3.0;
136 float L4region2 = 2.0/3.0;
137 float L4region3 = 1.0;
138 float L5region1 = 1.0/4.0;
139 float L5region2 = 1.0/2.0;
140 float L5region3 = 3.0/4.0;
141 float L5region4 = 1.0;
142 float f_ZVS;
143 float T_ZVS;
144 float T_const;
145
```

Main MCU Function

```
/*
 * main.c
 */
#include "F28x_Project.h"     // Device Headerfile and Examples Include File
#include "ZVS_FCML.h"
#include "initialize.h"
#include "global_variables.h"


#define RESULTS_BUFFER_SIZE 256
Uint16 AdcaResults[RESULTS_BUFFER_SIZE];
Uint16 resultsIndex;
Uint16 bufferFull;


void main(void)
{

enable = 0;
// Step 1. Initialize System Control:
InitSysCtrl();

// Step 2. Initialize GPIO:
    InitGpio();


// Step 3. Clear all interrupts and initialize PIE vector table:
// Disable CPU interrupts
    Init_interrupts();
    InitCpuTimers();
    ConfigCpuTimer(&CpuTimer0, 200, 10000);

    // Use write-only instruction to set TSS bit = 0
    CpuTimer0Regs.TCR.all = 0x4000;


//  Step 4. Initialize all the Device Peripherals:

// Initialize the ePWM

```

```
41      EALLOW;
42      CpuSysRegs.PCLKCR0.bit.TBCLKSYNC = 0; // disable PWM timer
43      //CpuSysRegs.PCLKCR0.bit.CPUTIMER0 = 0; // disable CPU timer
44      ClkCfgRegs.PERCLKDIVSEL.bit.EPWMCLKDIV = 0;
45      EDIS;
46
47      Init_phase_shifted_pwm(); // Initial PWM for phase shifted operation
48
49      //Init_cputimer(); // Initialize cputimer 1 for interrupt
50      Init_ADCb();
51
52      EALLOW;
53      CpuSysRegs.PCLKCR0.bit.TBCLKSYNC = 1; // enable synchronize ePWM
54      //CpuSysRegs.PCLKCR0.bit.CPUTIMER0 = 1; // start cpu timer
55      EDIS;
56
57          // Get ADC bias values for differential voltage
58          // and current sensor measurements
59      ADC_bias();
60
61          // Initialize global variables. Includes some ADC conversion
62          // calculations so must be called after
63       Init_global_variables();  ADC_bias().
64
65
66  // Step 5. User specific code, enable interrupts:
67
68  // Enable global Interrupts and higher priority real-time debug events:
69      EINT;  // Enable Global interrupt INTM
70      ERTM;  // Enable Global realtime interrupt DBGM
71
72
73  // Step 6. IDLE loop. Just sit and loop forever (optional):
74
75
76  // Interrupt
77  __interrupt void ADCInterrupt(void)
78  {
79
80  //GpioDataRegs.GPADAT.bit.GPIO14 = 1;
81  //ADC_calc();
82
```

```
83      // for each state (number of levels) calculate the effective duty cycle,
84      // and the switching frequency/period need for ZVS
85      // based on duty cycle region
86  if (state == 4){
87  //N_minus1 = state - 1;
88  N_minus1 = 3;
89
90  if (main_duty <= L4region1){
91  deff = main_duty*N_minus1;
92
93  // 1000 in denominator to make f_ZVS in kHz
94  f_ZVS = ((L4region1 - main_duty)*Vin*deff)/
95                              (1000*2*L*N_minus1*(Iout - Izvs));
96
97  // T_const = 1000*2*L, 1000 to make kHz
98  //T_ZVS = (T_const*N_minus1*(Iout - Izvs))/
99                              ((L4region1 - main_duty)*Vin*deff);
100 }
101 else if (main_duty > L4region1 && main_duty <= L4region2){
102 deff = (main_duty - L4region1)*N_minus1;
103
104 // 1000 in denominator to make f_ZVS in kHz
105 f_ZVS = ((L4region2 - main_duty)*Vin*deff)/
106                                  (1000*2*L*N_minus1*(Iout - Izvs));
107
108 // T_const = 1000*2*L, 1000 to make kHz
109 //T_ZVS = (T_const*N_minus1*(Iout - Izvs))/
110                             ((L4region2 - main_duty)*Vin*deff);
111 }
112 else if (main_duty > L4region2){
113 deff = (main_duty - L4region2)*N_minus1;
114
115 // 1000 in denominator to make f_ZVS in kHz
116 f_ZVS = ((L4region3 - main_duty)*Vin*deff)/
117                             (1000*2*L*N_minus1*(Iout - Izvs));
118 //T_ZVS = (T_const*N_minus1*(Iout - Izvs))/
119                             ((L4region3 - main_duty)*Vin*deff);
120 }
121
122 }
123 else {
124
```

```
125 N_minus1 = state - 1;
126 if (main_duty <= L5region1){
127 deff = main_duty*N_minus1;
128
129 // 1000 in denominator to make f_ZVS in kHz
130 f_ZVS = ((L5region1 - main_duty)*Vin*deff)/
131                             (1000*2*L*N_minus1*(Iout - Izvs));
132 //T_ZVS = (T_const*N_minus1*(Iout - Izvs))/
133                             ((L5region1 - main_duty)*Vin*deff);
134 }
135 else if (main_duty > L5region1 && main_duty <= L5region2){
136 deff = (main_duty - L5region1)*N_minus1;
137
138 // 1000 in denominator to make f_ZVS in kHz
139 f_ZVS = ((L5region2 - main_duty)*Vin*deff)/
140                             (1000*2*L*N_minus1*(Iout - Izvs));
141 //T_ZVS = (T_const*N_minus1*(Iout - Izvs))/
142                                 ((L5region2 - main_duty)*Vin*deff);
143 }
144 else if (main_duty > L5region2 && main_duty <= L5region3){
145 deff = (main_duty - L5region2)*N_minus1;
146
147 // 1000 in denominator to make f_ZVS in kHz
148 f_ZVS = ((L5region3 - main_duty)*Vin*deff)/
149                             (1000*2*L*N_minus1*(Iout - Izvs));
150 //T_ZVS = (T_const*N_minus1*(Iout - Izvs))/
151                             ((L5region3 - main_duty)*Vin*deff);
152 }
153 else if (main_duty > L5region3){
154 deff = (main_duty - L5region3)*N_minus1;
155
156 // 1000 in denominator to make f_ZVS in kHz
157 f_ZVS = ((L5region4 - main_duty)*Vin*deff)/
158                             (1000*2*L*N_minus1*(Iout - Izvs));
159 //T_ZVS = (T_const*N_minus1*(Iout - Izvs))/
160                             ((L5region4 - main_duty)*Vin*deff);
161 }
162
163 }
164
165     // select switching frequency based on calculations checked
166     // against limits
```

```
167 if (f_ZVS < f_lim){
168 // f_ZVS = flim;
169 periodZVS = sysclk/f_lim;
170
171 }
172 else if (f_ZVS > f_high){
173 periodZVS = sysclk/f_high;
174 }
175 else{
176 periodZVS = sysclk/f_ZVS;
177 //periodZVS = sysclk*T_ZVS;
178 }
179
180 AdcbRegs.ADCINTFLGCLR.bit.ADCINT2 = 1; //clear INT2 flag on ADCB
181 PieCtrlRegs.PIEACK.all = PIEACK_GROUP10; // Acknowledge read of PIE Group 10
182 //GpioDataRegs.GPADAT.bit.GPIO14 = 0;
183 }
184
185 // current sense, ADC calaculations
186 void ADC_calc(void)
187 {
188
189 // Read measured voltages from ADC results registers and
190 // subtract off zero bias.
191 // Get ADC result and subtract off initial bias
192 Iout_count = AdcbResultRegs.ADCRESULT0 - Iout_bias_count;
193
194 // Compute moving averages of measured voltages.
195 // Done in units of "counts" (int16)
196 // Compute a moving average (LPF) of measured Iout (in counts)
197
198 // Iout_sum = Iout_sum + newest value - oldest value
199 Iout_sum = Iout_sum + Iout_count - Iout_sample_array[Iout_pointer];
200
201 // replace the oldest value with the newest value
202 Iout_sample_array[Iout_pointer] = Iout_count;
203
204 // Divide the moving sum by the size of the moving
205 // average filter to compute the average value
206 Iout_avg = Iout_sum*mov_avg_size_div;
207 Iout_pointer++; //increment pointer by 1
208
```

```
209 // Reset the pointer to zero if it exceeds Iout array size
210 if (Iout_pointer == mov_avg_size) Iout_pointer = 0;
211
212 // Scale Iout from counts to full amps.
213 // Go from ADC counts to amps (full). Conversion derived analytically.
214 Iout = Iout_count*Iout_adc_count_to_fullamp_ratio;
215
216 }
217
218
```

Initialization Header File

```
/*
 * initialize.h
 */

#ifndef ZVS_FCML_28377D_INITIALIZE_H_
#define ZVS_FCML_28377D_INITIALIZE_H_

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

void Clear_interrupts(void);
void Init_phase_shifted_pwm(void);
void InitEPwm_1(void);
void InitEPwm_2(void);
void InitEPwm_3(void);
void InitEPwm_4(void);
void InitEPwm_5(void);
void InitEPwm_6(void);
void InitEPwm_7(void);

void Init_cputimer(void);
void Init_global_variables(void);


// Initialize the necessary interrupts (without enabling)
void Init_interrupts(void);


void Init_ADCb(void);
void ADC_bias(void);
void ADC_conversion_wait(void);

void ADC_calc(void);

#endif /* BUFFER_V5_INITIALIZE_H_ */


```

Initialization Function File

```
/*
 * initialize.c
 */

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

// Initialize all global variables to their nonzero values.
void Init_global_variables()
{

// Declare and define local variables for adc conversion from
// full voltage to counts (and vice versa)

// Full adc range in counts (w/ bias)
float Iout_adc_range_count = (4096 - Iout_bias_count);

// Inverse of full adc range in counts (w/ bias)
float Iout_adc_range_count_div = 1/Iout_adc_range_count;

// Full adc range in volts (full voltage)
float Iout_adc_range_fullamp = (Iout_ADC_Max_Amp - Iout_ADC_Min_Amp);

// Inverse of full adc range in volts (full voltage)
float Iout_adc_range_fullamp_div = 1/Iout_adc_range_fullamp;

// Define global adc conversion ratios for adc conversion from
// full voltage to counts (and vice versa)

// Full volt to count adc conversion. Count = Volt*Ratio.
Iout_adc_fullamp_to_count_ratio =
                        Iout_adc_range_count*Iout_adc_range_fullamp_div;

// Full volt to count adc conversion. Volt = Count*Ratio.
Iout_adc_count_to_fullamp_ratio =
                        Iout_adc_range_count_div*Iout_adc_range_fullamp;


mov_avg_size_div = 1/mov_avg_size; // Inverse of mov_avg_size.
```

```
41
42 }
43
44 void Init_phase_shifted_pwm()
45 {
46 feff = (N-1)*fs;
47
48 // enable PWM1, PWM2, PWM3, PWM4, PWM5, PWM6 PWM7
49 CpuSysRegs.PCLKCR2.bit.EPWM1=1;
50 CpuSysRegs.PCLKCR2.bit.EPWM2=1;
51 CpuSysRegs.PCLKCR2.bit.EPWM3=1;
52 CpuSysRegs.PCLKCR2.bit.EPWM4=1;
53 CpuSysRegs.PCLKCR2.bit.EPWM5=1;
54 CpuSysRegs.PCLKCR2.bit.EPWM6=1;
55 //  CpuSysRegs.PCLKCR2.bit.EPWM7=1;
56
57 // Initialize GPIO pins for ePWM1, ePWM2, ePWM3, ePWM4, ePWM5
58 // These functions are in the F28M36x_EPwm.c file
59 InitEPwm1Gpio();
60 InitEPwm2Gpio();
61 InitEPwm3Gpio();
62 InitEPwm4Gpio();
63 InitEPwm5Gpio();
64 //InitEPwm6Gpio();
65
66 // output pin for debug
67 GpioCtrlRegs.GPADIR.bit.GPIO14 = 1;
68 GPIO_SetupPinOptions(14, GPIO_OUTPUT, GPIO_PUSHPULL);
69 GpioDataRegs.GPADAT.bit.GPIO14 = 0; // set low for 5 L case
70
71     // output pin for debug
72 GpioCtrlRegs.GPADIR.bit.GPIO10 = 1;
73 GPIO_SetupPinOptions(10, GPIO_OUTPUT, GPIO_PUSHPULL);
74 GpioDataRegs.GPADAT.bit.GPIO10 = 0; //
75
76 period = sysclk/fs;    // ePWM timer period
77
78 T_const = 1000*2*L;
79 D_ePWM2 = main_duty;
80 D_ePWM3 = main_duty;
81 D_ePWM4 = main_duty;
82 D_ePWM5 = main_duty;
```

```
83
84 // Phase shift for each ePWM
85 phase = 360/(N-1);
86 // Effective periods
87 //    Tx = 1/3;
88 //    T1 = 1/3;
89 //    T2 = 1/3;
90 //    T4 = 1/3;
91 // 5 level
92 ps2_float = (phase*3.0/360.0);
93 ps3_float = (phase*2.0/360.0);
94 ps4_float = (phase*1.0/360.0);
95 ps5_float = 0;
96
97 /*// 4 level
98 ps2_float = 0;
99 ps3_float = (phase*1.0/360.0);
100 ps4_float = (phase*1.0/360.0);
101 ps5_float = (phase*2.0/360.0);
102 */
103
104 ps2=period*ps2_float;
105 ps3=period*ps3_float;
106 ps4=period*ps4_float;
107 ps5=period*ps5_float;
108
109 // Initialize each ePWM
110 InitEPwm_1();
111 InitEPwm_2();
112 InitEPwm_3();
113 InitEPwm_4();
114 InitEPwm_5();
115 InitEPwm_6();
116
117 }
118
119 void InitEPwm_1()
120 {
121
122 EPwm1Regs.TBPRD = period;                       // Set timer period
123 EPwm1Regs.TBCTR = 0x0000;                       // Clear counter
124
```

```
125 // Setup TBCLK
126 EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;    // Count up
127
128 // Disable phase loading for the first ePWM, this becomes the master ePWM
129 EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;
130
131 EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;      // Clock ratio to SYSCLKOUT
132
133 // Same frequency as main clock
134 EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;
135
136 // send sync output signal when counter is zero
137 EPwm1Regs.TBCTL.bit.SYNCOSEL = TB_CTR_ZERO;
138 EPwm1Regs.TBCTL.bit.PRDLD = TB_SHADOW;   // load period from shadow register
139 //EPwm1Regs.TBCTL.bit.PRDLD = TB_IMMEDIATE;
140
141 // Setup compare
142 EPwm1Regs.CMPA.bit.CMPA = period*.05;             // initial 50% duty ratio
143
144  // load compare value from shadow registor at CTR=ZERO
145 EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
146
147 // configure pwm as a slave (for syncing) (Note: the default is slave)
148 EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
149 //EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_IMMEDIATE;
150
151 // Set actions
152 EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR;             // Set PWM3A on Zero
153 EPwm1Regs.AQCTLA.bit.ZRO = AQ_SET;
154
155 // Active high complementary PWMs and Setup the deadband
156 EPwm1Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
157 EPwm1Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
158 EPwm1Regs.DBCTL.bit.IN_MODE = DBA_ALL;
159 EPwm1Regs.DBRED = 4;
160 EPwm1Regs.DBFED = 4;
161
162 //setup for ADC conversions
163 EPwm1Regs.ETSEL.bit.SOCAEN = 1;         // Disable SOC on A group
164 EPwm1Regs.ETSEL.bit.SOCASEL = ET_CTR_ZERO;         // Select SOC on up-count
165 EPwm1Regs.ETPS.bit.SOCAPRD = 1;         // Generate pulse on 1st event
166
```

```
167 EPwm1Regs.ETSEL.bit.INTEN = 1; // Enable ePWM interrupt
168
169 //enable event time-base counter equal to zero
170 EPwm1Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO;
171
172 EPwm1Regs.ETPS.bit.INTPSSEL = 0;
173 EPwm1Regs.ETPS.bit.INTCNT = 0;
174 EPwm1Regs.ETPS.bit.INTPRD = ET_1ST;
175 EPwm1Regs.ETCLR.bit.INT = 1; //clear interrupt flag intially
176 }
177
178 void InitEPwm_2()
179 {
180
181 EPwm2Regs.TBPRD = period;                          // Set timer period
182 EPwm2Regs.TBPHS.bit.TBPHS = ps2;           // Phase is 0
183 EPwm2Regs.TBCTR = 0x0000;                      // Clear counter
184
185 // Setup TBCLK
186 EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Count up
187 EPwm2Regs.TBCTL.bit.PHSEN = TB_ENABLE; // Enable phase loading
188 EPwm2Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1; // Clock ratio to SYSCLKOUT
189 EPwm2Regs.TBCTL.bit.CLKDIV = TB_DIV1; // Same frequency as main clock
190 EPwm2Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN; // pass sync in to sync out
191 EPwm2Regs.TBCTL.bit.PRDLD = TB_SHADOW; // load period from shadow register
192
193 // load period from shadow register at SYNC event
194 EPwm2Regs.TBCTL2.bit.PRDLDSYNC = TB_PRD_SYNC;
195
196 // Setup compare
197 EPwm2Regs.CMPA.bit.CMPA = period*main_duty; // initial 50% duty ratio
198
199 // load from shadow register at CTR=ZERO
200 //EPwm2Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
201
202 // configure pwm as a slave (for syncing) (Note: the default is slave)
203 EPwm2Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
204
205 // load from shadow register at SYNC event
206 EPwm2Regs.CMPCTL.bit.LOADASYNC = CC_SYNC;
207
208 // Set actions
```

```
209 EPwm2Regs.AQCTLA.bit.CAU = AQ_CLEAR; // Set PWM3A on Zero
210 EPwm2Regs.AQCTLA.bit.ZRO = AQ_SET;
211
212
213 // Active high complementary PWMs - Setup the deadband
214 EPwm2Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
215 EPwm2Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
216 EPwm2Regs.DBCTL.bit.IN_MODE = DBA_ALL;
217 EPwm2Regs.DBRED = deadtime_r;
218 EPwm2Regs.DBFED = deadtime_f;
219
220 }
221
222 void InitEPwm_3()
223 {
224     ... // same as InitEPwm_2()
225     // but with EPwm3Regs.TBPHS.bit.TBPHS = ps3;
226 }
227
228 void InitEPwm_4()
229 {
230     ... // same as InitEPwm_2()
231 // but with EPwm4Regs.TBPHS.bit.TBPHS = ps4;
232 }
233 void InitEPwm_5()
234 {
235 ... // same as InitEPwm_2()
236 // but withEPwm5Regs.TBPHS.bit.TBPHS = ps5;
237
238 }
239 void InitEPwm_6()
240 {
241     ... // same as InitEPwm_2()
242 // but withEPwm6Regs.TBPHS.bit.TBPHS = ps5;
243 }
244
245 void Init_interrupts()
246 {
247 // Step 1: Disable interrupts globally
248
249 // Disable CPU interrupts
250 DINT;
```

```
251
252 // Initialize the PIE control registers to their default state.
253 // The default state is all PIE interrupts disabled and flags
254 // are cleared.
255 InitPieCtrl();
256
257 // Disable CPU interrupts and clear all CPU interrupt flags:
258 EALLOW;
259 IER = 0x0000;
260 IFR = 0x0000;
261 EDIS;
262
263 // Step 2: Enable the PIE by setting the ENPIE bit of the PIECTRL register.
264
265 InitPieVectTable();
266
267 // Enable the PIE
268 PieCtrlRegs.PIECTRL.bit.ENPIE = 1;
269
270 // Step 3: Write the ISR vector for each interrupt to the appropriate
271 // location in the PIE vector table, which can be found in Table 2-2.
272
273 EALLOW; // This is needed to write to EALLOW protected registers
274
275     // ISR function address for ADCB interrupt #1
276     // PieVectTable.ADCB1_INT = &ADC_interrupt1;
277
278     //ISR function address for ADCC interrupt #1
279     // PieVectTable.ADCC1_INT = &ADC_interrupt2;
280
281 // Step 4: Set the appropriate PIEIERx bit for each interrupt.
282 // The PIE group and channel assignments can be found in Table 2-2.
283 // Map ISR functions
284
285   //PieVectTable.TIMER0_INT = &cpu_timer0_isr;
286
287     // ISR function address for ADCB interrupt #2
288 PieVectTable.ADCB2_INT = &ADCInterrupt;
289
290 // ISR function address for ePWM2 interrupt
291 PieVectTable.EPWM1_INT = &ePWMInterrupt;
292
```

```
293 EDIS; // This is needed to disable write to EALLOW protected registers
294
295 // Step 5: Set the CPU IER bit for any
296 // PIE group containing enabled interrupts.
297 // Enable PIE interrupt (see Table 2.2 of Technical Reference Manual)
298
299 // Enable TINT0 in the PIE: Group 1 interrupt 7
300 PieCtrlRegs.PIEIER1.bit.INTx7 = 1;
301
302 // Enable EPWM INTn in the PIE: Group 3 interrupt 1
303 PieCtrlRegs.PIEIER3.bit.INTx1 = 1;
304
305 // Enable ADCB2 INTn in the PIE: Group 10 interrupt 10.6
306 PieCtrlRegs.PIEIER10.bit.INTx6 = 1;
307
308 // Set the CPU IER bit for any PIE
309 // group containing enabled interrupts.,
310 IER |= M_INT1; //Enable group 1 interrupts
311 IER |= M_INT3; //Enable group 3 interrupts
312 IER |= M_INT10; //Enable group 10 interrupts
313 EDIS;
314
315 // Step 6: Enable the interrupt in the peripheral.
316
317 // This step is completed in main.c
318
319 }
320
321 void Init_ADCb(void)
322 {
323 EALLOW;
324 //write configurations
325 AdcbRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4
326 AdcbRegs.ADCCTL2.bit.RESOLUTION = ADC_RESOLUTION_12BIT;
327 AdcbRegs.ADCCTL2.bit.SIGNALMODE = ADC_SIGNALMODE_SINGLE;
328     //AdcSetMode(ADC_ADCB, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE);
329 //Set pulse positions to late (at the end of conversion)
330 AdcbRegs.ADCCTL1.bit.INTPULSEPOS = 1;
331 //power up the ADC
332 AdcbRegs.ADCCTL1.bit.ADCPWDNZ = 1;
333
334 //SOC0 measure Iout on pin B2
```

```
335
336 //SOC0 will convert channel 2 of ADCB (pin B2)
337 AdcbRegs.ADCSOC0CTL.bit.CHSEL = 2;
338
339 //sample window (# of SYSCLK, needs to corresponds to at least 75ns)
340 AdcbRegs.ADCSOC0CTL.bit.ACQPS = 50;
341
342 //trigger on CPU1 timer 0, see page 1467
343 AdcbRegs.ADCSOC0CTL.bit.TRIGSEL = 1;
344
345 // Enable interrupt for SOC0 of ADCB (in this case, B3 = SOC0)
346
347 //end of SOC0 (i.e. EOC0) will set INT2 flag
348 AdcbRegs.ADCINTSEL1N2.bit.INT2SEL = 0;
349 AdcbRegs.ADCINTSEL1N2.bit.INT2E = 1;   //enable INT2 flag
350
351 //No further ADCINT2 pulses are generated until
352 // ADCINT2 flag is cleared by user
353 AdcbRegs.ADCINTSEL1N2.bit.INT2CONT = 0;
354
355 AdcbRegs.ADCINTFLGCLR.bit.ADCINT2 = 1; //make sure INT2 flag is cleared
356
357 EDIS;
358
359 }
360
361
362 // This function calculates the bias on all ADC inputs
363 // (especially desirable for differential voltage and current sensors)
364 // and stores as a global variable for later use
365 void ADC_bias(void)
366 {
367
368 // The first ADC reading might not be accurate,
369 // so do a dummy read and throw away this value
370
371 ADC_conversion_wait();
372
373 // Wait for the ADC conversion to finish
374 dummy_read = AdcbResultRegs.ADCRESULT0; // (ADCB SOC0)
375
376 AdcbRegs.ADCINTFLGCLR.bit.ADCINT2 = 1; //clear INT2 flag on ADCB
```

```
377
378 // wait........
379 // make sure wait for 1s at least for all the external circuit
380 // to power on !!!!!
381 // 1s is the measured delay from power on to current sensing
382 // amp has valid signal
383 // otherwise the bias measurement might have unexpected error
384 DELAY_US(700000);
385
386 Uint32 Iout_bias_count_sum = 0;
387
388 // measure bias voltage of current sensing amplifier
389 Uint16 adc_read_count = 0;
390
391 // Number of bits to average for ADC measurement (9 bits = 512 counts)
392 Uint16 adc_read_count_num_bits = 9;
393
394 for (adc_read_count=0;
395     adc_read_count<(1<<adc_read_count_num_bits);
396     adc_read_count++)
397 {
398
399 ADC_conversion_wait(); // Wait for the ADC conversion to finish
400
401 // read result from ADCB SOC0
402 Iout_bias_count_sum += AdcbResultRegs.ADCRESULT0;
403
404 AdcbRegs.ADCINTFLGCLR.bit.ADCINT2 = 1; //clear INT2 flag on ADCB
405 }
406 Iout_bias_count = Iout_bias_count_sum>>adc_read_count_num_bits;
407
408 }
409
410 // This function waits until all enabled ADC conversions are finished.
411 // Note: call this function only after ADC triggering is enabled but
412 // before interrupts are enabled
413 void ADC_conversion_wait(void)
414 {
415 // Make sure all ADC conversions are finished (check interrupt flag)
416 while (AdcbRegs.ADCINTFLG.bit.ADCINT2 != 1);
417
418 }
```

419

FCML ZVS Header File

```
1  /*
2   * ZVS_FCML.h
3   */
4
5  #ifndef FCMC_H_
6  #define FCMC_H_
7
8  #ifdef __cplusplus
9  extern "C" {
10 #endif
11
12
13
14
15 /* Function prototypes */
16
17 void Init_cputimer_sin_TMU(void);
18 void Init_phase_shifted_pwm(void);
19
20 void InitEPwm_1(void);
21 void InitEPwm_2(void);
22 void InitEPwm_3(void);
23 void InitEPwm_4(void);
24 void InitEPwm_5(void);
25 void InitEPwm_6(void);
26 void InitEPwm_7(void);
27
28 //state functions
29 void Level5(void);
30 void PreAB54(void);
31 void AB54(void);
32 void PostAB54(void);
33 void Level4(void);
34 void PreAB45(void);
35 void AB45(void);
36 void PostAB45(void);
37
38
39 #endif /* FCMC_H_ */
```

FCML ZVS Function

```
/*
 * ZVS_FCML.c
 */



#include "F28x_Project.h" // Device Headerfile and Examples Include File
#include "ZVS_FCML.h"
#include "initialize.h"
#include "global_variables.h"



//__interrupt void cpu_timer0_isr(void)
__interrupt void ePWMInterrupt(void)
{

EPwm5Regs.CMPA.bit.CMPA = D_ePWM5; // set duty cycle ePWM 5
                                   // set here bc of mcu timing issue

if(i < 200000){
i++;
}
//
else if (i == 200000){  //after some time change levels
if(state == 5){ // if 5 levels
GpioDataRegs.GPADAT.bit.GPIO14 = 1;

state = 4;  // change to preAB state
i = 1; // reset count
index = 1; // reset AB count
}
else if(state == 4){ // if 4 levels
GpioDataRegs.GPADAT.bit.GPIO14 = 0;

state = 4; // AB to 5 levels
i = 1; // reset count
index = 1; // reset AB count
}
else
```

```
41 state = 5;
42 }
43 else
44 i = 1;
45
46     // this is base period, updates when change fs in debug terminal
47 periodbase = (sysclk/fs);
48
49 if(state == 4){
50 N = 4;
51 phase = 360/(N-1);
52 period4 = pfactor*periodbase;
53 // period = period4;
54
55 // period = periodZVS;
56 period = sysclk/f_set;
57 periodp = period;
58 freq = sysclk/period;
59 Level4();
60
61 if(index == 1){ // Adjusted Active Balancing on 4-levels
62 index++;
63 pshift = ps2 - ps3;
64
65 PreAB54();
66 pshift =  pshift - ps2 + ps3;
67
68 //pshift = 0;
69
70 period = period4 + pshift + shiftx;
71
72
73 }
74
75 else if(index > 1 && index <= abcount){ // Active Balancing on 4-levels
76 index++;
77 AB54();
78
79
80 }
81 // Adjusted Active Balancing on 4-levels before 4level operation
82 else if(index == abcount+1){
```

```
index++;
pshift = ps2 - ps3;

PostAB54();
pshift = ps2 - ps3 - pshift;
//pshift = 0;

period = period4 - pshift;
}
else if (index > abcount+1){
index++;
Level4();
pshift = ps2;

}

}

else if(state == 5){
N = 5;
phase = 360/(N-1);
period5 = periodbase;
period4 = pfactor*period5;
//period = period5;


//period = periodZVS;
period = sysclk/f_set;
freq = sysclk/period;
Level5();

     if(index == 1){ // Adjusted Active Balancing on 4-levels
index++;

pshift = ps2 - ps3;

PreAB45();
pshift = ps2 - ps3 - pshift;
period = period4 - pshift;

}

```

```
125 else if(index > 1 && index <= abcount){ // Active Balancing on 4-levels
126 index++;
127 AB45();
128 period1 = period;
129
130 }
131 else if(index == abcount+1){ // Active Balancing on 4-levels
132 index++;
133
134 pshift = ps2 - ps3;
135 period =  period4 + (periodbase*(0.75) - ps2);
136
137 PostAB45();
138 pshift = pshift - (ps2 - ps3);
139
140 }
141
142 else if (index > abcount+1){
143 index++;
144 period = periodbase;
145 Level5();
146
147
148 }
149 }
150
151     // update ePWM registers
152 EPwm5Regs.TBPRD = period;
153 EPwm4Regs.TBPRD = period;
154 EPwm3Regs.TBPRD = period;
155 EPwm2Regs.TBPRD = period;
156 EPwm1Regs.TBPRD = period;
157
158 GpioDataRegs.GPADAT.bit.GPIO10 = 1;
159     EPwm5Regs.TBPHS.bit.TBPHS = ps5;
160     EPwm4Regs.TBPHS.bit.TBPHS = ps4;
161     EPwm3Regs.TBPHS.bit.TBPHS = ps3;
162 EPwm2Regs.TBPHS.bit.TBPHS = ps2;
163 GpioDataRegs.GPADAT.bit.GPIO10 = 0;
164
165 EPwm4Regs.CMPA.bit.CMPA = D_ePWM4;
166 EPwm3Regs.CMPA.bit.CMPA = D_ePWM3;
```

```
167 EPwm2Regs.CMPA.bit.CMPA = D_ePWM2;
168
169
170 EPwm5Regs.DBRED = deadtime_r;
171 EPwm5Regs.DBFED = deadtime_f;
172 EPwm4Regs.DBRED = deadtime_r;
173 EPwm4Regs.DBFED = deadtime_f;
174 EPwm3Regs.DBRED = deadtime_r;
175 EPwm3Regs.DBFED = deadtime_f;
176 EPwm2Regs.DBRED = deadtime_r;
177 EPwm2Regs.DBFED = deadtime_f;
178
179
180
181
182
183 // Clear interrupt flag
184     EPwm1Regs.ETCLR.bit.INT = 1;
185     PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
186
187 }
188
```

State Logic Functions

```
/*
 * state_logic.c
 */

#include "F28x_Project.h"  // Device Headerfile and Examples Include File
#include "initialize.h"
#include "global_variables.h"
#include "ZVS_FCML.h"

void Level5(){

    // set ePWM registers
ps2_float = (phase*0.00833333); // 0.0083333=3.0/360.0
ps3_float = (phase*0.00555555); // 0.0055=2.0/360.0
  ps4_float = (phase*0.00277777); // 0.0027=1.0/360.0
ps5_float = 0;

ps2=period*ps2_float;
ps3=period*ps3_float;
ps4=period*ps4_float;
ps5=period*ps5_float;

D_ePWM2 = (int32) period*main_duty;
D_ePWM3 = (int32) period*main_duty;
D_ePWM4 = (int32) period*main_duty;
D_ePWM5 = (int32) period*main_duty;
}

void PreAB54(){

periodp = pfactor*periodbase;

    // T2 (period corresponding to pulse of EPWM3/4) adjusted from nominal
    // value by a factor, abfactor, which is calculated above based on
    // load current
T2 = abfactor54*Tx;

// T1 (period corresponding to pulse of EPWM5) adjusted from nominal value
T1 = (1-T2)/2;

```

```
// T4 (period corresponding to pulse of EPWM2) adjusted from nominal value
T4 = T1;

    // d2 (duty corresponding to pulse of EPWM3/4) adjusted from nominal
    // value based on new adjusted period
d2 = T2*deff;

//d1 (duty corresponding to pulse of EPWM5) adjusted from nominal value
// based on new adjusted period
d1 = T1*deff;

//d4 (duty corresponding to pulse of EPWM2) adjusted from nominal value
// based on new adjusted period
d4 = d1;

    // set ePWM registers
ps5_float = 0;
ps4_float = T2;
ps3_float = ps4_float;
ps2_float = T2+T4;

ps2=periodp*ps2_float;
ps3=periodp*ps3_float;
ps4=periodp*ps4_float;
ps5=periodp*ps5_float;


D_ePWM2 = periodp*d4;
D_ePWM3 = periodp*d2;
D_ePWM4 = periodp*d2;
D_ePWM5 = periodp*d1;
}
void AB54(){
period = period4;

    // T2 (period corresponding to pulse of EPWM3/4) adjusted from nominal
    // value by a factor, abfactor, which is calculated above based on
    // load current
T2 = abfactor54*Tx;

// T1 (period corresponding to pulse of EPWM5) adjusted from nominal value
T1 = (1-T2)/2;
```

```
83
84 // T4 (period corresponding to pulse of EPWM2) adjusted from nominal value
85 T4 = T1;
86
87     // d2 (duty corresponding to pulse of EPWM3/4) adjusted from nominal
88     // value based on new adjusted period
89 d2 = T2*deff;
90
91 // d1 (duty corresponding to pulse of EPWM5) adjusted from nominal value
92 // based on new adjusted period
93 d1 = T1*deff;
94
95 // d4 (duty corresponding to pulse of EPWM2) adjusted from nominal value
96 // based on new adjusted period
97 d4 = d1;
98
99     // set ePWM registers
100 ps5_float = 0;
101 ps4_float = T2;
102 ps3_float = ps4_float;
103 ps2_float = T2+T4;
104
105 ps2=period*ps2_float;
106 ps3=period*ps3_float;
107 ps4=period*ps4_float;
108 ps5=period*ps5_float;
109
110 D_ePWM2 = period*d4;
111 D_ePWM3 = period*d2;
112 D_ePWM4 = period*d2;
113 D_ePWM5 = period*d1;
114 }
115 void PostAB54(){
116
117 periodp = period4;
118
119     // set ePWM registers
120 ps2_float = (phase*0.00555555); // 0.0055=2.0/360.0
121 ps3_float = (phase*0.00277777); // 0.0027=1.0/360.0
122   ps4_float = (phase*0.00277777); // 0.0027=1.0/360.0
123 ps5_float = 0;
124
```

```
125 ps2=periodp*ps2_float;
126 ps3=periodp*ps3_float;
127 ps4=periodp*ps4_float;
128 ps5=periodp*ps5_float;
129
130 D_ePWM2 = periodp*main_duty;
131 D_ePWM3 = periodp*main_duty;
132 D_ePWM4 = periodp*main_duty;
133 D_ePWM5 = periodp*main_duty;
134 }
135
136 void Level4(){
137
138     // set ePWM registers
139 ps2_float = (phase*0.00555555); // 0.0055=2.0/360.0
140 ps3_float = (phase*0.00277777); // 0.0027=1.0/360.0
141   ps4_float = (phase*0.00277777); // 0.0027=1.0/360.0
142 ps5_float = 0;
143
144 ps2=periodp*ps2_float;
145 ps3=periodp*ps3_float;
146 ps4=periodp*ps4_float;
147 ps5=periodp*ps5_float;
148
149 D_ePWM2 = (int32) period*main_duty;
150 D_ePWM3 = (int32) period*main_duty;
151 D_ePWM4 = (int32) period*main_duty;
152 D_ePWM5 = (int32) period*main_duty;
153 }
154 void PreAB45(){
155
156 periodp = period4;
157
158     // T2 (period corresponding to pulse of EPWM3/4) adjusted from nominal
159     // value by a factor, abfactor, which is calculated above based on
160     // load current
161 T2 = abfactor45*Tx;
162
163 // T1 (period corresponding to pulse of EPWM5) adjusted from nominal value
164 T1 = (1-T2)/2;
165
166 // T4 (period corresponding to pulse of EPWM2) adjusted
```

```
// from nominal value
T4 = T1;

    // d2 (duty corresponding to pulse of EPWM3/4) adjusted from nominal value
    // based on new adjusted period
d2 = T2*deff;

// d1 (duty corresponding to pulse of EPWM5) adjusted from nominal value
// based on new adjusted period
d1 = T1*deff;

// d4 (duty corresponding to pulse of EPWM2) adjusted from nominal value
// based on new adjusted period
d4 = d1;

    // set ePWM registers
ps5_float = 0;
ps4_float = T2;
ps3_float = ps4_float;
ps2_float = T2+T4;

ps2=periodp*ps2_float;
ps3=periodp*ps3_float;
ps4=periodp*ps4_float;
ps5=periodp*ps5_float;

D_ePWM2 = periodp*d4;
D_ePWM3 = periodp*d2;
D_ePWM4 = periodp*d2;
D_ePWM5 = periodp*d1;
}
void AB45(){

period = period4;

    // T2 (period corresponding to pulse of EPWM3/4) adjusted from nominal
    // value by a factor, abfactor, which is calculated above based
    // on load current
T2 = abfactor45*Tx;

// T1 (period corresponding to pulse of EPWM5) adjusted from nominal value
T1 = (1-T2)/2;
```

```
209
210 //T4 (period corresponding to pulse of EPWM2) adjusted from nominal value
211 T4 = T1;
212
213     // d2 (duty corresponding to pulse of EPWM3/4) adjusted from nominal
214     // value based on new adjusted period
215 d2 = T2*deff;
216
217 // d1 (duty corresponding to pulse of EPWM5) adjusted from nominal value
218 // based on new adjusted period
219 d1 = T1*deff;
220
221 //d4 (duty corresponding to pulse of EPWM2) adjusted from nominal value
222 // based on new adjusted period
223 d4 = d1;
224
225     // set ePWM registers
226 ps5_float = 0;
227 ps4_float = T2;
228 ps3_float = ps4_float;
229 ps2_float = T2+T4;
230
231 ps2=period*ps2_float;
232 ps3=period*ps3_float;
233 ps4=period*ps4_float;
234 ps5=period*ps5_float;
235
236 D_ePWM2 = period*d4;
237 D_ePWM3 = period*d2;
238 D_ePWM4 = period*d2;
239 D_ePWM5 = period*d1;
240 }
241 void PostAB45(){
242
243 periodp = periodbase;
244
245     // set ePWM registers
246 ps2_float = (phase*0.00833333); // 0.0083333=3.0/360.0
247 ps3_float = (phase*0.00555555); // 0.0055=2.0/360.0
248   ps4_float = (phase*0.00277777); // 0.0027=1.0/360.0
249 ps5_float = 0;
250
```

```
251 ps2=periodp*ps2_float;
252 ps3=periodp*ps3_float;
253 ps4=periodp*ps4_float;
254 ps5=periodp*ps5_float;
255
256 D_ePWM2 = periodp*main_duty;
257 D_ePWM3 = periodp*main_duty;
258 D_ePWM4 = periodp*main_duty;
259 D_ePWM5 = periodp*main_duty;
260 }
261
262
```