IMPROVING DATA CENTER POWER DELIVERY EFFICIENCY AND POWER DENSITY WITH DIFFERENTIAL POWER PROCESSING AND MULTILEVEL POWER CONVERTERS

BY

ENVER CANDAN

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2019

Urbana, Illinois

Doctoral Committee:

Associate Professor Robert C. N. Pilawa-Podgurski, Chair
Professor Philip T. Krein
Professor Alejandro Domínguez-García
Assistant Professor Arijit Banerjee
Dr. Pradeep S. Shenoy, Texas Instruments

ABSTRACT

Existing data center power delivery architectures consist of many cascaded power conversion stages. The system-level power delivery efficiency decreases each time the requisite power is processed through the individual stages, and the total power converter footprint increases by each cascaded conversion stage. Innovative approaches are investigated in this dissertation for dc-dc step-down conversion and single-phase ac-dc conversion to improve power delivery efficiency and power density in data centers. This dissertation proposes a series-stacked architecture that provides inherently higher efficiency between a dc bus and dc loads through architectural changes, reporting above 99% power delivery efficiencies. The proposed series-stacked architecture increases power delivery efficiency by connecting the dc loads in series to allow the bulk of the requisite power to be delivered without being processed and by reducing overall power conversion using differential power processing. The series-stacked architecture exhibits voltage regulation and hot-swapping while delivering power to rapidly changing computational loads. This dissertation experimentally demonstrates series-stacked power delivery using real-life computational loads in a custom designed four-server rack. In order to provide a complete grid-to-12 V power delivery for data center applications, this dissertation also proposes a buck-type power factor correction converter that yields high power density between a single-phase grid and the dc bus, achieving 79 W/in^3 power density. The proposed buck-type power factor correction converter improves power density by eliminating the high-voltage step-down dc-dc conversion stage, which is typically cascaded to boost-type power factor correction converters in conventional data center power delivery architectures, and by leveraging recent developments in flying capacitor multilevel converters using wide-bandgap transistors. The buck-type flying capacitor multilevel power factor correction converter presents a unique operation condition where the flying capacitor voltages are required to follow the input voltage at 50/60 Hz. This dissertation experimentally explores the applicability of such an operation by using a digitally controlled six-level flying capacitor multilevel converter prototype.

To Miranda, my parents Ali and Mahinur, and my sister Sebnem.

ACKNOWLEDGMENTS

When I reminisce about my time in Champaign-Urbana, I very much appreciate this little town, which many would sniff at because of its location, for being the place where my path has crossed with those of many amazing people. I am grateful for those who directly or indirectly made my graduate student life remarkable, and will do my best to acknowledge them all here. It is said that the two most important people in a Ph.D. student's life are his/her advisor and spouse. I am so fortunate that I met with both of them at the University of Illinois at Urbana-Champaign.

I must start with thanking Professor Robert C. N. Pilawa-Podgurski, my advisor, for accepting me to his research group at Illinois, because it has been a life-changing opportunity for me. I am grateful for his guidance through my career, for his trust in me during the projects, and for his support in attending many conferences and workshops all around the globe. Owing to the freedom he provided me during my time at Illinois, my graduate school experience exceeded pure research and included organizing conferences and working at several technology companies. His enthusiasm for power electronics teaching and research has always been a source of inspiration for me. I admire the research group culture he created and upheld while guiding more than a dozen graduate students through some of the most challenging projects. I am fortunate to be able to learn from his hands-on experience in power electronics to become a better engineer and researcher.

I would like to thank Professor Philip T. Krein, Professor Arijit Banerjee, Professor Alejandro Domínguez-García, and Dr. Pradeep S. Shenoy for being on my doctoral committee and sharing their valuable time and knowledge. I especially would like to thank Prof. Philip T. Krein for his advice on the ac-dc power conversion portion of this dissertation, and Dr. Pradeep Shenoy for his feedback on the series-stacked power delivery portion of this dissertation, and for his mentorship over the years. I would also like to thank all the professors in the Power and Energy group at Illinois, especially Professor Pete Sauer for holding the Power and Energy Systems group together, his support in organizing the Power and Energy Conference at Illinois, and the amazing speeches he gave during many gatherings. Also, I am thankful to Professor George Gross, as I found out at the beginning of my final exam that he made my admission to Illinois possible by bringing my application to Professor Pilawa-Podgurski's attention.

I am pleased to be a member of the Pilawa-Group at Illinois. I learned a lot from every experience I had with members of the group, past and present: Yutian Lei, Shibin Qin, Christopher Barth, Andrew Stillwell, Thomas Foulkes, Nathan Pallo, Tomas Modeer, Nathan Brooks, Yizhe Zhang, Zichao Ye, Zitao Liao, Josiah McClurg, Derek Heeger, Maggie Blackwell, Pourya Assem, Derek Chou, Wen-Chuen Joseph Liu, Pei Han Ng, Samantha Coday, Jeff Wheeler, Yujia Zhang, Eric Saathoff, Intae Moon, Adwaita Dani, Benedict Foo, Won Ho Chung, Aaron Ho, Ben Macy, Yuqi Li, Roy Bell, Marcel Shuck, and many more. Thank you all for your sincere friendship in this journey. Nowadays, what one can learn from papers and textbooks is the same anywhere in the world. I believe what made my dissertation and research unique in many ways is working with you, so I thank you all for that. I would like to especially acknowledge Andrew Stillwell for his help on the power factor correction portion of this dissertation; Shibin Qin, Zitao Liao and Nathan Brooks for their help on developing the power factor correction microcontroller code; Nathan Pallo, Thomas Modeer, and Maggie Blackwell for their help in flying capacitor multilevel converter design; Thomas Foulkes and Christopher Barth for their help on frequency response analysis and countless discussions in the lab as we work on our own projects; and Josiah McClurg, Yizhe Zhang, Jeff Wheeler, and Yujia Zhang for their help in development of the series-stacked testbed. Your genuine help reduced the workload on my shoulders during the development of this dissertation. It is much appreciated.

Many incredible people at Illinois have become my friends and made my time in graduate school a very delightful experience: In addition to the members of the Pilawa-Group mentioned above, I thank Shamina Hossain-McKenzie, Giang-Chau Ngo, Mehmet Kurt, Itir Akgun, Onur Gur, Sezer Ozerinc, Bilge Acun, Tutku Buyukdegirmenci, Maryam Kazerooni, Andy Yoon, Cecilia Klauber, Adriano Lima Abrantes, Jason Galtieri, Dipanjan Das, Phuc Thanh Huynh, Siddhartha Nigam, Mariola Ndrio, Soteris Demetriou, Dimitra Apostolopoulou, Kai Van Horn, Stanton Cady, Matt Magill, Stanton Cady, Trevor Hutchins, and many more. Thank you all for your friendship; knowing you all has been a true pleasure. I hope we can keep in touch.

The Electrical and Computer Engineering staff have helped me tremendously during my time at Illinois. I would like to thank Joyce C. Mast and Robin Lynn Smith for administering the Power and Energy Systems group, from filing reimbursements to organizing many social events that I had always looked forward to attending. Thanks to Kevin Colravy for managing the research and teaching labs; Beverly Curtis, Stephanie Schneider, Joshua McNattin, and Clint Harper for handling an enormous amount of purchase orders and shipments that made the experimental work in this dissertation possible; and James Hutchinson for proofreading and editing many of my publications, including this dissertation.

I also would like to thank the Fulbright Commission in Turkey and the Institute of International Education for supporting my master's degree at Illinois. Thanks to them, I was able to make it to the United States and attend my dream with the privilege of being a Fulbright Scholar. I would like to thank the IEEE Power Electronics Society for choosing me as the 2017 recipient of the IEEE Joseph John Suozzi INTELEC Fellowship Award in Power Electronics. I also would like to acknowledge the financial support from Texas Instruments, Google, National Science Foundation under Grant 1509815, and Advanced Research Projects Agency - Energy (ARPA-e), U.S. Department of Energy, under Award Number DE-AR0000906 for funding the work in this dissertation in part.

I must extend special thanks to Miranda, my wife-to-be. She has just accepted my proposal which was long overdue. Although there have been times that we had to live on different continents, she has never left me alone in this journey. Working for a Ph.D. is not easy, but I am sure putting up with its roller-coaster effect on my personal life on a daily basis is not any easier. She deserves a special acknowledgment for that. In addition, I proudly thank her for proofreading many of my writings. Since she accepted my coffee offer in Espresso Royale in Urbana, not only has she been a substantial joy of my life, she has also shown me, by example, how to become a better person. I cannot overstate her love and support, for which I am and will be always grateful.

Last but certainly not least, I would like to thank my parents, Ali and Mahinur, and my sister, Sebnem, for their unconditional love and support. My parents, sister and Miranda have made the biggest sacrifice by giving up their time with me during my graduate school. This dissertation, like my all accomplishments, is thus dedicated to them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

As high performance computing and data storage transition towards becoming Internet-based services, the world has witnessed an ever-increasing demand for both the size and capacity of data centers. The growth of cloud-based services and applications shows no sign of slowing down, with additional custom hardware for machine learning algorithms beginning to be deployed at scale in dedicated data centers. Today's data centers accommodate many pieces of information technology (IT) equipment such as data processing units, data storage units, and communication devices. The technological developments in the early 2000s led to a rapid expansion of data centers. As a result, data center energy consumption has increased greatly, which has been noted in several reports [1, 2]. A 2016 report estimated the energy usage of data centers in the United States (US) alone at 70 billion kWh in 2014, corresponding to 1.8% of the total electric energy consumed in the country [3]. A 2018 survey noted that high density IT equipment started to require above 50 kW per rack, mainly driven by artificial intelligence algorithms and high performance computing applications [4]. Power electronics plays a critical role in achieving high power conversion efficiency and high power density in data centers. An in-depth review of power electronics, ranging from utility-scale to chip-level converters in data centers, can be found in [5].

Because IT equipment requires low dc voltage (typically ranging from a few volts to a few dozen volts) to operate, various power converters are needed in data centers to provide low dc voltage from utility and renewable resources. As data processing and cloud services continue to grow, any power conversion loss affects the operational cost of data centers both through the direct cost of electricity and the added cooling requirement. Improved conversion efficiency can be achieved, but this typically results in higher power converter volume, limiting power density in data centers. Alternative architectures that offer high efficiency and small overall volume for data center power delivery must be pursued for sustainable growth of this technology.

1.1 Research contribution

Although a typical data center employs a vast amount of IT equipment, the power delivery architectures for data centers still treat each piece of equipment as a separate load and utilize power delivery methods which were initially intended for single computer applications. This dissertation seeks to demonstrate how architectural changes can increase power delivery efficiency and reduce the requisite converter footprint in data centers. The two crucial power conversion stages, the dc-dc bus converter and the single-phase ac-dc power factor correction converter, are bottlenecks for higher system-level efficiency and power density in data centers. This dissertation focuses on innovative approaches for these power conversion stages.

For the dc-dc bus conversion stage, a series-stacked architecture leveraging an inherently high-efficiency power delivery architecture is proposed. In dc systems where multiple similar loads or sources are present, series stacking and differential power processing (DPP) offer improved overall efficiency [6]. The series-stacking and DPP part of this dissertation builds upon a master of science thesis [7], which was the first hardware demonstration of a server-to-virtual bus DPP architecture for 48 V to 12 V voltage conversion. The new work further shows practical implementation challenges such as hot-swapping and varying input voltage. These results are the first hardware demonstration of a series-stacked power delivery architecture where the IT equipment performs real-life data center operations.

For the single-phase ac-dc power factor correction (PFC) stage, a flying capacitor multilevel (FCML) buck converter leveraging an inherently high power density power conversion technique is proposed. The FCML buck converter offers high power density by employing capacitive elements, which inherently have up to 2-3 orders of magnitude higher energy density than inductors, in energy conversion [8]. Despite the notable power density that the FCML converter can offer, its usage in a buck PFC application presents an uncommon operation condition where the flying capacitors must drastically charge and discharge during each ac half-cycle. This dissertation also presents the first experimental study that explores the FCML buck converter in a single-phase ac-dc application.

1.2 Organization of this dissertation

The remainder of this dissertation is organized as follows.

- Chapter 2 provides brief background on data center power delivery architectures and summarizes conventional ac and dc configurations. Key considerations in data center power delivery such as efficiency, reliability, and backup power are explained in this chapter.

- In Chapter 3, background on series-stacked power delivery architectures is provided. The crucial practical considerations in data centers, initialization and hot-swapping of servers, are addressed conceptually using case studies. Mathematical expressions for processed and delivered power in the proposed virtual bus architecture are derived.

- Chapter 4 explains the bidirectional hysteresis control algorithm, which is a key enabler to achieve high efficiency power delivery in the proposed series-stacked power delivery architecture.

- Experimental results that validate the feasibility of the proposed series-stacked power delivery architecture for data center applications are reported in Chapter 5. Also, a prototype DPP hardware design which combines a differential converter and associated hot-swapping and initialization circuits on a single board is explained in this chapter.

- Chapter 6 provides background information on single-phase ac-dc power conversion, specifically focusing on buck-type PFC conversion. The advantages and challenges of the FCML buck converter in PFC operation are presented here.

- The proposed digital PFC control algorithm is explained in Chapter 7.

- In Chapter 8, an experimental study that explores the FCML buck converter in single-phase ac-dc conversion is reported. A thorough investigation of flying capacitor voltage balancing in FCML buck converters for PFC operation is provided here.

- Chapter 9 concludes the dissertation and suggests future research directions.

The material in this dissertation has been published in part in [9–13], and is reused here with permission.

CHAPTER 2

BACKGROUND ON DATA CENTER POWER DELIVERY

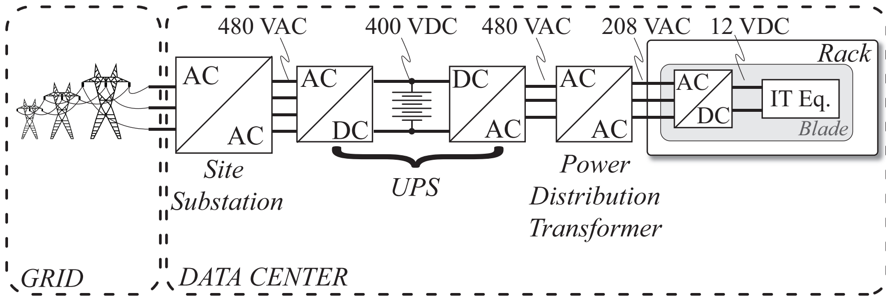
This chapter addresses major aspects of power delivery architectures in data centers including but not limited to efficiency, power density, reliability, integration with renewable resources, and protection.

2.1 Common power delivery architectures in data centers

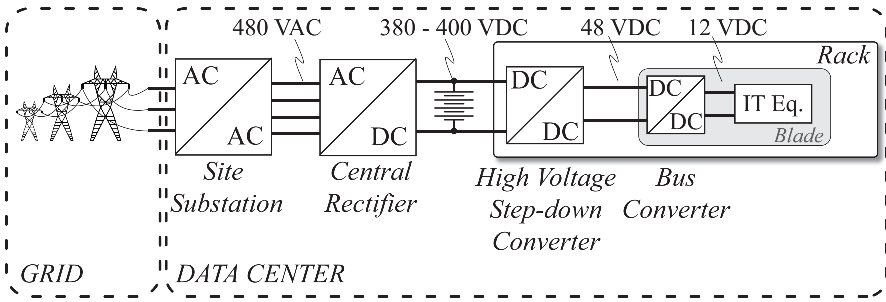
In data centers, since the major energy supply is the ac utility and the primary power consumers (i.e., information technology (IT) equipment) require low voltage dc, both ac and dc power infrastructures are concurrent. As power is delivered from the utility to low voltage dc loads, rectification (power conversion from ac to dc form) can be performed at various points, resulting in different power architecture configurations [14].

Conventionally, utility power is distributed in ac form inside a data center, and then, rectification and voltage step-down conversion are performed at the load end of the power architecture by a dedicated power supply unit (PSU) for each piece of IT equipment. Conversely, utility power can be rectified at the source end of the power architecture (i.e., data center input), and distributed in high dc voltage form within the data center. Then, a dedicated dc-dc converter per IT device steps down the high dc voltage. Alternatively, ac power can be distributed to racks that host the IT equipment, and rectifiers that are in the same rack (or in another rack that is in close proximity) provide dc power to the IT equipment. Simplified diagrams of these common data center power distribution architectures are depicted in Figure 2.1.

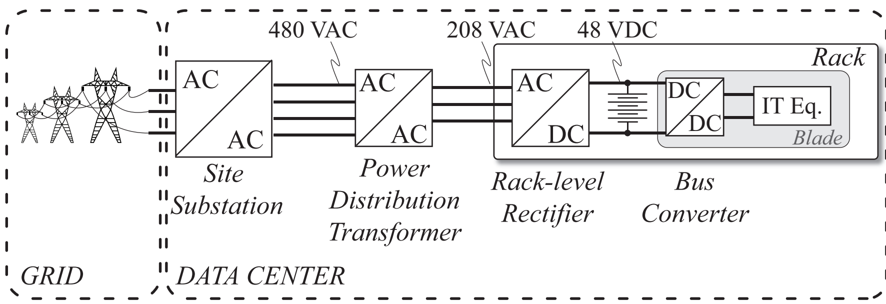
Modern IT equipment employs many dc-dc converters (i.e., point-of-load (POL) converters) in order to step down its input voltage for even lower voltage data processing and storage loads, where the power is eventually needed. For example, Figure 2.2 show key elements of IT equipment that includes a central processing unit (CPU), hard drive, and memory. Since each element requires a different dc voltage, the POL converters step down the IT equipment's input voltage to various



(a) AC power distribution in data centers. A single ac-dc converter provides rectification, voltage step-down and isolation for each piece of IT equipment.



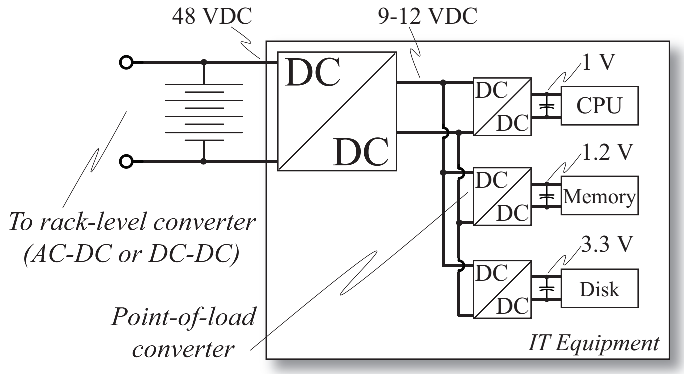
(b) DC power distribution in data center. A central rectifying stage at the data center input provides a high dc voltage which is distributed to the racks. Then, dc-dc converters step down the high dc voltage and provide isolation for each piece of IT equipment.



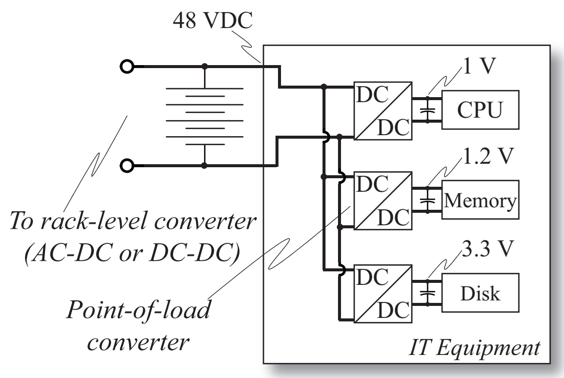
(c) AC power distribution in data center, dc power distribution in the rack or within a few racks. The ac power is delivered to the racks. Then, a rack-level rectification stage provides an intermediate high dc voltage to the rack. Separate dc-dc converters perform final voltage step-down and provide isolation at the IT equipment input.

Figure 2.1: Simplified drawings of common power delivery architectures in data centers. For prioritizing transition from ac to dc power distribution, protection equipment and cooling devices are not depicted in these figures, but of course exist in practical designs and may introduce additional power conversion stages.

lower well-regulated voltage levels. The POL converters may require a lower intermediate bus voltage (9-12 V) as shown in Figure 2.2(a), or they may directly interface the IT equipment input voltage as in Figure 2.2(b). Since the final voltage regulation for the key elements inside the IT equipment is governed by POL converters, conventional power delivery architectures sometimes



(a) 48 V to intermediate bus to POL architecture. A dc-dc converter at the IT equipment input creates an intermediate bus and provides isolation. Then, POL converters step down the intermediate bus voltage to a few volts to provide the ultimate low voltage for the data processing and storage units.



(b) 48 V-to-POL architecture. POL converters step down the IT equipment input voltage directly to a few volts to provide the ultimate low voltage for the data processing and storage units.

Figure 2.2: Simplified drawings of common point-of-load (POL) converter configurations in IT equipment motherboards. POL converters perform the final voltage step-down conversion in the power delivery architecture.

sacrifice precise voltage regulation at the IT equipment input or at the dc bus [15]. This may also enable efficient uninterruptible power supply (UPS) integration in the system, since the voltage typically varies when it is supplied by a UPS, especially during utility-level power loss.

Because of the extensive background, acceptability and well-established standardization of ac distribution in many other applications, a high percentage of existing data centers use variations of the power delivery architectures depicted in Figure 2.1(a) or 2.1(c), which are fundamentally inherited from telecom applications. Recently, dc power delivery architectures, as depicted in Figure 2.1(b), have gained attention, mainly because they involve fewer conversion stages and could potentially simplify the integration of ancillary distributed energy resources, such as solar PV and fuel cells. A well-cited 2008 report has underlined the benefits of high voltage dc power distribution in data centers [16], although the idea of using a voltage level higher than 48 V for data center power distribution appears in the literature as early as 1999 [17]. Over the years the literature has addressed various high dc voltage levels such as 270 V [17], 300 V [18], 325 V [19], 380 V [20, 21], and 400 V [20, 22]; however, the consensus for high voltage dc distribution in data centers eventually appears to have become nominal 400 V. The protracted discussion on voltage levels over 15 years, combined with relatively slow development of standards for IT equipment and power distribution such as ETSI EN 300 132-3-1 [23] for 400 V dc bus voltage, ETSI EN 301 605 [24] for grounding, and IEC 61643-21 [25] for protection, arguably have discouraged short-term adoption of dc distribution in data centers.

Following the potential of dc distribution for data centers noted in [16], ac and dc power distribution architectures for data centers have been both quantitatively and qualitatively compared by both academia and industry [26–30]. In addition, [31] has reviewed some highly cited comparison reports and notes that results vary widely and overstate the benefits of dc power architectures. In order to fairly evaluate comparison studies, it should be noted that over the years significant barriers such as lack of standardization, market share, and compatibility with IT equipment have prevented the widespread adoption of high voltage dc in data centers, while already well-established ac distribution architectures have kept developing to meet expectations.

2.2 Efficiency

In recent years, electricity has become the largest operating cost of data centers; thus, maintaining high power conversion efficiency has become critical. As shown in Figure 2.1, regardless of the preferred distribution architecture, utility power must go through several cascaded power conversion stages before it reaches IT equipment. Since power consumed by IT equipment must be processed by each power converter, the overall power infrastructure efficiency is the product of the efficiency of all power converters, and thus mainly limited by the least efficient power converter. Power delivery studies therefore must consider the entire power conversion stage, from high voltage ac input to the building, all the way down to processor and memory voltages, around 1 V.

Recent literature has focused on efficiency improvements of each major power electronics converter type for data center applications. Development and commercial availability of wide band gap devices have been leveraged in converter designs. Consequently, high power density designs have achieved high-90% efficiencies in three phase [32,33] and single-phase rectifiers [34–36], mid-90% efficiencies in 400 V to 12 V [37–39] or to 48 V [40] dc to dc converters, and mid/high-90% efficiencies in both silicon based [41] and GaN based [42–44] 48 V to 12 V bus converters, and 48 V [45–48] or 12 V [49–53] POL converters for data center applications. Since actual power conversion efficiency in a power converter changes depending on the output power, estimating overall power conversion efficiency of cascaded converters between grid and load is not trivial. A survey of prominent peak and full load efficiency values in [32–53] demonstrates a "best-case" combined efficiency between 93% (if the converters ideally operate at their peak efficiency point) and 86% (if the converters operate at full load) from the ac grid to low voltage dc loads.

One key approach to increasing system-level conversion efficiency is to eliminate – to the greatest extent possible – any double conversion, where voltage is stepped up and down or power is converted from ac to dc or dc to ac more times than the absolute minimum. An example of such a double conversion is the back-to-back ac-dc and dc-ac conversion of the centralized UPS approach shown in Figure 2.1(a), which is one reason why it is no longer a preferred approach. Below are some opportunities to reduce the number of power stages.

- A facility-level battery system for power backup in a high voltage dc power distribution architecture as shown in Figure 2.1(b) does not require an inverter stage while providing backup power to both the IT equipment and auxiliary loads such as lighting and cooling. Since electrochemical energy storage is inherently in dc form, a battery bank for a utility outage or failure scenario can be connected to the high voltage dc bus. Of course, additional circuitry may be needed to connect battery banks to the high voltage dc bus for regulatory or other operating reasons; however, such circuitry does not process the requisite power similar to an inverter that outputs a tightly regulated ac waveform and synchronizes with multiple converters across the bus.

- Twice-line frequency energy buffering is a well-known issue in single-phase rectifiers [54]. The recent Google Little Box Challenge [55] has accelerated research efforts in the area of twice-line frequency energy buffering, and as a result extremely high efficiencies for twice-line frequency energy buffering have been reported in the literature [56]. Nevertheless, moving from single-phase rectification at the IT equipment input to centralized three phase rectification at the high voltage dc bus eliminates the need for twice-line frequency energy buffering and the associated circuitry from the cascaded power chain. Similar benefits can be realized if three-phase rectifiers are employed in an ac distribution approach, generally at the rack level (i.e., rack-level three-phase rectifiers).

- Similar to twice-line frequency energy buffering, active power factor correction (PFC) circuitry is an essential requirement in both single and multiphase high power rectifiers. The most common single-phase PFC architecture is the boost-type converter, meaning the output of the PFC circuit is at a higher voltage than its input. Since IT equipment requires low voltage dc, employing PFC closer to the low voltage load requires a back-end, high conversion ratio, voltage step-down converter. Although a centralized three phase rectification does not eliminate PFC circuitry, it can remove a cascaded voltage step up and down conversion at the IT equipment input, which is potentially counterproductive since the load is at low dc voltage. Here, recent developments in step-down (e.g., buck-derived) PFC rectifiers [32, 57] show promise to achieve increased system-level efficiencies.

At a high level, it may appear that simply reducing the number of conversion stages would increase efficiency. However, one must be careful to consider that for a given power converter volume, a high-step-down power converter generally has lower efficiency than a converter with a modest voltage step-down ratio. This is particularly telling in the case of the 48 V to point-of-load concept, where the conventional architecture shown in Figure 2.2(a) involves first 48 V to 9-12 V conversion, followed by (typically several) 9-12 V to point-of-load (e.g., 1-2 V) converters. While it may be tempting to simply eliminate the two-stage conversion and design a single 48 V to 1 V converter as shown in Figure 2.2(b), such a converter is significantly more difficult to design to be highly efficient and power dense [45–48]. For example, consider the reference design of [58], which represents a single-stage buck converter, achieving a peak efficiency of 84%. Other more complex topologies likewise achieve limited performance in both efficiency and power density. In comparison, recent hybrid switched-capacitor power converters have been shown to achieve near 99% efficiency with extraordinarily high power density (106.5 kW/L) for 48 V to 12 V conversion [41], and separate 12 V to 1 V converter can similarly achieve high overall efficiency and power density [49, 51–53]. While there may be other considerations (such as reliability, cost, reduced complexity, etc.) to prefer fewer numbers of stages, the above discussion highlights that increased efficiency is not necessarily an outcome of this approach.

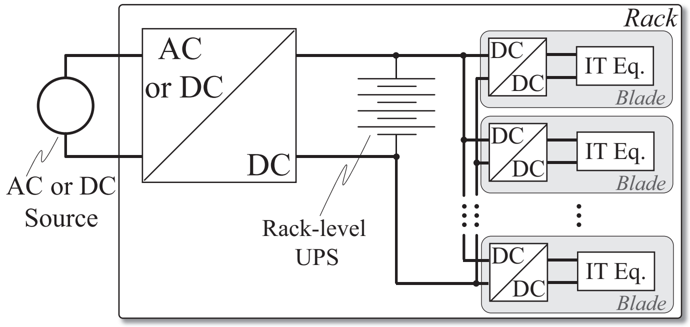
2.3 Reliability

Maintaining high reliability in data centers is crucial because of our society's dependence on uninterrupted IT services. Today, any outage of IT services can have a large impact, both financial and in terms of societal impact. A typical target reliability for a data center is 99.99% uptime (often called "four nines"), which corresponds to 52.5 minutes downtime per year [59]. This requirement, combined with maintaining high efficiency, makes data center power delivery architecture design challenging. Fortunately, similar to improving the power delivery architecture efficiency, reducing the number of cascaded power stages typically reduces the overall risk of system failure and the mean time between failures. Therefore, opportunities to reduce the number of power conversion stages in dc data centers facilitates higher reliability and uptime. Elimination of conversion stages such as power distribution transformers and the inverter at the UPS output in high voltage dc distribution is considered a major advantage for dc data centers [22, 60, 61]. However, the analysis in these past works is qualitative and the details are unclear. A 2010 study quantitatively compared the reliability of ac and dc power distribution for data centers with emphasis on UPS, and concluded that dc distribution would be more reliable than ac distribution without supplementary effects of redundancy [27]. A more recent quantitative reliability analysis for data centers considering UPS, power converter failure mechanisms, and redundancy options is missing in the literature. Nevertheless, reducing the number of conversion stages alone is not sufficient to assure reliable power distribution; backup power and redundancy must be incorporated in data center power architecture design to achieve the desired reliability level.

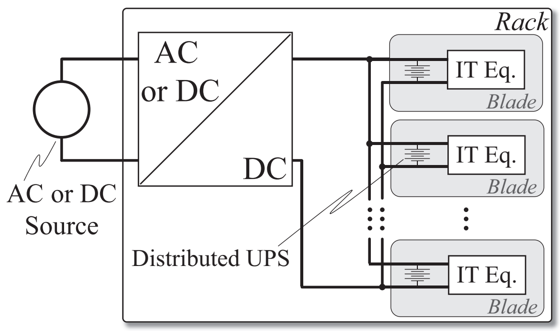
2.4 Backup power

Utility power loss is an expected scenario in data centers instead of a failure. Data centers employ uninterrupted power supplies (UPS), backup generators, and recently fuel-cells at specific points in the power delivery architecture to be able to compensate for both utility loss and power stage failure. Various possible configurations preferred in recently developed data center power delivery architectures are depicted in Figure 2.3.

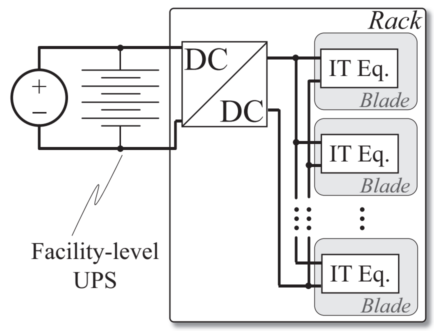
In case of utility loss, the facility level UPS must be able to provide enough backup power to critical loads until backup generators can initialize and output sufficient energy to the facility. On the other hand, UPS placement close to the load (IT equipment) can compensate for any component



(a) Rack-level UPS. UPS is located inside the racks and provides backup power for multiple pieces of IT equipment.



(b) Distributed UPS. Each piece of IT equipment has a dedicated UPS. This configuration offers uninterruptible power in case of any converter failure.



(c) Facility-level UPS. UPS is located outside of the rack and provides backup power for multiple racks.

Figure 2.3: Various UPS configurations for data centers.

failure between the utility and the IT equipment. Uptime Tier Certification requires data centers to have on-site fuel storage for at least 12 hours of utility loss [62].

Recent data center power delivery designs are moving away from central UPS double-conversion towards rack-level UPS single-conversion in ac distribution architectures (e.g., Figure 2.1(c)). While there are some benefits in terms of maintenance and costs associated with a central UPS system, rack-level UPS can also help mitigate power distribution faults in data centers, as each rack can operate directly from its own UPS.

2.5 Redundancy

Redundancy is typically achieved through the incorporation of additional and separate power conversion stages, UPS, and power distribution paths in data center power infrastructure. Redundant components may be operated at all times (e.g., each running at partial load to increase peak ef-

Table 2.1: Tier certificate requirements summary

	Tier I	Tier II	Tier III	Tier IV
Redundancy level	N	N+1	N+1	N After any failure
Distribution path	1	1	1 active, 1 alternate	2 active

ficiency) but, strictly speaking, are only needed to meet the power demand of the load in case of failures. Typical redundancy levels for data centers are N+1, 2N and 2N+1, where N represents the number of power converters or UPS systems in parallel to meet the load demand. The Uptime Institute defines Tier Classification levels for data centers depending on the redundancy level of the data center [62]. Tier I represents basic data center infrastructure without any redundancy. Tier II certification requires redundant power stages and UPS; however, the power distribution path is not redundant. Tier III certification requires the data center to have both redundant power stages and multiple independent power distribution paths, although only one distribution path is actively used at any time, while the other is for maintenance purposes. Tier IV certification requires redundant power stages and multiple active power distribution paths configured to serve the entire data center under any infrastructure failure. Tier Certificate requirements are summarized in Table 2.1 and details can be found in [62].

2.6 Isolation and grounding

Electrical (galvanic) isolation has been an essential part of data center power delivery architectures. Provided by transformers, electrical isolation offers to filter grid disturbances, harmonic currents, and electrical noise. Also, electrical isolation limits ground loops and circulation of dc currents between IT equipment and racks [14]. Because of these benefits, electrical isolation through isolation transformers is a recommended practice under IEEE STD 1100 for ac power distribution architectures [63].

Electrical isolation may be implemented at several points throughout the data center power delivery architecture. In Figure 2.1 common power delivery architectures were shown, where power distribution transformers provide electrical isolation at 50/60 Hz. In addition to power distribution transformers, isolated dc-dc converters can also be used to provide electrical isolation in data center power delivery architectures.

The most obvious shortcoming of introducing electrical isolation is the trade-off between efficiency and density. 50/60 Hz transformers can be highly efficient but bulky, while high frequency transformer design and optimization are nontrivial to be included in dc-dc converters. A common conception in the data center power delivery area is that an added isolation stage reduces power conversion efficiency by 3%. For instance, in [37] losses of a carefully optimized transformer in an LLC converter for data center applications correspond to 1.5-3% of the rated power. Although the exact percentage penalty value in isolated dc-dc converter efficiency is questionable, it may not be inherently linked to high frequency transformer designs. Recent developments in transformer design, wide bandgap devices [37, 39, 64], and optimization approaches [65] enable high efficiency and compact isolated dc to dc converters. Also, in order to provide only electrical isolation (without voltage conversion), unity transformation ratio has been demonstrated in dc to dc converters for data center applications at 400 V [66] and 48 V [67].

Although electrical noise, ground loops and circulation of dc currents may still be present in data centers, enforcing electrical isolation may not be the only practice to overcome such issues. For example, modern communication links typically provide inherent isolation, either through the medium itself (fiber optics), signal isolation transformers (Ethernet), or ac coupling capacitors (high speed serial links). With adequate system grounding, high safety may be obtainable without requiring galvanic electrical isolation. An example of this transition is the case of grid-tied photovoltaic inverters, which until recently were required to have galvanic isolation in the US, while transformerless inverters were adopted earlier in Europe since they offered higher power efficiency and density at a lower cost [68]. Similar efforts may lead to elimination of electrical isolation from data center power distribution architectures in the future.

Proper grounding is an essential requirement in power infrastructure for protection, safety and signal integrity. Lightning protection is the primary driver for grounding in data centers. In addition, grounding contributes to safety from electrical hazards by routing damaging currents away from IT equipment and personnel. Proper grounding also enables a common voltage reference for the overall electrical system in a data center, including power infrastructure and communications equipment.

2.7 Protection

IT equipment and power infrastructure in data centers represent large investments. Therefore, any damage due to power system faults must be prevented by protection equipment such as fuses, relays and circuit breakers. Protection is activated to isolate failed IT equipment or section off the power infrastructure from the rest of the system. Therefore, the location of protection equipment in the power infrastructure is vital to enable isolation of any IT equipment and power stages whenever needed.

A periodic current zero-crossing is inherent in ac distribution architectures as the polarity of voltage and current alternates 50 or 60 times per second; however, in dc distribution voltage and current are controlled to be constant values and do not naturally cross zero. The lack of a periodic zero-crossing of dc voltage and current complicates protection equipment design if dc power delivery is preferred, and can result in self-sustaining faults. In addition, although sometimes overlooked, abrupt interruption of dc current results in high current slew rates (di/dt), which induce high voltages in any parasitic inductive loops along the power path. This may endanger semiconductors and capacitors if not considered.

The basic operation principle of protection equipment (i.e., moving electrical contacts away from each other when triggered) is similar in ac and dc systems, but with added challenges for dc protection. In order to successfully extinguish an arc in dc distribution architectures, the electrical contacts must move not only farther away from each other, but also faster than in ac distribution. Alternatively, protection equipment involving electronics to force the dc current towards zero can be used.

2.8 Renewable and distributed energy sources

Photovoltaics (PV) [69] and fuel cells (FC) [70] are two energy sources for data centers commonly preferred by the industry. Since both PV and FC are inherently dc energy sources, their integration into dc power infrastructure is simplified. While it is theoretically possible to design a PV array to directly interface with a high voltage bus in a power delivery architecture, typically it is preferred to do so through a dedicated dc-dc converter that ensures that the PV system operates at its maximum power point, which varies with irradiation and temperature. Alternatively, multiple panel-embedded power converters [71, 72] can be employed to provide this voltage conversion, in addition to improved PV array performance.

2.9 Cooling

In addition to computational, data storage, and networking loads, cooling requires substantial electrical energy in data centers. Since IT equipment mainly consists of digital circuits, combined with losses in the power distribution architecture, almost all energy consumed in data centers results in heat that must be dissipated. In order to maintain safe operation of IT equipment, temperature control is critical at all times. Therefore, the reliability discussion above also applies to the cooling system, and backup power should be designed to support the cooling load in case of a utility outage.

In order to provide sustainable heat removal from IT equipment, typical data center cooling infrastructure includes air conditioning equipment and a chilled water system, which require pumps and fans. Such equipment involves electric motors which are typically controlled by adjustable speed drives (ASDs) for maximum efficiency and performance. Similar to UPSs, dc data centers require only an inverter stage for ASDs, compared to ac-driven ASDs, which first perform rectification, followed by the adjustable frequency and voltage inverter.

Since power converters are in close proximity to IT equipment, existing cooling infrastructure in data centers is leveraged to extract heat generated in power converters. Conventionally, heat sinks are attached to power transistors to improve heat transfer quality by increasing surface area. On the other hand, recent literature explores innovative heat transfer mechanisms using jumping droplet condensation for actively cooling hot spots in power converters [73, 74].

2.10 Total cost of ownership

Total cost of ownership (TCO) is an important consideration in data center design because building a data center is a substantial investment for businesses. TCO extends beyond the cost of power equipment and electricity, but as far as power architecture is a concern, there are two main expenses: capital and operational expenses. Capital expenses include up front costs such as installation and equipment purchases, while operational expenses include electricity and maintenance cost and therefore involve efficiency and reliability aspects. A detailed explanation of TCO beyond power infrastructure can be found in [59, 75].

Unfortunately, TCO is rather overlooked in the literature, since the primary motivation for most work is efficiency and reliability. A 2011 white paper quantifies oversizing of power infrastructure as the primary cost driver of data center TCO and suggests measuring the TCO on a per-rack basis [76]. Power infrastructure oversizing can be as severe as triple of what is needed because of uncertainties in final power demand and inadequate assumptions [77]. A three phase rectifier for data center applications is optimized for TCO in [32]. In [61], the cost advantage of high voltage dc data centers is reported as 15% less in capital cost and 36% less in lifetime cost, but detailed analysis is missing. It should be noted that improvements in efficiency and reliability do not translate to business decisions unless TCO analysis is incorporated into the benefits.

Critically, because TCO analysis requires accurate field data regarding reliability, it is likely that estimates of hypothetical designs without empirical results vary widely. Hence, TOC aspects of data centers are difficult to assess by researchers at this time. Moreover, major corporations that carefully track these metrics (e.g., Facebook, Amazon, Google, etc.) generally view these numbers as key competitive features of their respective designs, and are unlikely to share them with researchers.

As this chapter briefly explained, data center power delivery architectures ultimately aim to provide high quality power to IT equipment. The IT workhorses in data centers are servers, which are essentially collections of data processing and storage units, and operate at typically 12 V or 48 V dc voltage. Various power delivery architectures are available in various voltage and power levels throughout data centers as explained in this chapter. The remainder of this dissertation focuses on single-phase rectification (i.e., 240 V_{RMS} to 48 V dc) and bus conversion (i.e., 48 V dc to 12 V dc) stages.

CHAPTER 3

SERIES-STACKING AND DIFFERENTIAL POWER PROCESSING FOR DATA CENTER POWER DELIVERY

This chapter focuses on high efficiency dc-dc power conversion for data center power delivery through series stacking and differential power processing.

3.1 Motivation

Although today's data centers employ a large number of servers, conventional data center power delivery architectures are still based on designs originally developed for single computer applications. As the number of servers and their power consumption increase, conventional architectures suffer from increased power conversion loss because a more efficient power converter does not translate to reduced power conversion loss. Shown in Figure 3.1 is an example conventional data center power delivery architecture which was mentioned in Section 2.1. Here, it is depicted again to facilitate the discussion.

In the power delivery architecture shown in Figure 3.1, ac power is delivered to each server rack, and a central rectifier regulates a dc bus (typically at 48 or 400 V) for the rack. Following this, each server has a dc-dc converter that steps down the high dc voltage to a suitable motherboard voltage for the server (typically at 12 or 48 V). A key observation from inspecting the conventional power

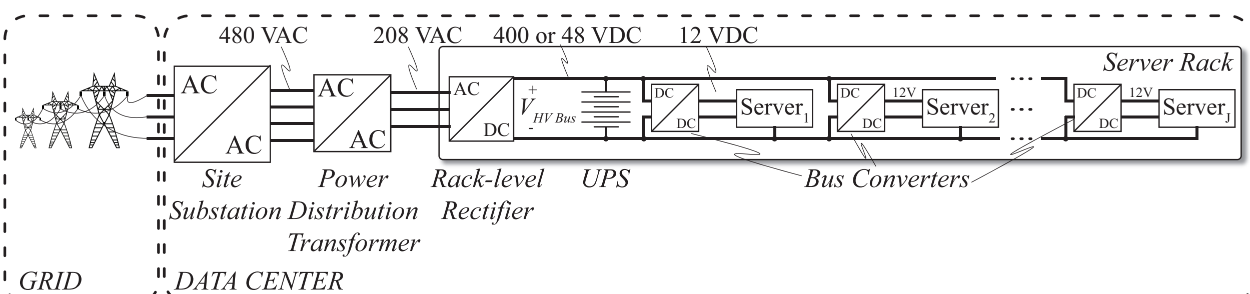


Figure 3.1: AC power distribution in data center, dc power distribution in the rack. The ac power is delivered to the racks. Then, a rack-level rectification stage provides an intermediate high dc voltage to the rack. Separate dc-dc converters perform final voltage step-down at the server input.

delivery architecture depicted in Figure 3.1 (or the other architectures depicted in Chapter 2) is that system-level efficiency is limited by efficiency of the power converters, since *full* server power must be processed during server operation. In other words, there exists a direct coupling between delivered power and associated power losses at each converter, which intensifies as rated server power or the number of servers increases.

This dissertation presents an inherently more efficient dc-dc conversion architecture for data centers by proposing a system-level solution that decouples power conversion losses from delivered power, rather than focusing on efficiency improvements in individual dc-dc converters. By electrically connecting the servers in series, the proposed power delivery architecture greatly reduces requisite power conversion inside server racks. This architecture yields significantly reduced power loss, and thus higher system-level efficiency.

3.2 Background on series stacking and differential power processing

Low-voltage elements (sources or loads) are commonly connected in series to interface with a high dc bus voltage in applications such as photovoltaic sources, battery systems, and LEDs. In these applications, the elements are connected in series because the desired operating voltage of each element is lower than the available or desired dc bus voltage. The large number of low-voltage servers in data centers represents a similar scenario, where series connection within a server blade may be beneficial.

Series stacking and various configurations of differential power processing (DPP) architectures have been proposed for dc systems where a group of low-voltage elements must be connected to a higher voltage dc bus [6]. Series-stacked architectures and differential or partial power processing have been explored in various fields, such as solar photovoltaics [72, 78–93], digital circuits [94–102], biomedical implants [103, 104], and active battery cell balancing applications [105–113] and have provided significant performance improvements.

Series-stacked power delivery for data center applications initially attempted to regulate the server voltages in software [114]. Although this work achieved reasonable voltage balancing with power-aware load balancing web traffic management software, computational performance was slightly compromised, and each series-stacked server also needed a UPS in parallel. In order to demonstrate the potential of series-stacked power delivery in data center applications, various DPP techniques have been theoretically and experimentally studied as a hardware solution. Server-to-virtual bus DPP [115], server-to-server DPP [116], server-to-bus [117] and hybrid DPP [118–120] architectures have been verified experimentally. In [115] and [116], series-stacked architectures are also compared with the conventional power delivery architecture shown in Figure 2.1(c) based on best-in-class dc-dc converters, and have achieved up to 40 times reduction in average power conversion loss under real-world data center operations such as web traffic management and computation. Although various DPP architectures exist as summarized and qualitatively compared for data center applications in [115], this dissertation focuses on server-to-virtual bus DPP architecture and seeks to demonstrate how a series-stacked topology can be applied to server power delivery.

Figure 3.2 shows a series-stacked architecture with the server-to-virtual bus DPP technique, which is the architecture used in this dissertation. Similar to the other DPP architectures described in [121], in this architecture, low-voltage loads (servers) are connected in series in order to directly interface to a higher voltage dc bus. This series connection not only eliminates the step-down voltage conversion stage, but also enables bulk power consumed by servers to be delivered without being processed by a dc-dc converter. Since series-connected loads must conduct the same amount of current, any power mismatch between servers can alter their input voltages if not compensated. In order to compensate power mismatch between series-stacked servers, the server-to-virtual bus DPP architecture features bidirectional and isolated dc-dc converters (referred to as differential converters in this work) and a shared energy reservoir (i.e., the virtual bus). The virtual bus is essentially a capacitor bank that is isolated from the dc bus and connected in parallel with the secondary sides of the differential converters. In comparison to the conventional systems in Figure 2.1(c), where each server's power converter must process *full* server power, the server-to-virtual bus architecture only processes the *difference* in power between servers. This architecture

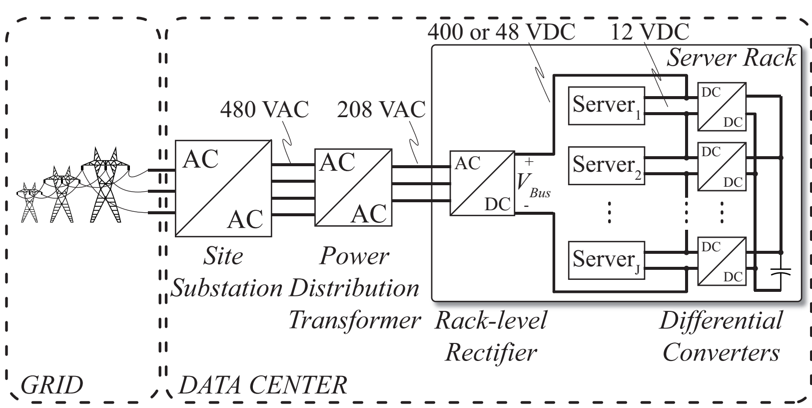


Figure 3.2: Proposed server-to-virtual bus DPP power delivery architecture for server racks.

thus separates the total amount of processed power from delivered power, resulting in considerable reduction of power conversion losses, in particular when individual server power consumption is similar.

There are a few other key advantages of the server-to-virtual bus DPP architecture. First, since the virtual bus voltage is an unrestricted design parameter, it can be chosen to be the same as the nominal server voltage. This not only eliminates the voltage conversion need in the system, but also enables the differential converters to be designed using low-voltage high-frequency switches on both the primary and secondary side of the transformer. Another key benefit of the proposed virtual bus architecture is the scalability of the approach. Here, each DPP converter must only be rated for the server voltage and the virtual bus voltage, while the number of servers can be increased to accommodate a higher bus voltage, given that the transformer is properly rated. Moreover, since in the server-to-virtual bus DPP architecture any differential converter in the series stack can exchange power with the others by injecting current to or extracting current from the virtual bus, the power mismatch order of the series stack has no effect on the total amount of processed power as long as the virtual bus voltage is regulated within a limit. It should be noted that virtual bus voltage regulation slightly increases total processed power and control complexity compared to the server-to-bus DPP architecture. However, as shown in [10], a suitable control implementation can achieve excellent regulation and efficient power delivery.

Note that while the virtual bus capacitor is shown as a discrete element in Figure 3.2, it can also be distributed among the secondary side capacitors of the differential converters. Also, although the power is distributed in ac form within the data center in Figure 3.2, neither the proposed series-stacked architecture nor the discussion in this dissertation changes if the power is distributed in dc form within the data center.

It should be also noted that the virtual bus capacitor topology employed here is quite similar to active battery balancing using isolated power converters [106, 107, 109, 112]. Moreover, it has been successfully employed in differential power processing architectures for photovoltaic applications [82, 88]. However, this work is the first in the literature that uses a virtual bus topology for regulating series-stacked server voltages which inherently exhibit a more dynamic power profile than batteries and photovoltaics.

3.3 Analysis of the server-to-virtual bus DPP architecture

The power delivery nature of the series-stacked and DPP architecture enables the processing of only the difference in power between series-connected servers. The total processed power varies depending on average power consumption of each series-connected server. Derivation of the total processed power in the system for a given load distribution scenario is thus an important consideration when evaluating the proposed system. This section analyzes the proposed architecture by deriving total average power processed and comparing it with the total average delivered power to the servers for various power consumption distributions under steady-state and hot-swapping operation.

3.3.1 Steady-state operation

Shown in Figure 3.3 is a schematic diagram of the server-to-virtual bus DPP architecture that facilitates the discussion. Here, an ideal dc voltage source models the output of an ac-dc converter (or connection to a high voltage dc bus) which provides power (by V_{Bus} and i_{Bus}) to the series-stacked servers. The terms $i_{S,j}$ and $v_{S,j}$ represent the jth server current and voltage, where the subscript j may refer to any server in the series stack, and J is the total number of servers (i.e., $j\epsilon\{1, 2, 3, \ldots, J\}$) in the series stack. The virtual bus capacitor voltage and current are represented by v_{VB} and i_{VB}, respectively. The input and output currents of the dc-dc converters are shown separately as $i_{\Delta,j}$ and $i_{\delta,j}$. Since each dc-dc converter is connected between a server in the series stack and the virtual bus, the terminal voltages of the jth dc-dc converter are $v_{S,j}$ and v_{VB}.

In the server-to-virtual bus DPP architecture shown in Figure 3.3, KVL around the series stack and KCL at every intermediate node of the series stack result in

$$V_{Bus} = \sum_{j=1}^{J} v_{S,j} \tag{3.1}$$

and

$$i_{Bus} = i_{S,j} + i_{\Delta,j}, \quad \forall j, \tag{3.2}$$

respectively. Also, since the virtual bus capacitor is connected in parallel with all dc-dc converters,

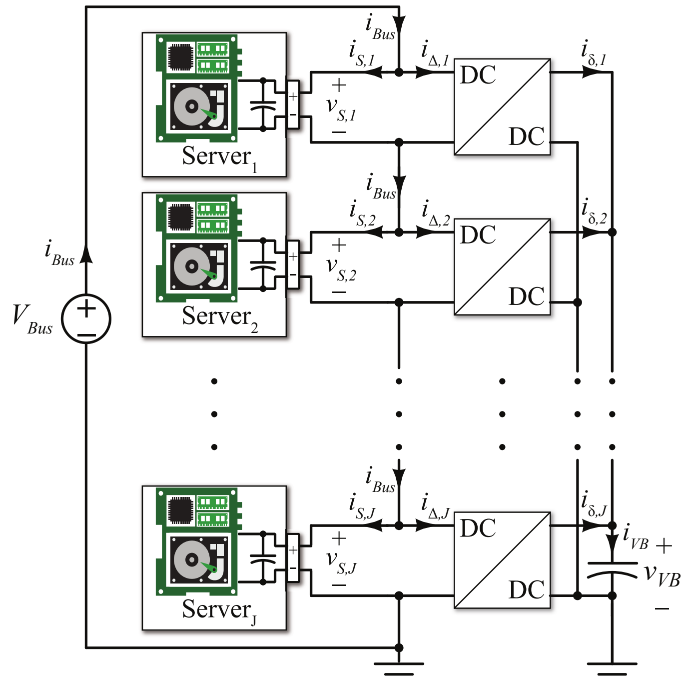


Figure 3.3: Server-to-virtual bus DPP architecture schematic used for analysis.

the virtual bus current is given by

$$i_{VB} = \sum_{j=1}^{J} i_{\delta,j}. \tag{3.3}$$

Assume that all series-stacked server voltages and the virtual bus voltage are regulated to their nominal steady-state values ($V_{S,nom}$ and $V_{VB,nom}$, respectively) with a general time period. (A control algorithm that accomplishes this will be presented in Chapter 4.) Averaging (3.1), (3.2) and (3.3) over this general time period results in

$$V_{Bus} = \sum_{j=1}^{J} V_{s,j} = J \times V_{S,nom}, \tag{3.4}$$

$$I_{Bus} = I_{S,j} + I_{\Delta,j}, \quad \forall j, \tag{3.5}$$

and

$$I_{VB} = \sum_{j=1}^{J} I_{\delta,j}, \tag{3.6}$$

respectively. Recall that the virtual bus is a capacitive buffer. As the controller regulates its voltage to a constant value ($V_{VB,nom}$), the average current into the virtual bus becomes zero over the time period. For simplicity of the analysis, assume that the differential converters are ideal, and they provide one-to-one voltage conversion (i.e., $V_{S,nom} = V_{VB,nom}$, $i_{\Delta,j} = i_{\delta,j}$ and $I_{\Delta,j} = I_{\delta,j}$). Therefore,

$$I_{VB} = \sum_{j=1}^{J} I_{\delta,j} = 0 \implies \sum_{j=1}^{J} I_{\Delta,j} = 0. \tag{3.7}$$

The node current equations given by (3.5) are valid for every server and differential converter pair throughout the series stack, as can be seen in Figure 3.3. The sum of J node current equations gives

$$J \times I_{Bus} = \sum_{j=1}^{J} (I_{S,j} + I_{\Delta,j}) = \sum_{j=1}^{J} I_{S,j} + \sum_{j=1}^{J} I_{\Delta,j}. \tag{3.8}$$

Given the constraint of (3.7), (3.8) simplifies to

$$I_{Bus} = \frac{1}{J} \sum_{j=1}^{J} I_{S,j}, \tag{3.9}$$

which states that the average bus current equals the mean of the time average of the server currents.

The average processed power by the differential converters (i.e., the total processed power in the system) is

$$P_{processed} = \sum_{j=1}^{J} |P_{\Delta,j}| = \sum_{j=1}^{J} |V_{s,j} I_{\Delta,j}|, \tag{3.10}$$

where $P_{\Delta,j}$ is the average processed power by the jth differential converter. With a control algorithm that successfully regulates the series-stacked server voltages to their nominal values ($V_{S,nom}$), (3.10) simplifies to

$$P_{processed} = V_{s,nom} \sum_{j=1}^{J} |I_{\Delta,j}|. \tag{3.11}$$

Moreover, using the KCL constraint given by (3.2), (3.11) can be rewritten as

$$P_{processed} = V_{s,nom} \sum_{j=1}^{J} |I_{Bus} - I_{S,j}|. \tag{3.12}$$

Since the average bus current (I_{Bus}) is equal to the mean of the time average of the server currents, as in (3.9), the total amount of processed power given by (3.12) can also be expressed as

$$P_{processed} = V_{s,nom} \sum_{j=1}^{J} \left| \left(\frac{1}{J} \sum_{j=1}^{J} I_{S,j} \right) - I_{S,j} \right|. \tag{3.13}$$

It is illustrative to compare the total processed power with the total delivered power to the servers in the server-to-virtual bus DPP architecture. Assuming ideal converters, the total delivered power to the servers is the sum of all server powers, and it can be expressed as

$$P_{delivered} = \sum_{j=1}^{J} P_{s,j} = V_{s,nom} \sum_{j=1}^{J} I_{S,j}, \tag{3.14}$$

where $P_{s,j}$ is the average power consumed by the jth server. Recall that I_{Bus} is the mean of the time average of the server currents in (3.9), which implies

$$\min(I_{S,j}) < I_{Bus} < \max(I_{S,j}) \quad \forall j. \tag{3.15}$$

Since $I_{S,j} > 0$ for all j when all servers are operational, it can be observed that the processed power by (3.12) is always less than the delivered power by (3.14). This is an important feature of the series-stacked power delivery architecture.

Case Study I

A statistical case study is performed in order to compare the processed power and delivered power in the server-to-virtual bus DPP architecture using (3.13) and (3.14). In this case study, a server rack consisting of 32 servers (each rated at 300 W) is used to illustrate the effect of various mismatch conditions. The average computational load for the server rack is swept from 50% to 95% of the rated power. For every average computational load scenario, the computational load range within the server rack is randomly assigned using a Gaussian distribution, and examined for 1000 iterations, causing different mismatch conditions. Equations (3.13) and (3.14) are used to calculate the total processed and delivered power at each iteration, and then the results of 1000 iterations are averaged for every scenario. The result is plotted in Figure 3.4.

Figure 3.4 shows the total processed and delivered power in the server-to-virtual bus DPP architecture versus average computational load scenarios (and computational load ranges). As the average computational load is increased from 50% to 95% of rated power, the computational load range narrows (i.e., there is less mismatch between the series-stacked servers). As expected, the delivered power increases as the average computational load increases. However, the processed power decreases as the average computational load increases. This is because the average server current is delivered through the dc bus without being processed and the Gaussian distribution of

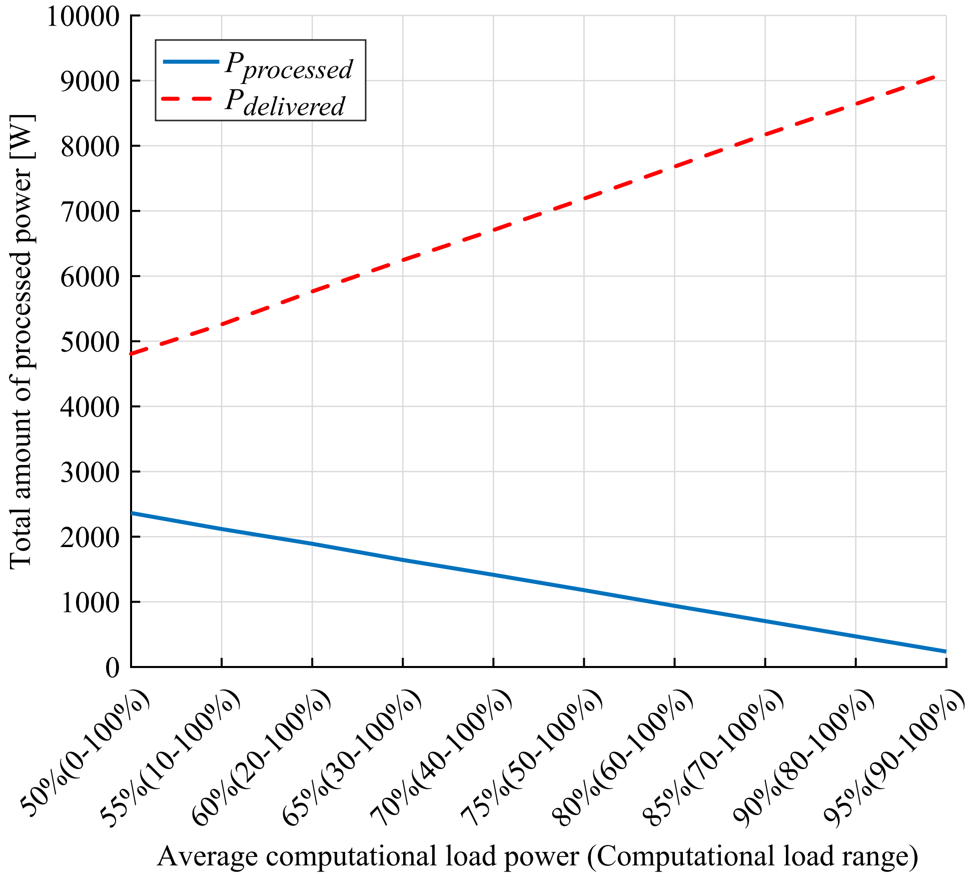


Figure 3.4: A statistical comparison between total processed power and delivered power in the server-to-virtual bus DPP architecture, assuming ideal converters and a Gaussian distribution of computational load mismatch.

the computational load range narrows as the average computational load increases.

3.3.2 Hot-swapping operation

An operation essential for servers in data centers is hot-swapping, which is inserting or removing individual servers while other servers in a rack are operational. Hot-swapping may be required if a server is intentionally removed for maintenance or if a server unexpectedly fails and requires repairs. Regardless of the reason, when a server is hot-swapped in a series-stacked power delivery architecture, the flow of bus current must be maintained by the differential converters and the virtual bus, or a bypass switch must keep the remaining servers operational. Maintaining the flow of bus current through differential converters and the virtual bus will increase processed power in the system. On the other hand, using a bypass switch to maintain the flow of bus current by shorting the differential converter corresponding to the hot-swapped server would result in increased

Table 3.1: Processed and delivered power in the example operations of the six-server rack in Figure 3.5, calculated by (3.13) and (3.14)

	Fig.3.5(a)	Fig.3.5(b)	Fig.3.5(c)	Fig.3.5(d)	Fig.3.5(e)	Fig.3.5(f)
$P_{delivered}$ $[W]$	1701	1422	1125	828	558	288
$P_{processed}$ $[W]$	63	474	750	828	744	480

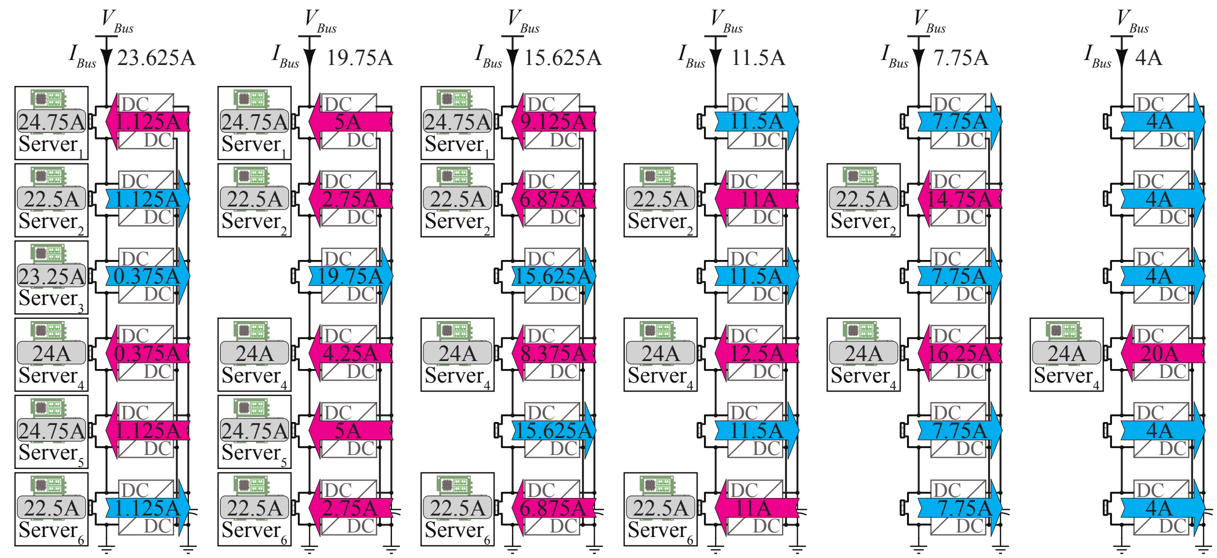
input voltage for the remaining servers in the series stack. However, this can be mitigated by temporarily reducing the dc bus voltage, or by designing wider input voltage range servers and differential converters. This dissertation focuses on hot-swapping achieved by maintaining bus current through differential converters and the virtual bus.

In the remainder of this section, hot-swapped operation using differential converters and the virtual bus to maintain bus current is conceptually explained in a case study through an example server rack that includes six series-stacked servers. Mathematical derivation of processed and delivered power in the system is extended for hot-swapped operation. In a final case study, the 32-server rack and two of the average computational load scenarios in *Case Study I* are studied again to show processed and delivered power under hot-swapped operation.

Case Study II

Case Study I compared processed and delivered power in the server-to-virtual bus architecture when the computational load distribution within series-stacked servers follows a Gaussian distribution. Although hot-swapping or a server failure occurs rarely, it represents a severe mismatch in terms of computational load distribution for a series-stacked architecture, thus increasing processed power in the system.

Figure 3.5 illustrates examples of normal and hot-swapped operation of a six-server rack employing the server-to-virtual bus DPP architecture. Assuming 300 W rated power and 12 V nominal voltage servers, the annotated server currents in Figure 3.5(a) represent 95% average computational load and ±5% computational load range within the rack, similar to *Case Study I*. It can be seen in Figure 3.5(a) that the bus current equals the average of the six server currents. Each differential converter injects or rejects the difference in current between its corresponding server current and the bus current given by (3.5). The total current into the virtual bus capacitor is zero, resulting in 63 W processed power as in (3.11), and 1701 W delivered power as in (3.14).



(a) All servers are operational. (b) One server is swapped out. (c) Two servers are swapped out at the same time. (d) Three servers are swapped out at the same time. (e) Four servers are swapped out at the same time. (f) Five servers are swapped out at the same time.

Figure 3.5: Examples of normal and hot-swapped operations of a six-server rack. Annotated server currents correspond to a 95% average computational load with ±5% computational load range scenario when all servers are operational. Annotated differential currents show direction and average amount of current flow, assuming ideal converters.

Shown in Figure 3.5(b) is an example where the third server is swapped out from the six-server rack, under the same load distribution as in Figure 3.5(a). Treating the absence of the third server as the third server consuming 0 A, the differential currents can be calculated from (3.5). Note that the differential converter for the third server ensures the flow of bus current in the series stack by injecting it into the virtual bus. In order to ensure virtual bus regulation, the remaining differential converters share the extra current on the virtual bus, and inject it back to their servers. For this example, the processed and delivered power are 474 W and 1422 W, respectively.

Figures 3.5(c) through 3.5(f) illustrate examples in which the servers are swapped out from the six-server rack one by one. It can be observed that the bus current decreases as more servers are swapped out since the bus current equals the average of the six server currents (again, treating the swapped-out server currents as 0 A). Also, the differential converters of the swapped-out servers inject the bus current to the virtual bus while the remaining differential converters share the total extra current and inject it back to their servers. The processed and delivered power for each example in Figure 3.5 are given in Table 3.1.

Processed and delivered power during hot-swapping

Recall the expressions for processed power (i.e., (3.13)) and delivered power (i.e., (3.14)) in the server-to-virtual bus DPP architecture with ideal differential converters. Since the physical location of a server in the series stack does not affect (3.14) and (3.12), the hot-swapped servers can be lumped on top of the series stack in order to simplify the expressions of the following equations:

$$P_{delivered} = V_{s,nom} \sum_{j=H+1}^{J} I_{S,j}, \tag{3.16}$$

and

$$P_{processed} = V_{s,nom} \left[\sum_{j=1}^{H} I_{Bus} + \sum_{j=H+1}^{J} |I_{Bus} - I_{S,j}| \right], \tag{3.17}$$

where H is the number of simultaneously hot-swapped servers (i.e., $I_{S,j} = 0$ for $j \epsilon \{1, 2, \ldots, H\}$, and $I_{S,j} > 0$ for $j \epsilon \{H+1, H+2, \ldots, J\}$).

In order to further simplify (3.17), assume that the bus current is less than all remaining server currents during any hot-swap, which means $|I_{Bus} - I_{S,j}| < 0$ for $j \epsilon \{H+1, H+2, \ldots, J\}$. Note that this assumption is the situation studied in *Case Study II* and Figure 3.5. Then, (3.17) becomes

$$\begin{aligned} P_{processed} &= V_{s,nom} \left[\sum_{j=1}^{H} I_{Bus} + \sum_{j=H+1}^{J} (-I_{Bus} + I_{S,j}) \right] \\ &= V_{s,nom} \left[H \times I_{Bus} - (J - H) \times I_{Bus} + \sum_{j=H+1}^{J} I_{S,j} \right] \\ &= V_{s,nom} \left[(2H - J) \times I_{Bus} + \sum_{j=H+1}^{J} I_{S,j} \right]. \end{aligned} \tag{3.18}$$

Recall that the bus current is the mean of the time average of the server currents, given by (3.9). Since $I_{S,j} = 0$ is valid for all hot-swapped servers, (3.9) can be restated as

$$I_{Bus} = \frac{1}{J} \sum_{j=H+1}^{J} I_{S,j}. \tag{3.19}$$

Using (3.19) in (3.18), the processed power becomes

$$\begin{aligned}
P_{processed} &= V_{s,nom}\left[(2H-J)\times\frac{1}{J}\sum_{j=H+1}^{J} I_{S,j} + \sum_{j=H+1}^{J} I_{S,j}\right] \\
&= V_{s,nom}\left[\left(\frac{(2H-J)}{J}+1\right)\times\sum_{j=H+1}^{J} I_{S,j}\right] \\
&= V_{s,nom}\left[\left(\frac{2H}{J}\right)\times\sum_{j=H+1}^{J} I_{S,j}\right] \\
&= \frac{2H}{J}P_{delivered}. \qquad (3.20)
\end{aligned}$$

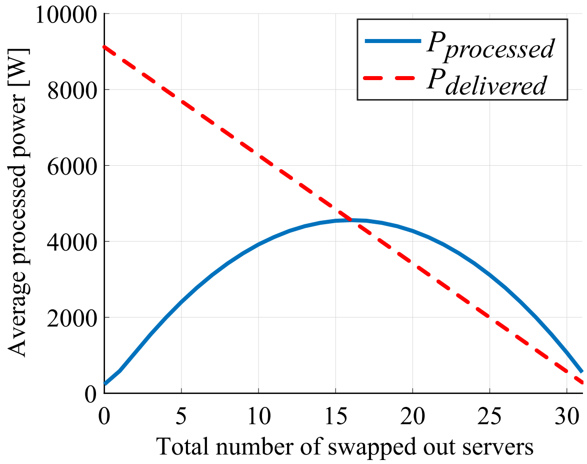
On the other hand, if the bus current is higher than at least one of the remaining server currents during a hot-swap (i.e., $|I_{Bus} - I_{S,j}| > 0$ for at least one $j\epsilon\{H+1, H+2, \ldots, J\}$), further simplification of (3.17) requires more information about the server currents, and therefore does not produce a closed form expression.

Equation (3.20) can be used to relate the processed power to the delivered power in the series-stacked architecture when the bus current is less than all remaining server currents during any hot-swap (i.e., $I_{Bus} < I_{S,j}$ for $j\epsilon\{H+1, H+2, \ldots, J\}$). In such a scenario, it can be observed that due to the $\frac{2H}{J}$ term, processed power is guaranteed to be less than the delivered power in the server-to-virtual bus DPP architecture unless half of the servers are hot-swapped at the same time. (Note that $P_{processed}$ and $P_{delivered}$ in Table 3.1 also follow (3.20).)

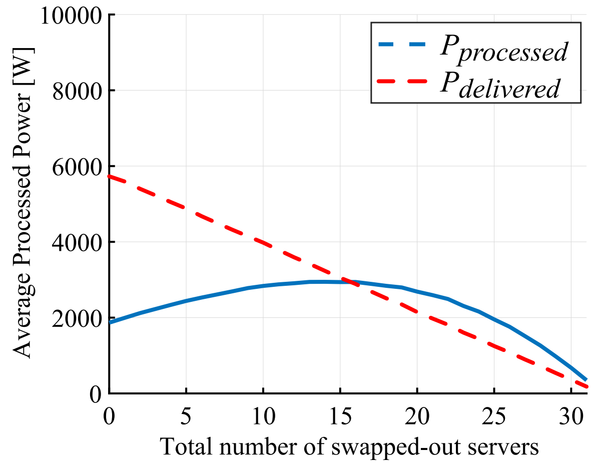
Case Study III

In this case study, hot-swapped operation is applied to the 32-server rack used in *Case Study I.* For 60% and 95% average computational load scenarios in *Case Study I*, (3.13) and (3.14) are plotted versus the total number of swapped-out units in Figures 3.6(a) and 3.6(b), respectively.

As shown in Figures 3.6(a) and 3.6(b), unless half of the servers in a series stack are swapped out at the same time, the processed power in the server-to-virtual bus DPP architecture is less than the delivered power, yielding an efficiency improvement for the series-stacked approach compared to conventional solutions. Although processed power is more than delivered power after more than half of the servers are hot-swapped at the same time, processed power decreases as more servers are hot-swapped since processed power is related to delivered power by (3.20).



(a) 60% average computation load and ± 40% computational load range.



(b) 90% average computation load and ± 5% computational load range.

Figure 3.6: Comparison of total processed and delivered power in the server-to-virtual bus architecture.

3.4 Key implementation challenges

Even though the series-stacked architecture has an inherent advantage in terms of power delivery efficiency, there are some implementation challenges that are unique to data center applications.

3.4.1 Communication across voltage domains

Since there is no common ground between servers in a series-stacked architecture to be used as a reference level for communication purposes, communicating across different voltage domains may be considered an implementation challenge. However, commonly used communication interfaces in data center applications typically involve inherent isolation which is sufficient for the series-stacked architecture. For example, the signal isolation transformers in standard Ethernet are rated for 1.5 kV dc isolation. Also, high-speed fiber optic cables and ac coupling capacitors in serial links can be used to communicate across serially connected servers. Note that such interfaces are already in use in data centers, reducing the additional implementation effort needed to achieve communication in series-stacked architectures [114].

3.4.2 Initialization

In a series-stacked configuration, when the dc bus voltage is applied to the series stack, voltage balance between the series-stacked hardware may be an issue before any voltage control by the differential converters can take place. One solution to this challenge could be connecting shunt resistors in parallel with the series-stacked hardware to ensure proper voltage balancing when there is no voltage regulation. However, the continuous employment of shunt resistors reduces the high efficiency power delivery promise of the series-stacked architecture. Therefore, a circuit is needed to disable the shunt resistors after voltage regulation in the stacked architecture is successfully initialized.

3.4.3 Hot-swapping

Hot-swapping implementation in conventional power delivery architectures is straightforward due to the individual power delivery paths. However, in series-stacked architectures, hot-swapping implementation becomes challenging since series connected servers form the main power delivery path. In order to implement hot-swapping in any series-stacked architecture, the system must be capable of isolating a swapped-out server from the series stack, the main power delivery path must be maintained by using a bypass switch or by sinking bus current through the differential converter, and in-rush current must be limited when a server is swapped in.

Hot-swapping operation first requires complete isolation of a server from the series stack. This is crucial for operator safety, since each server is at a different voltage level in the series stack. Moreover, once a server is isolated from the series stack, the absence of that server in the main current flow path must be detected by the control algorithm and the flow of bus current in the series stack must be maintained by the corresponding differential converter or a bypass switch as mentioned in Section 3.3.2. When a server is plugged in after a hot-swap event, inrush current occurs due to the high server input capacitance [122]. In the series-stacked architecture, inrush current tends to flow through the series connected servers, and if unchecked, this may lead to a voltage imbalance across the series stack, potentially damaging adjacent servers and differential converters. Therefore, limiting in-rush current is an important requirement in a series-stacked architecture.

CHAPTER 4

BIDIRECTIONAL HYSTERESIS CONTROL FOR THE SERVER-TO-VIRTUAL BUS DPP ARCHITECTURE

In this chapter, the details of a control algorithm developed for the server-to-virtual bus DPP architecture are presented. The fundamental challenge in the server-to-virtual bus DPP architecture is voltage regulation of the series-stacked servers and the virtual bus. Since series connected servers need to conduct the same amount of current, any power mismatch between servers results in voltage variation at server input. This variation may easily exceed the allowed input voltage rating and cause damage to the servers. Bidirectional differential converters capable of injecting or rejecting current are used to regulate the series-stacked server voltages. Special attention must be given to the virtual bus voltage while regulating the server voltages. In order for the virtual bus to be an instantaneous energy buffer, its voltage must be regulated within safe limits. Voltage regulation in the server-to-virtual bus DPP architecture thus becomes a challenging control problem. Here, the operation of the server-to-virtual bus DPP architecture is revisited with concentration on control aspects of the system. First, a control method is proposed and explained for steady-state operation at fixed dc bus voltage. Then, it is extended for hot-swapped operation and for varying dc bus operation. Key features of the control are supported with simulations in PLECS [123].

4.1 System analysis for control discussion

The server-to-virtual bus DPP architecture was given in Figure 3.3. It is depicted again in Figure 4.1 to facilitate the discussion.

As can be seen in Figure 4.1, since the inputs and outputs of all differential converters are the same, neither the physical location of a server in the series stack nor the number of series-stacked servers affects the following discussion. Therefore, S_j represents any server in the series stack, ζ represents the set containing all servers in the system (i.e., $S_j \in \zeta$), and J represents the total number of series-stacked servers (i.e., $J = |\zeta|$). Following this notation, i_{Sj} and v_{Sj} represent the jth server current and voltage, respectively, and $i_{\Delta,j}$ represents the differential current for S_j,

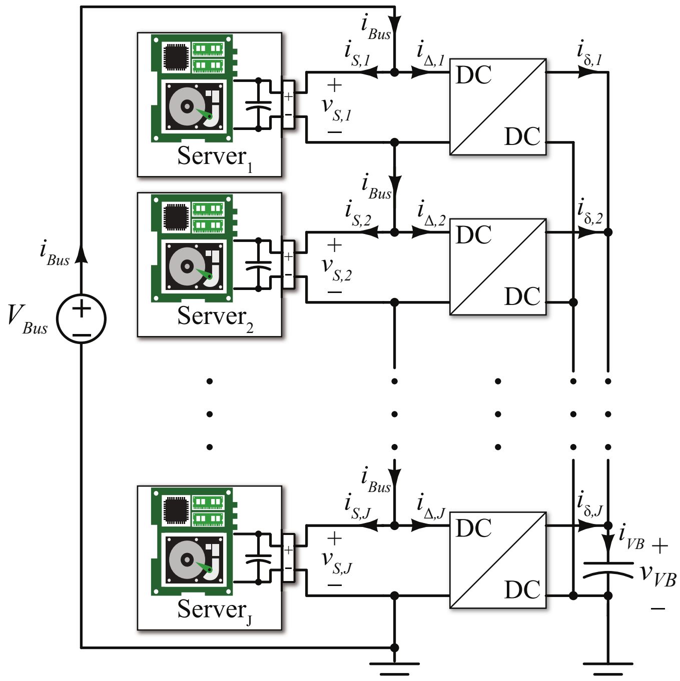


Figure 4.1: Server-to-virtual bus DPP architecture schematic used for analysis.

which is defined as the difference between the server current and the bus current. The outputs of all differential converters are connected at the virtual bus capacitor; therefore, the output voltage of all differential converters is v_{VB}. Also, the dc bus is shown as an ideal voltage source for now since in practice it can be regulated by a rectifier and supplied with capacitor banks in order to provide a bus voltage with a larger time constant than the server dynamics. Therefore, in the following discussion the dc bus has constant voltage (i.e., $v_{Bus} = V_{Bus}$) and it can provide any amount of bus current (i_{Bus}).

In Section 3.3, steady-state operation of the server-to-virtual bus DPP was explained. Here, for the purpose of control discussion, three key circuit constraints that need to be satisfied in the server-to-virtual bus DPP architecture are restated. Since all servers are connected in series, server input voltages must sum up to the fixed bus voltage,

$$V_{Bus} = \sum_{j \epsilon J} v_{s,j}. \tag{4.1}$$

Another constraint of the system is KCL at every intermediate node in the series stack. Since bus current must continuously flow through the series-stack, it must be shared by each server and

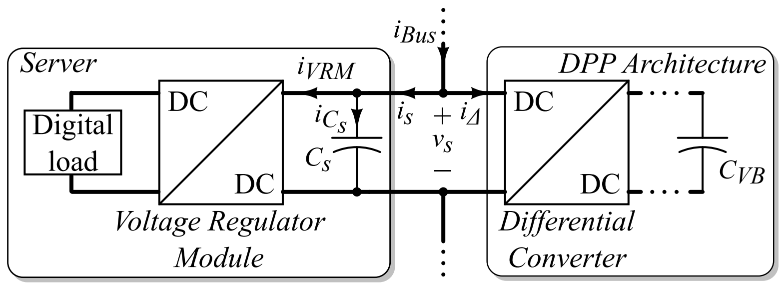


Figure 4.2: Server model in the series-stack.

differential converter pair in the stack,

$$i_{Bus} = i_{S,j} + i_{\Delta,j}. \tag{4.2}$$

Since the virtual bus is connected at the output of every differential converter, the current into the virtual bus is the sum of all differential converter currents (assuming the differential converters are ideal for simplicity),

$$i_{VB} = \sum_{j \epsilon J} i_{\Delta,j}. \tag{4.3}$$

In addition to these three key circuit constraints, a simplified model of a server in the server-to-virtual bus DPP architecture shown in Figure 4.2 introduces another KCL expression. The model in Figure 4.2 consists of one input capacitor ($C_{S,j}$) in parallel with one voltage regulator module (VRM) that provides a lower voltage level for a computing module. Since series-stacked servers share the same bus current (i_{Bus}), the difference between i_{Bus} and $i_{VRM,j}$ is supplied or stored in $C_{S,j}$ absent any differential converter current injection or rejection (i.e., $i_{\Delta,j} = 0$). This difference causes a variation in server voltage,

$$i_{C_S} = i_{Bus} - i_{VRM} = C_S \frac{dv_S}{dt}, \tag{4.4}$$

again for the case where $i_{\Delta,j} = 0$. The virtual bus is also a capacitive buffer for instantaneous power mismatch between the servers and its voltage variation can be captured by

$$i_{VB} = \sum_{j \epsilon J} i_{\Delta,j} = C_{VB} \frac{dv_{VB}}{dt}. \tag{4.5}$$

Depending on the power mismatch between series stacked servers, each server must be an element in one of three subsets as explained below.

Let S_k refer to servers that require higher current than the bus current (i.e., $i_{S,k} = i_{VRM,k} + i_{C_{S,k}} > i_{Bus}$). Before differential current takes effect, this requirement results in a voltage decrease given by (4.4). $INJ \subset \zeta$ is defined as the subset that contains servers which need current injection (i.e., $S_k \epsilon INJ$), and K is the number of servers in this subset (i.e., $K = |INJ|$).

Let S_l refer to servers that require less current than the bus current (i.e., $i_{S,l} = i_{VRM,l} + i_{C_{S,l}} < i_{Bus}$). Before differential current takes effect, this requirement results in a voltage increase given by (4.4). $REJ \subset \zeta$ is defined as the subset that contains servers which need current rejection (i.e., $S_k \epsilon REJ$), and L is the number of servers in this subset (i.e., $L = |REJ|$).

For the sake of completeness, let S_n refer to any remaining servers (i.e., $i_{S,n} = i_{Bus}$). $NONE \subset \zeta$ is defined as the subset that contains servers which do not require any current (i.e., $S_n \epsilon NONE$), and N is the number of servers in this set (i.e., $N = |NONE|$). Following these definitions, one can deduce that $K + L + N = J$, $INJ \cap REJ \cap NONE = 0$, and $INJ \cup REJ \cup NONE = \zeta$.

Note that since current consumption of servers is not constant over time, the elements of INJ, REJ, and $NONE$ may change. However, the voltage constraint of the series connection given by (4.1) enforces the following relations: all servers cannot require current injection or rejection at the same time (as given in (4.6)), if there exists one server that requires current injection, there must be at least one other server that requires current rejection or does not require any action (as given by (4.7)). Likewise, if there exists one server that requires current rejection, there must be at least one other server that requires current injection or does not require any action (as given by (4.8)).

$$K \neq J \quad \& \quad L \neq J \tag{4.6}$$

$$K \geq 1 \rightarrow L + N \geq 1 \tag{4.7}$$

$$L \geq 1 \rightarrow K + N \geq 1 \tag{4.8}$$

4.2 Control objectives

The server-to-virtual bus DPP architecture consists of J servers and J differential converters; however, there are $J+1$ voltages to regulate since the voltage of the virtual bus is free to vary, but must be regulated to within acceptable limits. Each differential converter is thus responsible for regulating its server's voltage while also ensuring there is always energy in the virtual bus capacitor, and that the virtual bus voltage is within limits. In addition, in this work, the differential converters must be rated for full server power in order to accommodate any abnormal operational conditions such as server initialization, shutdown, and hot-swapping. However, during normal operation when servers are (ideally) equally loaded, the mismatch power between servers is much less than rated server power. This requires the differential converters to work at light load with high efficiency. For these reasons, the control algorithm must be able to determine both the direction and the amount of power flow in each differential converter. Moreover, only voltage feedback is desired for control simplicity and to avoid wasting energy at current shunt monitors. Also, distributed digital control eliminates the need for communication between converters, making the server-to-virtual bus architecture easy to implement. In order to achieve these objectives, a distributed bidirectional hysteresis control method that uses only voltage measurements is developed here for the server-to-virtual bus DPP architecture.

4.3 Proposed control

The fundamental approach of the proposed control method is for each local controller to monitor the input and output voltage of its local converter in order to determine whether the converter needs to turn on or off. Note that if the converter needs to turn on, the direction of power flow should be dynamically determined as well. This approach can be grouped into three sequential steps: a voltage sampling step, a current need determination step, and a power flow direction decision step. Each of these steps is discussed below. Following this discussion, an example scenario is explained in detail, emphasizing both the distributed and bidirectional nature of the proposed control method.

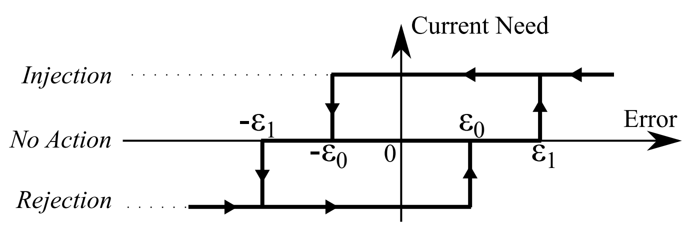


Figure 4.3: Proposed bidirectional hysteresis action.

4.3.1 Voltage sampling

Each differential converter samples both its server voltage and the virtual bus voltage, and the voltage variation of each voltage domain is given by

$$\varepsilon_{S,j} = V_{nom} - v_{S,j} \quad \text{and} \quad \varepsilon_{VB} = V_{nom} - v_{VB} \tag{4.9}$$

for each converter, where V_{nom} is the nominal server input voltage.

4.3.2 Current need

The current needs of both the server and virtual bus are determined by referring to the proposed bidirectional hysteresis shape in Figure 4.3, which is valid for both server and virtual bus voltages independently. The hysteresis shape in Figure 4.3 has three "current need states" on the y-axis (current injection, current rejection and no action) and four predefined error values on the x-axis ($\pm\varepsilon_0$ and $\pm\varepsilon_1$). Depending on the calculated voltage error and the current need state of the voltage domain during the previous sampling time, the state for the present sampling time is determined. As a result of this step, each server becomes an element in one of the subsets INJ, REJ, or $NONE$. Note that Figure 4.3 is demonstrated in general, but applies to both the converter input and output separately.

4.3.3 Power flow direction

For each differential converter the current need determination step has nine possible outcomes since each differential converter is responsible for two voltage domains (i.e., the server and the virtual bus) and each voltage domain has three possible current need states (i.e., current injection, current rejection and no action). These nine possible outcomes and the corresponding decisions are tabulated in Table 4.1, where $+I_{\Delta,j}$ corresponds to average current being injected to the server ($+ < i_{\Delta,j} >$) at that sampling time, $-I_{\Delta,j}$ corresponds to the current being removed from the server ($- < i_{\Delta,j} >$) at that sampling time (i.e., injected to the virtual bus), and OFF corresponds the converter being kept off at that sampling time.

Although Table 4.1 is unique and generated for each differential converter separately at every sampling time, the current need of the virtual bus is common to all differential converters. Therefore, the information in the columns of Table 4.1 is the same for and naturally shared among all

Table 4.1: Decision table for bi-directional hysteresis control

		Virtual Bus No Action	Virtual Bus Injection	Virtual Bus Rejection
Server	No Action	OFF	$+I_{\Delta,j}$	$-I_{\Delta,j}$
Server	Injection	$-I_{\Delta,j}$	OFF	$-I_{\Delta,j}$
Server	Rejection	$+I_{\Delta,j}$	$+I_{\Delta,j}$	OFF

differential converters. However, the current need of each server depends on the instantaneous power mismatch between servers. The information in the rows of Table 4.1 is thus different for each differential converter.

In Table 4.1, for outcomes that are located in off-diagonal cells, neither the server nor the virtual bus requires current injection (or rejection) at the same time. For these off-diagonal outcomes, current injection (or rejection) decisions made to satisfy the current need of one voltage domain have no adverse effect on the other voltage domain. In addition, keeping the differential converter off when the server and virtual bus do not require any current injection or rejection follows from the hysteresis control algorithm. On the other hand, for remaining off-diagonal outcomes in Table 4.1, both the server and the virtual bus require current injection (or rejection) at the same time. For these diagonal outcomes, the control decision relies on the series-stacked system properties analyzed in Section. 4.1 and keeps the converter off, as can be seen in Table 4.1. This decision is explained through the example scenario below.

Consider one of the differential converters in the stack and let a digital controller sample both its server voltage ($v_{S,j}$) and the virtual bus voltage (v_{VB}). The current need state of each voltage domain for this sampling time is determined as explained in Section 4.3.2 for both voltage domains. For example, consider the outcome that server S_j and the virtual bus simultaneously require current injection. This occurs when both $v_{S,j}$ and v_{VB} are lower than their nominal values. It results in S_j being $S_k \epsilon INJ$ and $K \geq 1$. Recall that the sum of all series-stacked server voltages is enforced to be fixed by (4.1). This implies that there must be at least one other server in the series stack that is $S_l \epsilon REJ$ and/or $S_n \epsilon NONE$, as in (4.7). As in Table 4.1, the decision made to satisfy S_l and/or the decision that has no adverse effect on S_n is to inject current into the virtual bus to satisfy the requirements of the virtual bus. Also, since (3.2) needs to be valid at every intermediate node in the series stack, differential current rejection from S_l and/or S_n increases the bus current by exactly the same amount that S_k would get if the virtual bus did not require current injection.

This increase in i_{Bus} satisfies the current injection requirement of S_k and increases its voltage.

The same idea is valid when both a server and the virtual bus simultaneously require current rejection. Therefore, when both a server and the virtual bus simultaneously have the same demands, the appropriate decision is to keep the converter off as shown in Table 4.1.

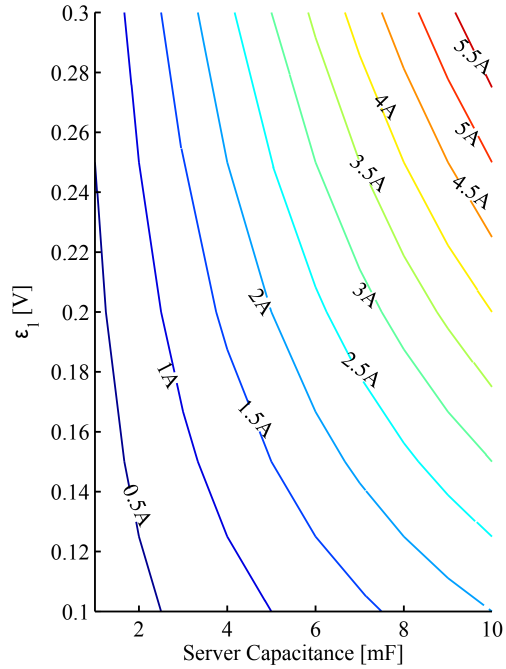
4.4 C_s and C_{VB} sizing considerations

Important design considerations for light load operation of the proposed control algorithm are the size of C_s and C_{VB}, and the hysteresis bands for both the server ($\varepsilon_{S,j,1}$) and the virtual bus ($\varepsilon_{VB,1}$), since these parameters affect how often the differential converters need to process power. Also, the magnitude of the differential current (i_Δ) affects the amount of processed power. These directly affect the system-level efficiency since power loss in the system only occurs when differential converters are operational. Keeping the differential converters off as long as possible is thus an important way to maximize system-level efficiency. A four-server rack system is planned for the experimental study in this work; the discussion thus continues for a four-server rack.

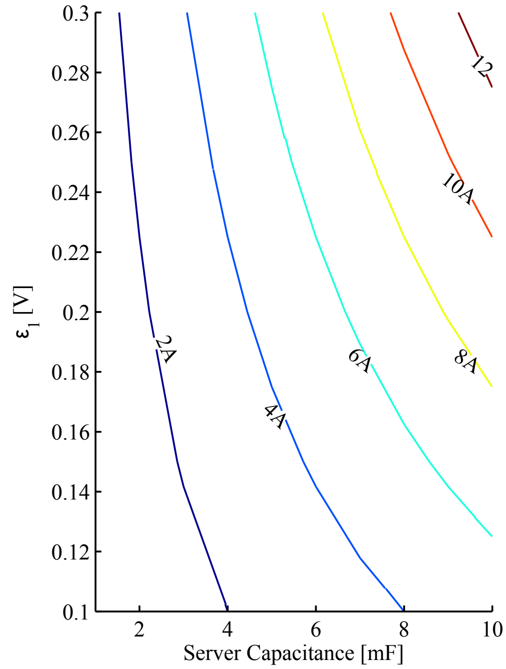
For design intuition about C_s, C_{VB}, $\varepsilon_{S,j,1}$, $\varepsilon_{VB,1}$ and i_Δ values, contour plots in Figure 4.4 are plotted using (4.4) and (4.5) for certain operating conditions.

When the differential converter is kept off, contour lines in Figure 4.4(a) show the allowed difference between i_{Bus} and i_{VRM} for various values of C_S and $\varepsilon_{S,j,1}$. Once the difference between i_{Bus} and i_{VRM} exceeds its allowed value for a chosen pair of C_S and $\varepsilon_{S,j,1}$, the server voltage varies higher than $\varepsilon_{S,j,1}$. The differential converter then turns on and provides current to the system. Contour lines in Figure 4.4(b) show the required magnitude of i_Δ for the same values of C_S and $\varepsilon_{S,j,1}$ as in Figure 4.4(a). For a given allowed difference between i_{Bus} and i_{VRM}, a feasible pair of C_S and $\varepsilon_{S,j,1}$ can be selected by referring to Figure 4.4(a). The minimum differential current magnitude to regulate the server voltage back within the hysteresis limits is determined by referring to Figure 4.4(b).

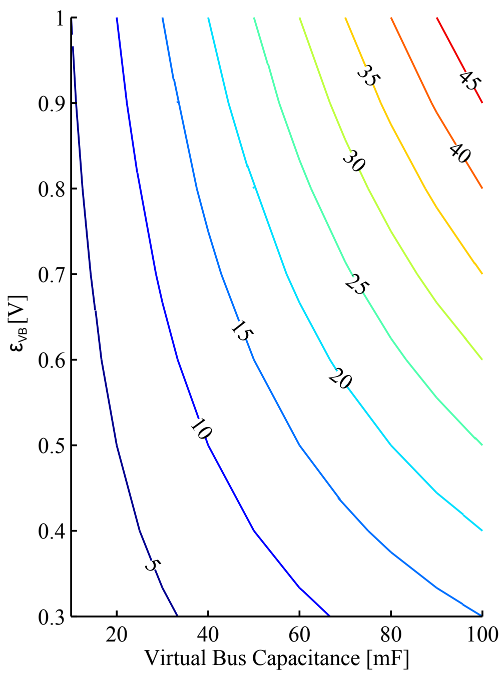
Since voltage regulation requirements on the virtual bus capacitor are more relaxed in terms of both magnitude and duration, sizing of the virtual bus capacitor is considered for a worst case scenario. In the four-server rack considered in this work, the worst case occurs when $K = 3$ and $L = 1$ (or when $K = 1$ and $L = 3$). In this scenario, the net virtual bus current is twice the differential current. Contour lines in Figure 4.4(c) show the allowed consecutive sampling times under the worst case scenario before the virtual bus requires regulation action for different values



(a) Allowed voltage variation of server voltage versus C_S for the difference between i_{Bus} and i_{VRM}.



(b) Minimum amount of i_Δ in order to regulate the server voltage.



(c) Allowed consecutive sampling times before the differential converters start regulating the virtual bus.

Figure 4.4: Charts for C_s, C_{VB}, ε_S, ε_{VB} and i_Δ decisions.

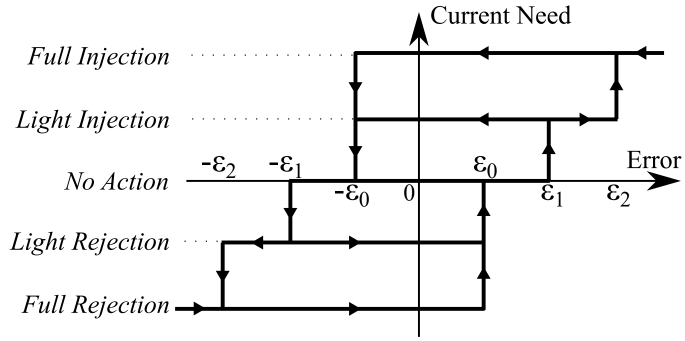


Figure 4.5: Extended bi-directional hysteresis shape. Note that the bidirectional hysteresis shape is generic and independently valid for each series-stacked server and the virtual bus.

of C_{VB} and ε_{VB}.

Considering the above method $C_s = 5$ mF, $\varepsilon_{S,j,1} = 0.15$ V, $i_\Delta = 3$ A, $C_{VB} = 50$ mF, and $\varepsilon_{VB} = 0.5$ V are chosen as initial control parameters for simulation study. The sampling time is chosen to be 50 μs, corresponding to 20 kHz control bandwidth in simulation.

4.5 Extension of bidirectional hysteresis control

Before proceeding to a simulation study, the operation of the bidirectional hysteresis algorithm must be extended to enable its usage for hot-swapping and varying dc bus operation.

4.5.1 Hot-swapping

Here, light load operation of the bidirectional hysteresis algorithm is extended to enable its usage for abnormal operation where more current injection or rejection may be necessary such as during server initialization and hot-swapping.

Shown in Figure 4.5 is the extended bidirectional hysteresis shape having two more "current need states" added on the y-axis (heavy current injection and heavy current rejection) and two more predefined error values added on the x-axis ($\pm\varepsilon_2$).[1] The five different current need states in Figure 4.5 result in 25 possible outcomes for each differential converter, which are tabulated in Table 4.2. In addition to the notation used in Table 4.1, the superscript * in Table 4.2 means that the differential converter operates at a higher current level. For example, in Figure 4.5 when the voltage error of any voltage domain is less than $\pm\varepsilon_2$, current injection or rejection in light-load mode

[1] More quantized states and predefined error values are possible as mentioned in [7], and might provide better regulation at the expense of a more complicated decision table.

Table 4.2: Decision table for extended bidirectional hysteresis control

		Virtual Bus				
		No Action	Light Injection	Light Rejection	Full Injection	Full Rejection
Server	No Action	OFF	$+I_{\Delta,j}$	$-I_{\Delta,j}$	$+I^*_{\Delta,j}$	$-I^*_{\Delta,j}$
	Light Injection	$-I_{\Delta,j}$	OFF	$-I_{\Delta,j}$	$+I_{\Delta,j}$	$-I_{\Delta,j}$
	Light Rejection	$+I_{\Delta,j}$	$+I_{\Delta,j}$	OFF	$+I_{\Delta,j}$	$-I_{\Delta,j}$
	Full Injection	$-I^*_{\Delta,j}$	$-I_{\Delta,j}$	$-I^*_{\Delta,j}$	OFF	$-I^*_{\Delta,j}$
	Full Rejection	$+I^*_{\Delta,j}$	$+I^*_{\Delta,j}$	$+I_{\Delta,j}$	$+I^*_{\Delta,j}$	OFF

is sufficient to regulate the voltage domain within $\pm\varepsilon_0$. However, if the voltage error of any voltage domain is more than $\pm\varepsilon_2$, the differential converter needs to operate in a higher current mode until the voltage domain is regulated within $\pm\varepsilon_0$. In the extended bidirectional hysteresis control, differential converters are kept off to maximize power delivery efficiency when their corresponding voltage domains are within $\pm\varepsilon_1$. Further explanation of both Figure 4.5 and Table 4.2 follows the same principles as explained in Section 4.3 and are not repeated here.

4.5.2 Varying dc bus voltage

So far, the proposed control algorithm assumes a fixed dc bus voltage, which may not be compatible with battery backup systems where the dc bus is supplied through a UPS, as shown in Figure 4.6. Here, the proposed control algorithm is extended to maintain continuous operation of series-stacked servers when the dc bus voltage is not constant.

Recall (3.1), the KVL expression around the series connection, which is valid under all circumstances:

$$v_{bus} = \sum_{j=1}^{J} v_{S,j}. \tag{4.10}$$

Here, v_{bus} and $v_{server,j}$ refer to instantaneous voltages of the dc bus and jth series-stacked server, respectively. Since the virtual bus voltage is a free design parameter in the system, it can be enforced to be the same as the average of the instantaneous server voltages (i.e., v_{bus}/J). These two properties of the server-to-virtual bus DPP architecture enable all $J+1$ voltages (J servers and one virtual bus voltage) to be coupled through the series connection.

Assume $V_{bus,min}$ and $V_{bus,max}$ are the minimum and maximum expected voltages at the dc bus in a server rack under unregulated bus conditions (i.e., $V_{bus,min} < v_{bus} < V_{bus,max}$). For example, for a server rack that employs an UPS at its dc bus, $V_{bus,min}$ and $V_{bus,max}$ refer to the UPS output voltage range. Further, assume that POL converters inside a server motherboard can perform output voltage regulation for server input voltage values between $V_{bus,min}/J$ and $V_{bus,max}/J$. The server-to-virtual bus DPP architecture should regulate all $J+1$ voltages to v_{bus}/J in order to ensure operation of the series-stacked servers while maintaining the virtual bus at a safe energy exchange value.

The proposed extension to the bidirectional hysteresis control algorithm can be summarized as follows: At every sampling time, the reference voltage (v_{ref}) for the series-stacked servers and the virtual bus is determined by measuring the dc bus voltage and dividing it by the number of series-stacked servers (i.e., $v_{ref} = v_{Bus}/J$). The server voltage and the virtual bus voltage are measured locally by each differential converter. Each differential converter compares its server voltage and the virtual bus voltage to the reference voltage to calculate errors given by $v_{error,Server} = v_{ref} - v_{Server,j}$ and $v_{error,VB} = v_{ref} - v_{VB}$. Depending on the sign and magnitude of the errors, the current need of the server or virtual bus can be determined as explained in Section 4.3. Note that the dc bus voltage must be known by each differential converter controller at every sampling time for this extended bidirectional hysteresis control to function in a distributed manner.

Before proceeding to a simulation study of the proposed bidirectional hysteresis control for the server-to-virtual bus DPP architecture, one last thing to be noted is that regulating all series-stacked server voltages and the virtual bus voltage to v_{bus}/J can also be achieved locally by differential converters using a proportional controller. (An implementation of such a control idea for photovoltaic systems can be found in [85].) Each differential converter can sample its input (i.e., server) and output (i.e., virtual bus) voltage, and feed back the error between its input and output voltage to a local proportional controller in order to determine the amount and direction of current flow.

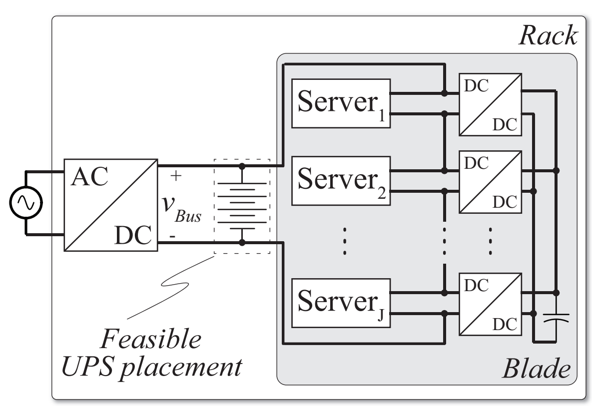


Figure 4.6: A circuit diagram depicting feasible UPS placement in the series-stacked architecture.

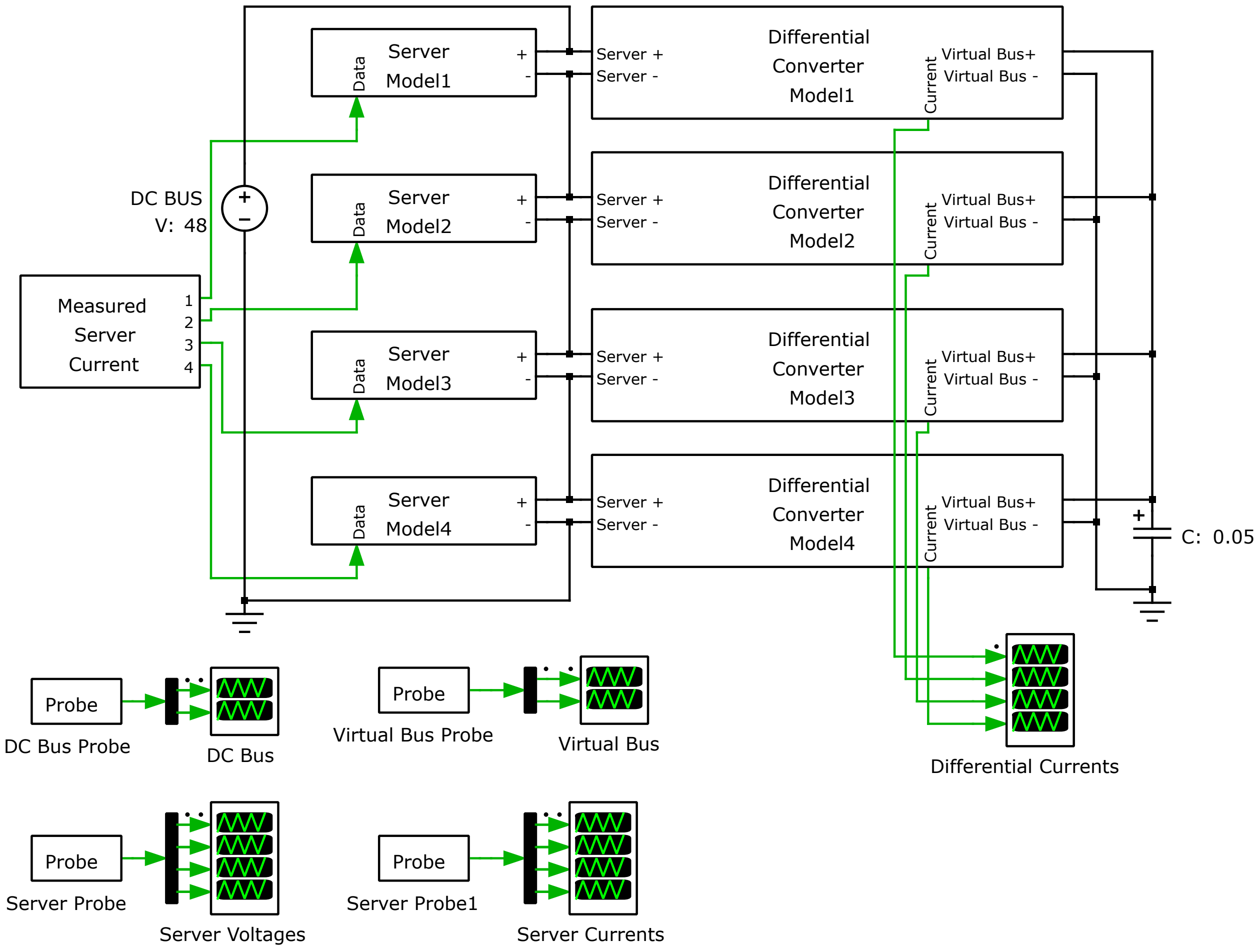
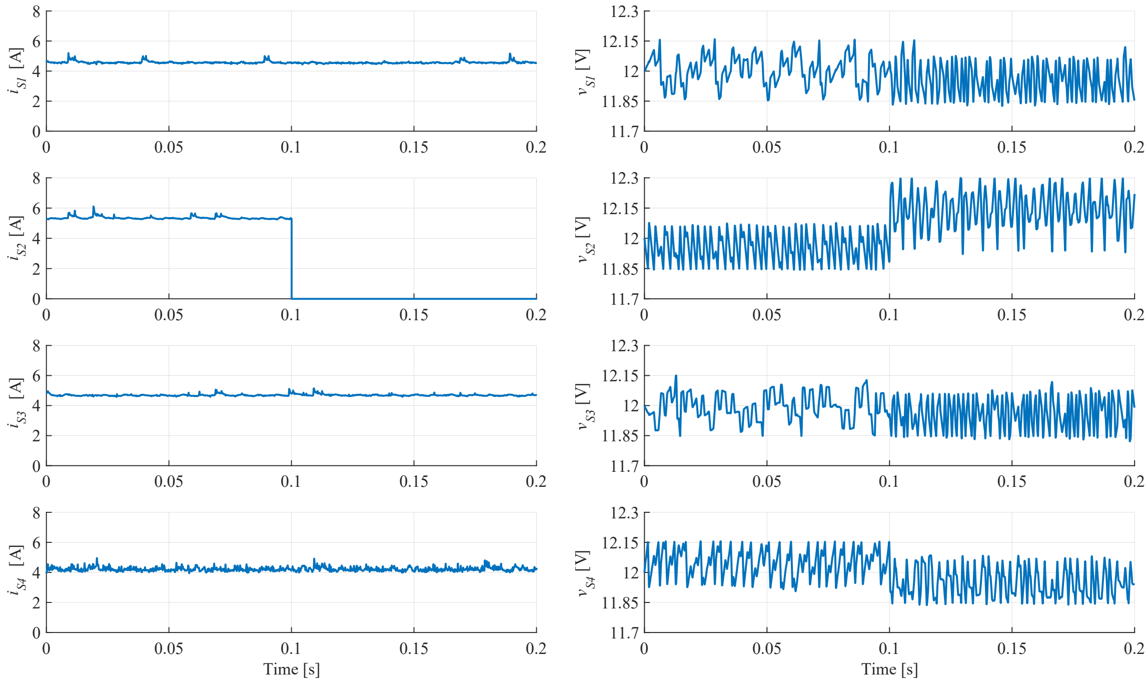


Figure 4.7: Simulation schematic model in PLECS.

As the local proportional controller tries to reduce the error between the input and output voltage of its differential converter, the series-stacked server voltages and virtual bus voltage converge to v_{bus}/J. Although the local proportional controller idea can achieve the desired control objective, inefficient light-load operation of the differential converters may reduce overall power conversion efficiency; thus, it is not explored in this dissertation.

4.6 Simulation Study

The distributed bidirectional hysteresis algorithm is validated in a simulation environment with the simulation model shown in Figure 4.7. Both normal (i.e., when servers are consuming similar currents) and hot-swapped operation are simulated in order to show the operation of the proposed control algorithm. Varying dc bus operation of the server-to-virtual bus DPP architecture



(a) Server currents. (b) Server voltages.

Figure 4.8: Simulated server waveforms.

is validated in the experimental study in Section 5.4.2.

The server models in Figure 4.7 employ an input capacitor in parallel with a controlled current sink which takes its values from measured server currents. A mathematical model for 95% efficient bidirectional converters was built to count for potential power loss in the differential converters. The proposed control algorithm is coded in a C-Script block of the simulator. Simulated server waveforms, differential currents, and the virtual bus voltage for normal operation are given in Figures 4.8 through 4.10.

There is no severe mismatch between server currents, given in Figure 4.8(a), during the first half of the simulation. This represents normal operation in which all series-stacked servers are similarly loaded. At t=0.1 seconds, the current of the second server is pulled down to zero to simulate a hot-swapping scenario. The simulated server and virtual bus voltages are plotted in Figure 4.8(b) and 4.10, showing voltage regulation during both steady-state operation and hot-swapping operation. The differential currents given in Figure 4.9 show both the bidirectional and hysteresis nature of the control. Note that after the second server current pulls down to zero, the second differential converter starts to process higher power in order to guarantee continuous operation of the remaining servers.

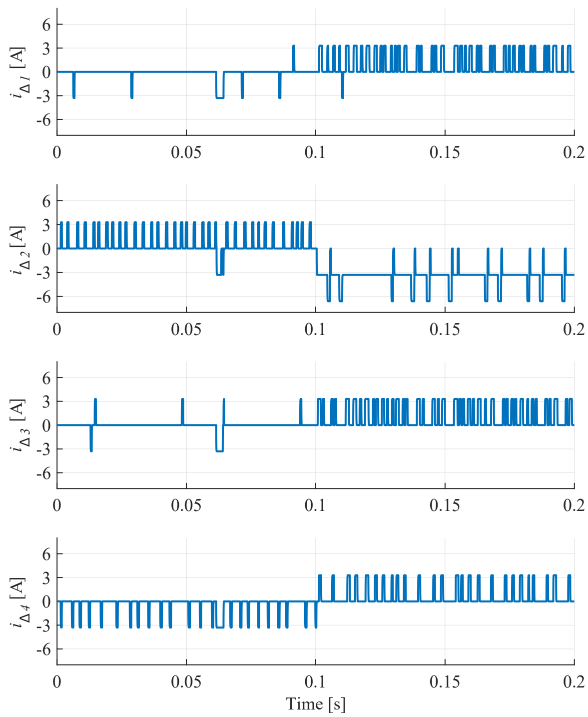


Figure 4.9: Simulated differential currents.

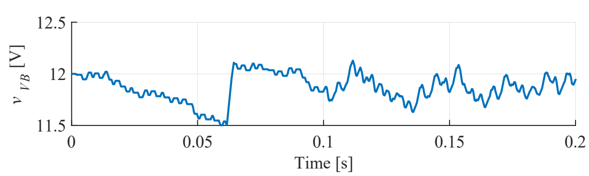


Figure 4.10: Simulated virtual bus voltage.

CHAPTER 5

EXPERIMENTAL STUDY OF THE SERVER-TO-VIRTUAL BUS DPP ARCHITECTURE

In this chapter, implementation details of prototype hardware and an experimental testbed for the server-to-virtual bus DPP architecture are described, and the experimental results are reported.

5.1 Prototype DPP hardware

Prototype DPP hardware for the server-to-virtual bus architecture is depicted in detail in Figure 5.1. This hardware has three terminals: a server terminal (*Server+* and *Server-* in Figure 5.1), a virtual bus terminal (*Virtual Bus+* and *Virtual Bus-* in Figure 5.1), and a series-stack terminal (*Series-Stack+* and *Series-Stack-* in Figure 5.1) for interconnecting to the series stack. These three terminals separate the hardware into two stages: the interface stage and the differential converter stage. The interface stage between the server and the series-stack terminals holds stack initialization circuitry and also is responsible for hot-swapping operation. The differential converter stage is placed between the series stack and the virtual bus terminals. It is responsible for server and virtual bus voltage regulation.

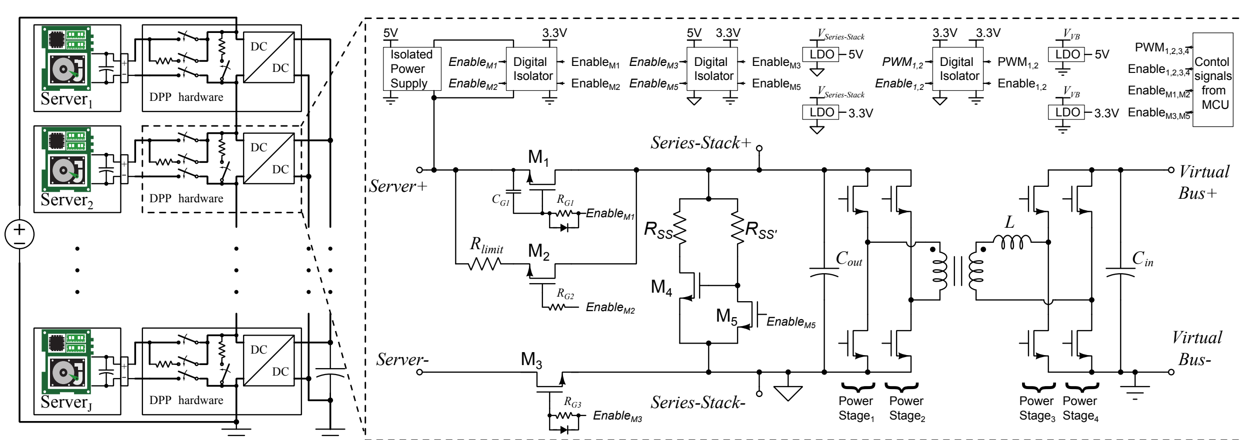


Figure 5.1: Schematic of prototype DPP hardware.

Table 5.1: Key components of the differential converter

Power Stage	TI CSD95372BQ5MC
Digital Isolators	TI ISO724x series
LDOs	TI LP298x series
Transformer	Coilcraft SMT PL160 $\times$ 2
Inductor	Coilcraft SLC1480
C_{in} and C_{out}, ceramic	TDK 10μF 16V X5R $\times$ 6
C_{in} and C_{out}, aluminum	Panasonic 1mF 16V SMD $\times$ 2

5.1.1 Differential converter

The differential converter stage depicted in Figure 5.1 is a dual active bridge (DAB) dc-dc converter which offers bidirectional power flow with symmetric design at both sides of the transformer when the input (server) and output (virtual bus) voltages are nominally the same [124–128]. The DAB converter is implemented with off-the-shelf discrete components that are common in many server power supply designs such as a power stage that employs a high-side and a low-side MOSFET with integrated gate driver circuitry for each half bridge in the converter. Digital isolators are used as level shifters in order to transfer necessary digital signals to the different levels in the series stack. The switching frequency is 200 kHz. The power flow direction and output power in the DAB converter in this work are determined by a simple phase-shift modulation technique [126]. The key components of the differential converter are listed in Table 5.1.

Hot-swapping as pursued in this dissertation requires the differential converters and the virtual bus to maintain the bus current. As explained in Section 3.3.2, hot-swapping operation increases processed power in the system. When not in hot-swapping, the differential converters only process the difference in power, and thus, do not need to be rated for full server power. However, during hot-swapping, the differential converters must be able to handle maximum bus current at nominal server voltage. For example, recall *Case Study II* where six 300 W rated 12 V servers were handling 95% average computational load with ±5% computational load range. Under this load distribution, Figure 3.5(a) showed a server being hot-swapped, causing 237 W processed power in its differential converter. Although this is almost 80% of the rated power, note that in *Case Study II*, the differential converters are assumed to be ideal to simplify the analysis, and average power consumption is used to calculate the processed power. In this work, experimental hot-swapping is demonstrated under a similar load distribution; however, possible power loss in differential converters and instantaneous power demand during hot-swapping and initialization transients are considered when

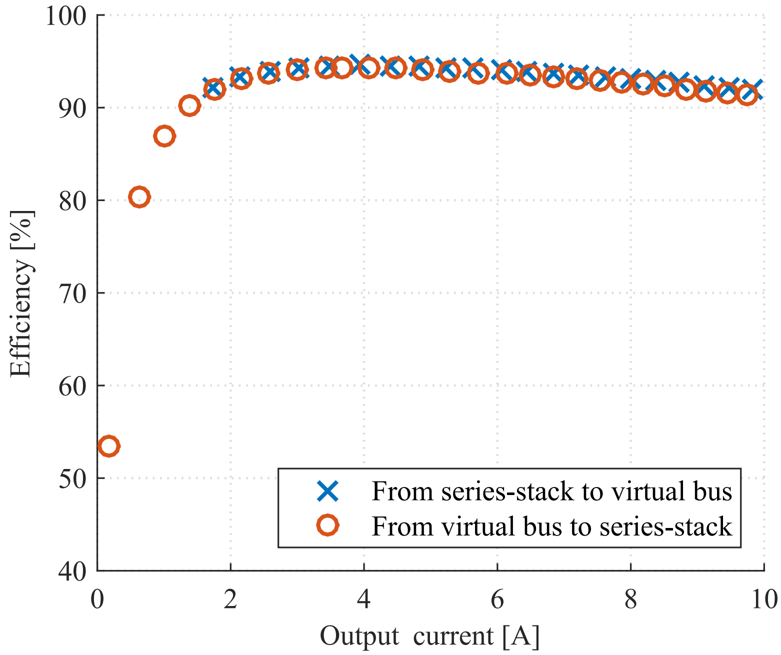


Figure 5.2: Efficiency of the power stage in the hardware prototype

rating the differential converters. Each differential converter is designed to be able to sink or source rated server power (i.e., 120 W at 12 V) from the series-stack terminal, depending on the power flow direction. It is acknowledged that, given an expected load distribution in the series-stacked servers and careful modeling of converter losses, differential converter rating can be optimized and presumably reduced. On the other hand, hot-swapping could have been achieved by using a bypass switch, as explained in Section 3.3.2, which would require bus voltage to be temporarily decreased. If a bypass switch is used to maintain bus current during hot-swapping, the differential converter rating can be further optimized and reduced.

The measured efficiency of the DAB converter prototype is plotted in Figure 5.2 for both power flow directions. As shown in Figure 5.2, due to the symmetric design of the converter, almost identical efficiency curves for both power flow directions are achieved with a peak at 95% around 40 W. An annotated photograph of the prototype hardware is given in Figure 5.3. The printed circuit board (PCB) layouts of the prototype DPP hardware can be found in Appendix A.

5.1.2 Stack initialization circuitry

The *Series-Stack* terminals of the hardware prototype (shown as *Series-Stack+* and *Series-Stack-* in Figure 5.1) facilitate connecting multiple hardware prototypes to each other in order to build the

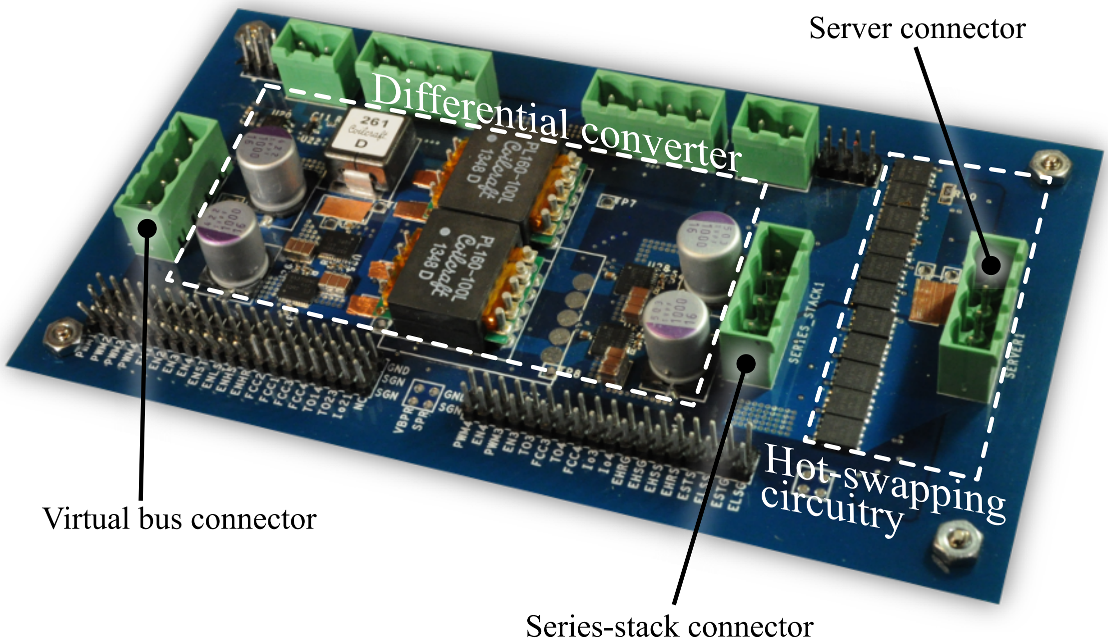


Figure 5.3: Annotated photograph of the prototype DPP hardware.

stacked architecture. The dc bus is connected to the stacked architecture at the *Series-Stack+* terminal of the top board and *Series-Stack-* terminal of the bottom board. In such a series-connected configuration, when dc bus voltage is applied to the series stack, voltage balance between the series-stacked boards can be preserved with shunt resistors between the *Series-Stack* terminals of each board. Continuous employment of shunt resistors reduces the high power conversion efficiency of the series-stacked architecture; therefore, shunt resistors should be disabled after stacked architecture is successfully initialized.

Shown in Figure 5.1, between the *Series-Stack+* and *Series-Stack-* terminals of the hardware prototype, is the proposed stack initialization circuitry, which consists of a shunt resistor (R_{SS}) and auxiliary components ($R^{'}_{SS}$, M_4 and M_5). As the bus voltage is applied to the stacked architecture, M_4 naturally turns on since its gate is pulled high through $R^{'}_{SS}$ (provided that the gate signal of M_5 is kept low), connecting R_{SS} between the *Series-Stack* terminals. This ensures that the dc bus voltage is equally divided between *Series-Stack* terminals of the stacked boards. As the dc bus voltage ramps up to its nominal value, the linear regulators in Figure 5.1 start to provide logic voltages to both the hot-swapping circuitry and the differential converter. After closed-loop operation of the converters is activated, M_4 can be turned off by turning on M_5, and R_{SS} is disconnected from the series stack. Key components of the stack initialization circuitry are listed

Table 5.2: Key components of the stack initialization circuitry

Switches	TI CSD85301Q2
R_{SS}	120 Ω
$R^{'}_{SS}$	100 kΩ

in Table 5.2.

5.1.3 Hot-swapping circuitry

As mentioned before, once a server is hot-swapped in a series-stacked architecture, the main current flow path can be ensured by the corresponding differential converter or a bypass switch. In this dissertation, hot-swapping is achieved using differential converters instead of a bypass switch. A hot-swapping circuitry is designed to achieve this operation.

Hot-swapping circuitry for the series-stacked architecture should provide complete isolation when a server is swapped out, and also should limit the in-rush current due to the large input capacitor of the server during swapin. In a hot-swapping event, complete isolation between the hot-swapped server and the series-stack is achieved by turning off M_1, M_2, and M_3 in Figure 5.1. While M_1, M_2, and M_3 comprise transistors in this implementation, galvanically isolated switches such as relays may be employed, depending on the safety and regulatory requirements. At the end of the hot-swapping event, M_2 and M_3 in Figure 5.1 are turned on, enabling a resistive path to limit in-rush current to the server. Once the input capacitance of the hot-swapped server is slowly charged to the voltage at the *Series-Stack* terminals (v_{Stack}), M_1 is enabled for a low-resistance path between the server and the series stack to supply energy efficiently to the server and resume normal operation. M_2 is turned off a few seconds after M_1 is turned on.

The turn-on transients of M_2 and M_3 during swapin are important to consider for reliable operation. Although R_{limit} is employed to limit the in-rush current into the server, fast turn-on transients of M_2 and M_3 can still interfere with the DPP control algorithm. Therefore, the gate resistances of M_2 and M_3 (i.e., R_{G2} and R_{G3} in Figure 5.1) are set to 1.5 kΩ to increase the RC turn-on time constant of M_2 and M_3. As M_2 and M_3 turn on slowly, the current flow through the R_{limit} forces M_2 to operate in its linear region until the input capacitor of the server is charged since the enable signal of M_2 ($Enable_{M2}$ in Figure 5.1) is referenced to *Server+* terminal. Note that the input capacitor of the server cannot be charged exactly to the stack voltage (v_{Stack}) due to

Table 5.3: The key components of the hot-swapping circuitry

Hot-Swapping Switches	TI CSD18540Q5
Digital Isolator	TI ISO724x series
Isolated Power Supply	TI DCP010505BP
LDO	TI LP2985
R_{limit}	2.8 Ω

the impedance network formed by the server and R_{limit} when M_2 and M_3 are on. Before the server is initialized, the impedance at the input terminals of the server can be modeled as a high resistor (due to a non-operational voltage regulator module) in parallel with the input capacitor. When charging of the input capacitor is completed, v_{Stack} is actually divided between R_{limit} and the high resistance in parallel with the input capacitor. Although R_{limit} is small, the voltage drop across it creates a small current spike as M_1 turns on, which affects the other series-stacked voltages if not managed. The turn-on transient of M_1 is thus decelerated by increasing gate resistance R_{G1} as well. An additional discrete capacitor (C_{G1}=1 μF in Figure 5.1) is added to the gate of M_1 to further decelerate the turn-on transient of M_1 without increasing R_{G1} beyond 100 kΩ. Excessive gate resistance can cause the transistor to operate in the linear region, owing to the inherent gate leakage of the device. The turn-off transient of M_1 and M_3, on the other hand, must be as rapid as possible to quickly isolate a malfunctioning server from the series stack during a swapout event. Ultrafast Schottky diodes are added to the gate driving circuit of M_1 and M_3 in order to bypass R_{G1} and R_{G3} while their gate capacitors are discharged during a swapout event.

The goal of this work was to design hot-swapping circuitry that did not reduce the efficiency noticeably. The TI CSD16570Q5B was found to have the lowest on resistance, 0.59 mΩ with a 5 V gate signal. In order to achieve lower on resistance, 5 MOSFETs are paralleled in the hot-swapping circuit. The additional components of the hot-swapping circuitry are listed in Table 5.3. Note that the hot-swapping switches used in this implementation are rated at 60 V since the dc bus voltage in the experimental testbed is 48 V.

5.2 Experimental setup

In order to validate the operation of the server-to-virtual bus DPP architecture, an experimental setup that consists of a four-server testbed, a controller, and measurement units is built in addition to the four prototype DPP hardware. An annotated schematic of the experimental setup is given in Figure 5.4. This section summarizes details of the experimental setup.

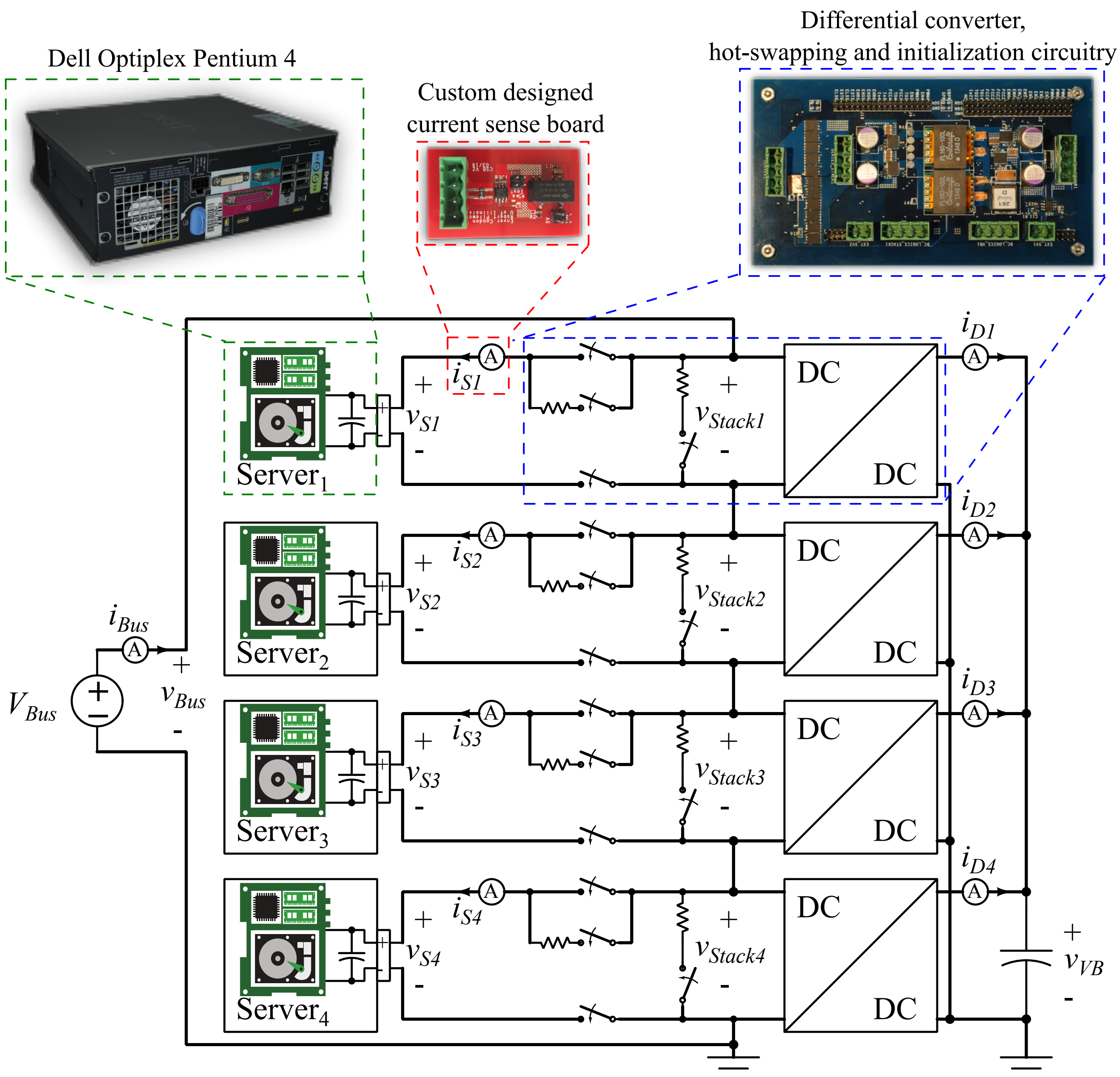


Figure 5.4: Annotated schematic of the experimental setup.

5.2.1 Testbed

A flexible and modular laboratory testbed was developed for experimental validation of the server-to-virtual bus DPP architecture. The servers are Dell Optiplex SX280 workstations with a Pentium 4 CPU, a 2.5" magnetic hard drive disk, and DDR2 memory. These workstations have a single 12 V motherboard input; however, the operating input voltage range is empirically found to be 10.5-13 V.

Each workstation is rated for 120 W peak power. Although these workstations consume lower power than typical servers in data centers, their terminal characteristics in response to computational loads are similar to high-end servers. The workstations used in this experimental work serve as scaled down versions of costly servers and enable validation of series-stacked power delivery at reasonable cost. All workstations run the Linux Ubuntu 14.04 operating system.

The testbed employed a power supply (HP 6674A) that has 0-60 V programmable voltage output to feed the 48 V dc bus. This power supply was used to model both the output of an ac-dc rectification stage and a 48 V UPS at the dc bus. The testbed was grounded to earth at the negative terminal of V_{Bus} and v_{VB}; therefore in this work all voltages are positive with respect to earth. As noted in Section 2.6, various other grounding options may exist in data centers. While the proposed architecture is independent of ground location, care must be taken in safety isolation of floating servers, as well as in implementation of hot-swapping circuitry with respect to polarity and blocking capabilities of switches. Although not used in the test-bed built in this dissertation, diode ORing devices can be employed to enforce current flow in certain directions.

A 32 mF discrete capacitor was used as the virtual bus capacitor. Although in this setup one capacitor was used, the virtual bus capacitor could have been distributed among differential converters, corresponding to 8 mF per converter.

Doubled 16 AWG copper wire is used for all interconnections in the testbed. An annotated photograph of the testbed is given in Figure 5.5.

5.2.2 Controller

A single off-board microcontroller (TI C2000 Piccolo F28069) samples all series-stack voltages, runs the control algorithm explained in Chapter 4 to generate PWM signals for the four differential converters, and manages enable/disable signals for all interface boards. The hysteresis control decision, as explained in Chapter 4, is executed at 2 kHz, corresponding to 500 μs. This is decided empirically considering that differential converters have turn-on and turn-off transients, and must provide requisite charge to increase or decrease their terminal voltages.

The microcontroller code is given in Appendix B.

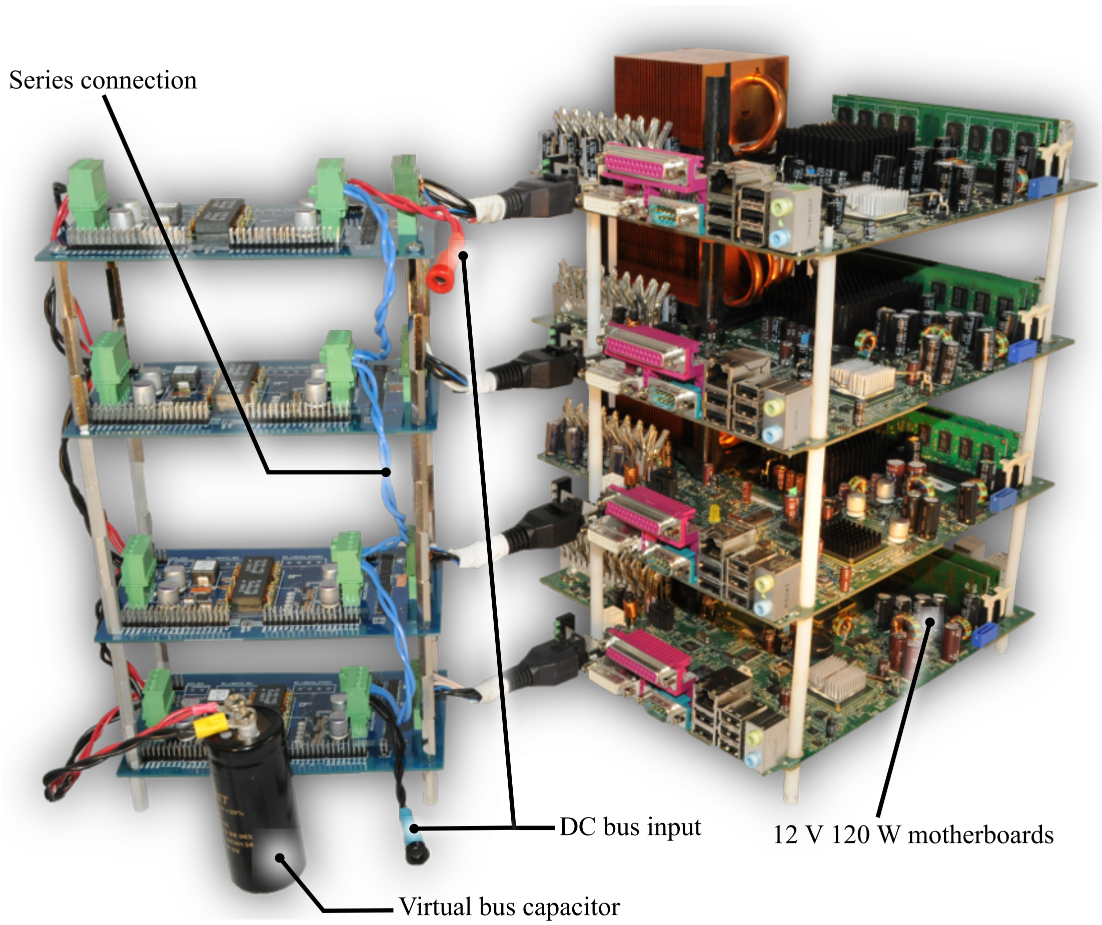


Figure 5.5: Annotated picture of the testbed.

5.2.3 Measurement system

A data acquisition unit was used to sample the annotated signals in Figure 5.4 simultaneously: server voltages (v_{S1} - v_{S4}) and currents (i_{S1} - i_{S4}), series-stack voltages (v_{Stack1} - v_{Stack4}), bus voltage (v_{Bus}) and current (i_{Bus}), differential currents (i_{D1} - i_{D4}) and virtual bus voltage (v_{VB}), at 5000 samples per second. The data acquisition unit consists of an NI PXIe-1078 chassis, PXIe-4300 analog input module, TB-4300 (feedthrough) and TB-4300B (30 to 1 attenuation) terminal blocks. The PXIe-4300 analog input module has 0.02% error, the TB-4300B attenuator has 0.05% tolerance, together yielding 0.07% tolerance in voltage measurements. All voltages are measured by the analog input module and attenuator. Each current measurement is performed with a custom designed current sense board, followed by the analog input module of the data acquisition unit.

Shown in Figure 5.6 is the schematic of the current sense board. It includes a 3 mΩ high power current sense resistor, a high common-mode voltage current shunt monitor, and an isolated dc-dc

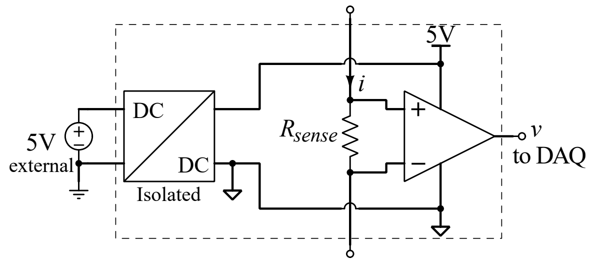


Figure 5.6: Schematic of the custom designed current sense board.

Table 5.4: The key components of the custom design current sense board

Current Sense Resistor	Stackpole Electronics CSNL1206FT3L00 (3 mΩ)
Current Shunt Monitor	Analog Devices AD8210
Isolated Power Supply	CUI PQM1-S5-S5-M

converter with a single regulated output (see Table 5.4 for part numbers). All current sense boards are energized with a separate 5 V DC power supply regulated through an on-board isolated dc-dc converter. The voltage output of each current shunt monitor is calibrated with an Agilent 34410A 6 1/2 digit digital multimeter at each corresponding common mode voltage in order to capture the very high efficiencies of the series-stacked system. After calibration, each current sense board has 0.07% tolerance. Combined with 0.02% tolerance of PXIe-4300 analog input module, current measurements have 0.09% tolerance.

By using this measurement system, power data provided in the remainder of this chapter will carry 0.16% uncertainty.

5.2.4 Efficiency and power loss calculations

The system level efficiency is calculated as follows. The instantaneous input power to the system is calculated by multiplication of the measured instantaneous bus current and voltage,

$$p_{in} = v_{Bus} \times i_{Bus}. \tag{5.1}$$

Each server's instantaneous power is calculated by multiplying the measured server current and voltage, and the total instantaneous output power is the sum of each server's power consumption,

$$p_{out} = \sum_{j=1}^{4} v_{S,j} \times i_{S,j}. \tag{5.2}$$

The difference between (5.1) and (5.2) is the power loss in the system,

$$p_{loss} = p_{in} - p_{out}. \tag{5.3}$$

The loss can be grouped into four categories as follows:

1. Measurement loss ($p_{loss,meas.}$): The current sense boards used to measure the server and differential currents are placed inside the series-stacked system as shown in Figure 5.4. The measurement loss due to the sense resistors is

$$p_{loss,meas.} = \underbrace{\sum_{j=1}^{4} R_{sense} \times i_{S,j}^2}_{\text{due to server current}} + \underbrace{\sum_{j=1}^{4} R_{sense} \times i_{D,j}^2}_{\text{due to differential current}}, \tag{5.4}$$

   where R_{sense} is 3 mΩ.

2. Hot-swapping circuitry loss ($p_{loss,HS}$): The hot-swapping circuitry shown in Figure 5.1 comprises transistors that cause conduction loss due to their on-state resistance. In this work, this is referred to as *hot-swapping circuitry loss*,

$$p_{loss,HS} = \sum_{j=1}^{4} (v_{Stack,j} - v_{Out,j}) \times i_{S,j}, \tag{5.5}$$

   where $v_{Out,j}$ is the voltage at the server terminal of jth prototype DPP hardware,

$$v_{Out,j} = v_{S,j} + i_{S,j} \times R_{Sense} \quad \forall j. \tag{5.6}$$

3. Cabling loss ($p_{loss,cabling}$): As mentioned in Section 5.2.1, the dc bus and hardware prototypes are connected to each other with doubled 16 AWG aluminum wire in order to form the series-stack connection. The total voltage drop in the series-stack connection is

$$v_{drop} = v_{Bus} - \sum_{j=1}^{4} v_{S,j},$$

   which causes conduction loss in the series connection. In this work, this is referred to as *cabling loss*,

$$p_{loss,cabling} = v_{drop} \times i_{Bus}, \tag{5.7}$$

since the current in the series-stack connection is the same as the bus current.

4. Power conversion loss ($p_{loss,conv.}$): As mentioned before, the differential converters in Figure 5.4 are bidirectional dc-dc converters. The measurement of power loss due to power processing in each differential converter thus requires instant detection of power flow. Instead, in this work, all remaining power loss in the system is lumped together in power conversion loss,

$$p_{loss,conv.} = p_{loss} - p_{loss,meas.} - p_{loss,HS} - p_{loss,cabling}. \tag{5.8}$$

In order to calculate efficiency of the series-stacked system, the instantaneous power calculations given by (5.1) - (5.8) are averaged over a time interval:

$$P = \frac{1}{T \times f_s} \sum_{1}^{T \times f_s} p, \tag{5.9}$$

where T is the duration of the time interval, f_s is the sampling rate and P is the average value of the instantaneous power of interest p.

System-level efficiency includes power conversion loss, hot-swapping circuitry loss, and cabling loss but excludes the loss due to the current sensing. It is given by

$$\eta_{sys} = 1 - \frac{P_{loss,sys}}{P_{in}}, \tag{5.10}$$

where

$$P_{loss,sys} = P_{loss,HS} + P_{loss,cabling} + P_{loss,conv.}.$$

On the other hand, the system-level *power conversion efficiency* (i.e., the power processing efficiency of all differential converters) is

$$\eta_{conv.} = 1 - \frac{P_{loss,conv.}}{P_{in}}. \tag{5.11}$$

5.3 Test scenarios

Several test scenarios were executed on the testbed in order to validate the server-to-virtual bus DPP architecture. A test scenario that demonstrated stack initialization and hot-swapping of a

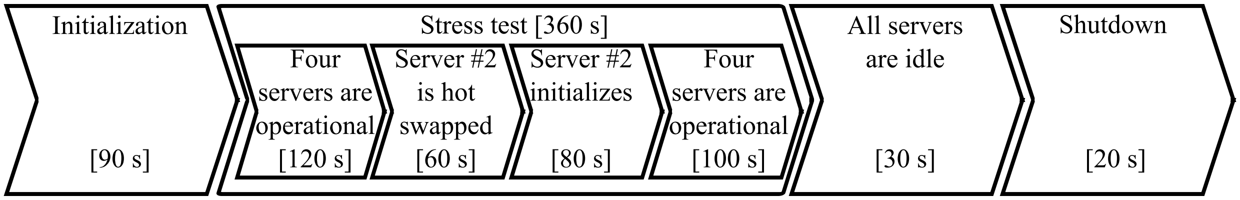


Figure 5.7: A timing diagram for initialization and hot-swapping test scenario.

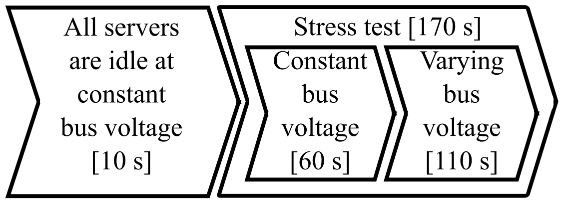


Figure 5.8: A timing diagram of decaying bus voltage test scenario.

server while bus voltage is constant, and another test scenario that demonstrated operation of all series-stacked servers when the bus voltage is decaying, are explained below. In both scenarios, the standard Linux "stress" utility [129] was used as a computational load on servers in order to replicate a real-world computation scenario. Also, continuous operation of series-stacked servers under a web traffic management algorithm is not mentioned here but can be found in [115].

5.3.1 Test I: Initialization and hot-swapping

A 500 s test was executed on the testbed in order to validate the hot-swapping concept with the server-to-virtual bus DPP architecture. The test started with initialization of the series-stacked converters with shunt resistors as described before. Then, four servers were connected to the series stack with the hot-swapping circuitry. As the operating system on the servers initialized, an Ethernet connection was established in order to start the stress utility for 360 s. Two minutes into the stress test, one of the servers was swapped out and kept isolated from the stack for one minute, while the other servers continued the stress test. The swapped-out server was then swapped into the stack and re-initialized, and the stress test continued for almost two more minutes. Following the conclusion of the stress test, shutdown commands are sent to the servers. The shunt resistors were then connected back to the series-stack nodes in order to keep voltage balanced throughout the stack while the dc bus was disconnected. A timing diagram of this initialization and hot-swapping test scenario is given in Figure 5.7.

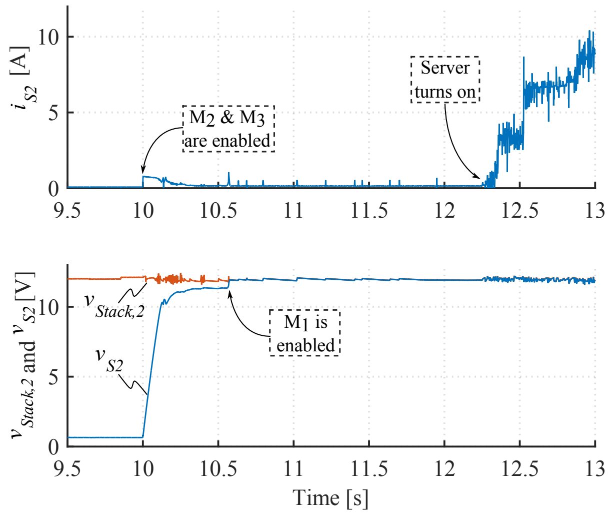


Figure 5.9: Measured instantaneous data showing swapin transient of the second server.

5.3.2 Test II: Decaying bus voltage

A 180 second test was executed on the testbed in order to validate operation of the server-to-virtual bus DPP architecture under a varying dc bus. This test skipped initialization of the series-stacked converters, and assumes it can be achieved as explained in Test I. In this test, four servers stayed idle for 10 seconds and executed a computational load generated by the standard Linux "stress" utility for 170 seconds. During the first 60 seconds of this test, the bus voltage was kept constant at 52 V in order to represent fixed bus voltage operation of the proposed architecture. Then, the bus voltage was manually decreased by adjusting the programmable output of the HP 6674A dc power supply from 52 to 44 V to represent a decaying UPS voltage after a power loss. A timing diagram of decaying bus voltage test scenario is given in Figure 5.8.

5.4 Results

5.4.1 Test I: Initialization and hot-swapping

When the dc bus was first applied to the series stack at the beginning of the test, all servers were isolated from the stack and the shunt resistors were connected between the series-stack terminals of the DPP hardware. The applied bus voltage was thus equally divided between the shunt resistors, allowing linear regulators on DPP hardware boards to provide logic voltages to digital isolators and gate drives. The differential converters were then enabled to regulate both their input and

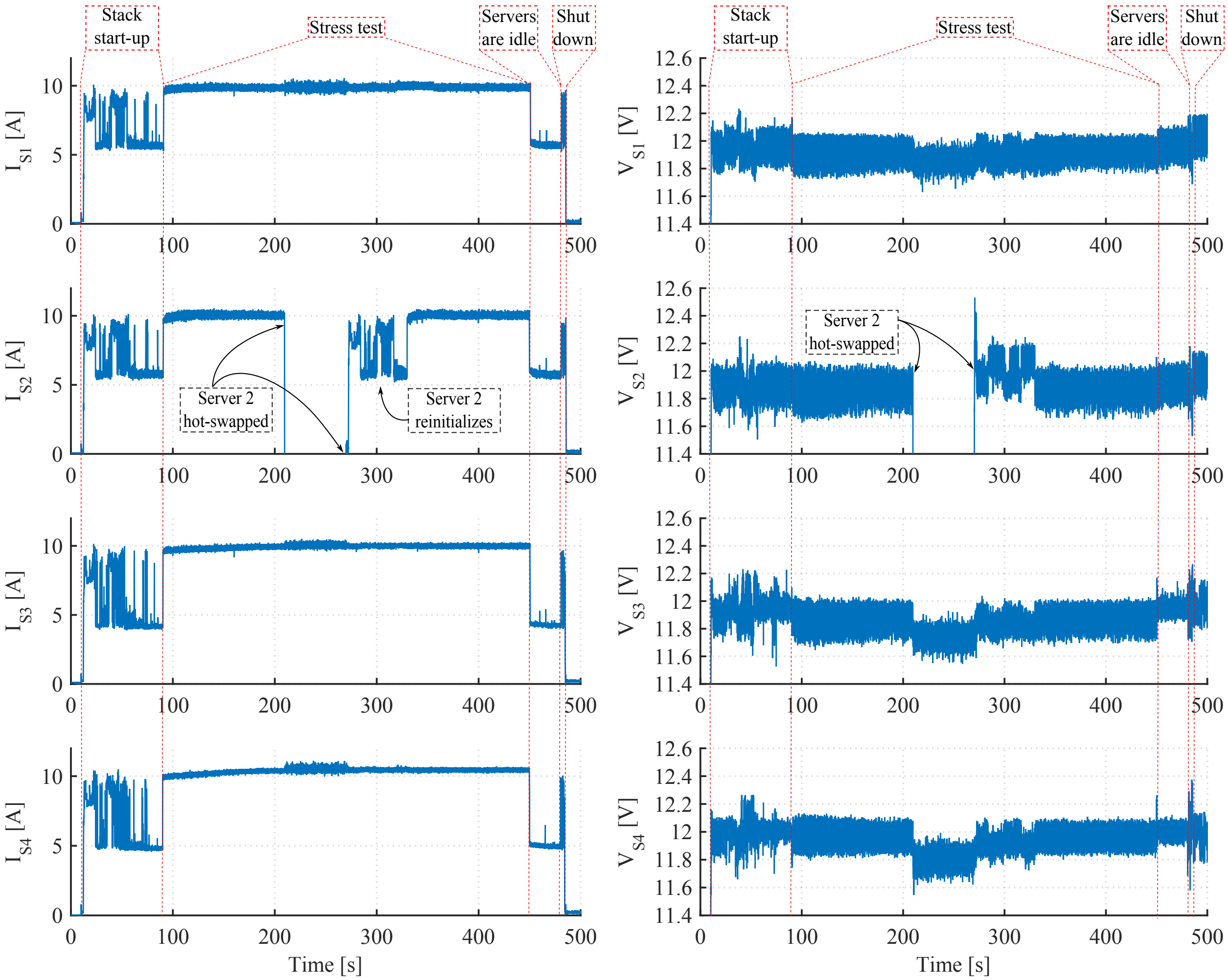


Figure 5.10: Measured server currents and voltages. (Measured data is 10 ms window averaged for better illustration of the entire test on a single plot.) Major events during the experiment are annotated on the plot. Note the absence of the second server current and voltage when it is hot-swapped out of the stack.

output voltages to 12 V. The shunt resistors were disconnected a few seconds after the control algorithm initialized as explained in Section 5.1.3. The servers were then connected simultaneously to the series stack, 10 s into the experiment, by using the hot-swapping circuitry. This operation (previously explained in detail in Section 5.1.3) is shown in Figure 5.9 through the measured and annotated current and voltage of the second server as an example of the swapin transient during this experiment. At 10 s, the high resistance path of the hot-swapping interface (given in Figure 5.1) was enabled by turning on M_2 and M_3. As can be seen in Figure 5.9, the server current was limited to less than 1 A by M_2 operating in its linear region, causing a linear increase of the server voltage (v_{S2}). At about 10.7 s, the low-resistance path of the hot-swapping interface was slowly enabled

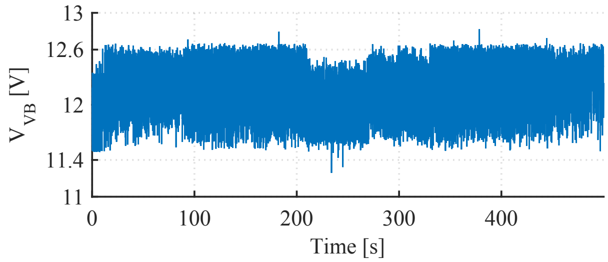
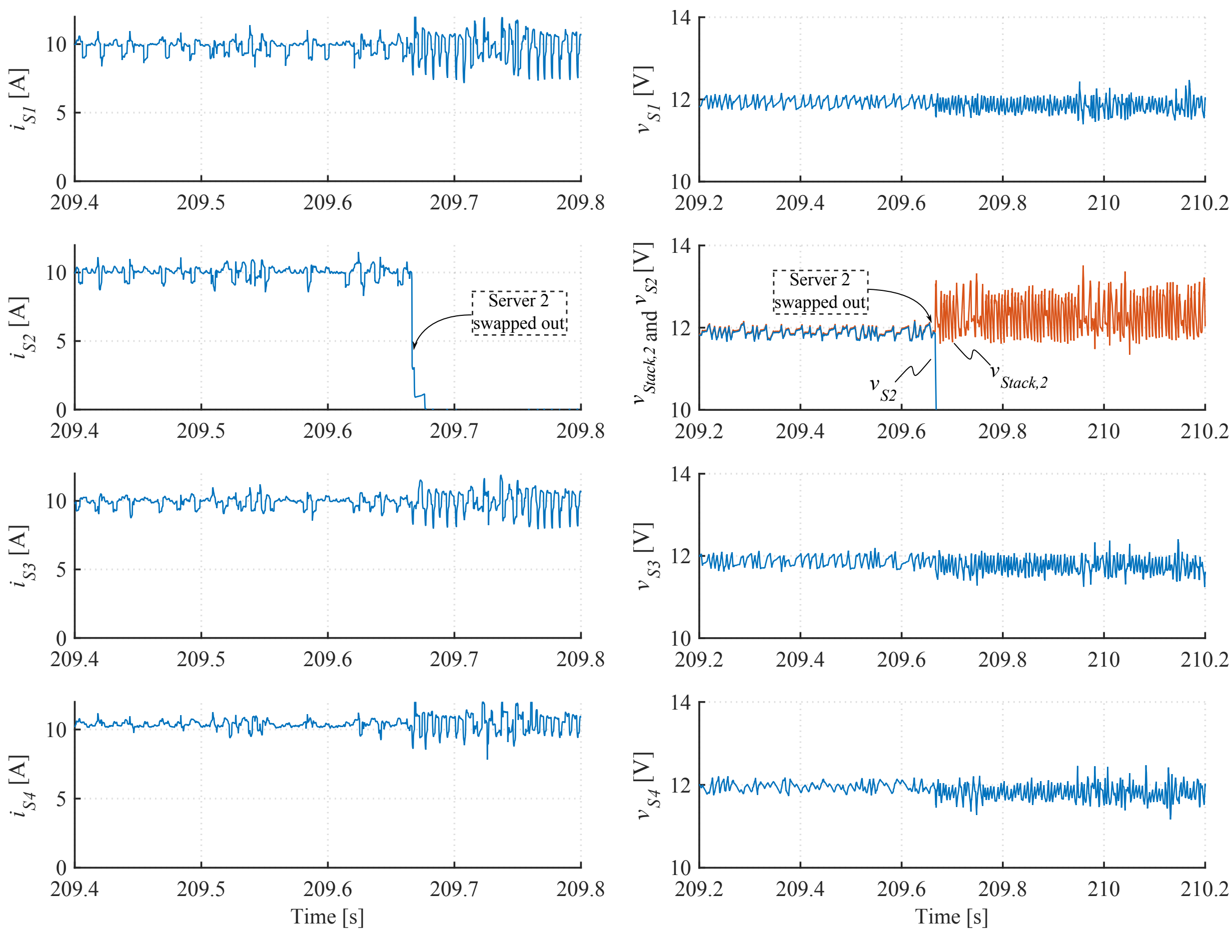


Figure 5.11: Measured virtual bus voltage that shows its successful regulation. (Measured data is 10 ms window averaged for better illustration of the entire test on a single plot.)

through M_1, causing a small current increase due to the voltage drop across R_{limit} in Figure 5.1. The server turned on at about 12.3 s. Although this mechanism is demonstrated for one server here, similar behavior was observed in every server in the series stack.

Figure 5.10 shows measured and 10 ms window averaged server currents and voltages, and Figure 5.11 shows measured and 10 ms window averaged virtual bus voltage during the entire test. Following swapin of all servers at about 10 s, initialization of servers started at approximately 12 s and takes approximately 80 s. During this time interval, system-level efficiency is measured as 97.2%. The computation test was started on servers at 90 s. During the first two minutes of the test, all four servers were executing the computation test and system-level efficiency is measured as 98.4%. At about 210 s, the second server was swapped out, and kept isolated for approximately one minute. As shown in Figure 5.10, the differential converters were able to regulate the operating server voltages and the virtual bus voltage while the second server's current and voltage were zero. During this time interval, system-level efficiency decreased to 95% because the differential converter of the second server processed full bus current and acted as a dc voltage sink by regulating $v_{Stack,2}$. The second server was swapped into the series stack and re-initialized around at about 270 s, while the other servers were still executing the computation test. During this time interval, system-level efficiency was 97.9%. After the second server's re-initialization was completed, the computation test continued on all four servers for two more minutes, and system-level efficiency during the last minute of the test was 98.3%. After the stress test was completed, the servers were kept in their idle state before the shutdown command was executed at about 480 s. During this time interval, system-level efficiency was 96.9%. The server currents immediately went to zero; however, server voltages were regulated to 12 V until all servers were isolated from the series stack by using the hot-swapping circuitry. The series-stacked system then returned to its initial state by reactivating the shunt resistors to allow safe voltage transients as the dc bus was disconnected from the series



(a) Server currents.

(b) Server voltages.

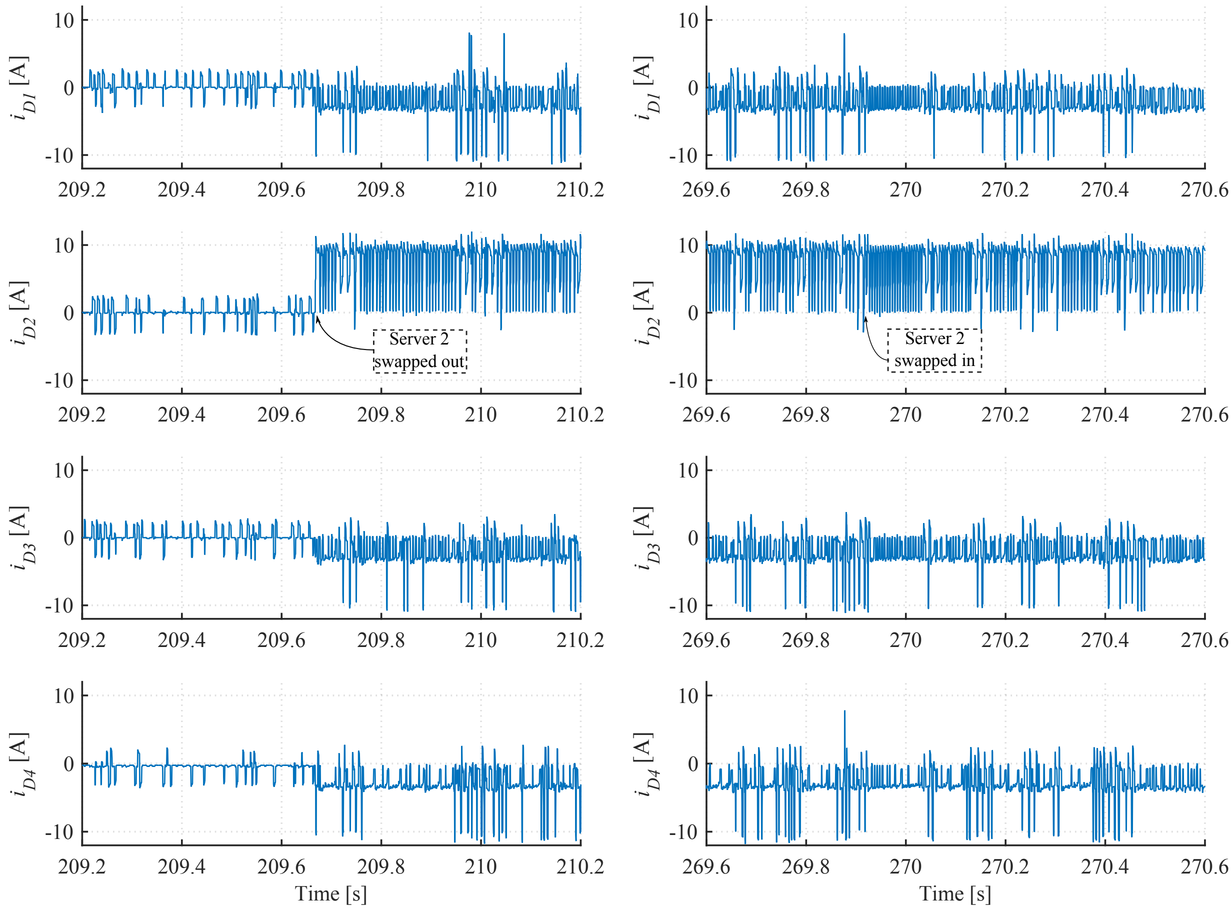
Figure 5.12: Instantaneous server waveforms during swapout.

stack. A breakdown of the average input and output powers, the efficiency, and the average power loss is given in Table 5.5 for the entire test. Recall that the power measurements provided in Table 5.5 carry 0.16% uncertainty.

Figure 5.12 plots the instantaneous server waveforms during the swapout of the second server at about 210 s. In order to demonstrate and explain the operation of the bidirectional hysteresis algorithm, V_{Stack2} is plotted along with V_{S2} in Figure 5.12(b), and also the instantaneous differential currents into the virtual bus node are plotted in Figure 5.13. Before the second server was swapped out at about 210 s, all server voltages were regulated to a hysteresis band (as shown in Figure 5.12(b)) while the differential converters were operating in light-load mode with bidirectional hysteresis control (as shown in Figure 5.13(a)). Right after the second server was swapped out, the light-load operation of the second differential converter was not sufficient to regulate V_{Stack2}

Table 5.5: Breakdown of average input and output powers, efficiency, and average power loss during the experiment

	Stack Startup	Stress Test	Server 2 Hot-Swapping	Server 2 Startup	Stress Test	Shut Down
Time Interval [s]	$0 < t < 90$	$90 < t < 210$	$210 < t < 270$	$270 < t < 330$	$330 < t < 450$	$450 < t < 500$
$< P_{in} >$ [W]	264.9	481.9	378.8	452.5	488.5	192.6
$< P_{out} >$ [W]	257.1	472.8	358.6	441.7	479.2	186.4
$< P_{loss,meas.} >$ [W]	0.5	1.2	1.2	1.1	1.2	0.3
$< P_{loss,sys} >$ [W]	7.3	7.9	19.0	9.7	8.1	5.9
$< P_{loss,HS} >$ [W]	1.1	1.6	1.1	1.5	1.6	0.5
$< P_{loss,cabling} >$ [W]	0.9	2.4	1.6	2.2	2.5	0.5
$< P_{loss,conv.} >$ [W]	5.3	3.9	16.3	6.0	4.0	4.9
$< \eta_{sys} >$ [%]	97.2	98.4	95.0	97.9	98.3	96.9
$< \eta_{conv.} >$ [%]	98.0	99.2	95.7	98.7	99.2	97.5



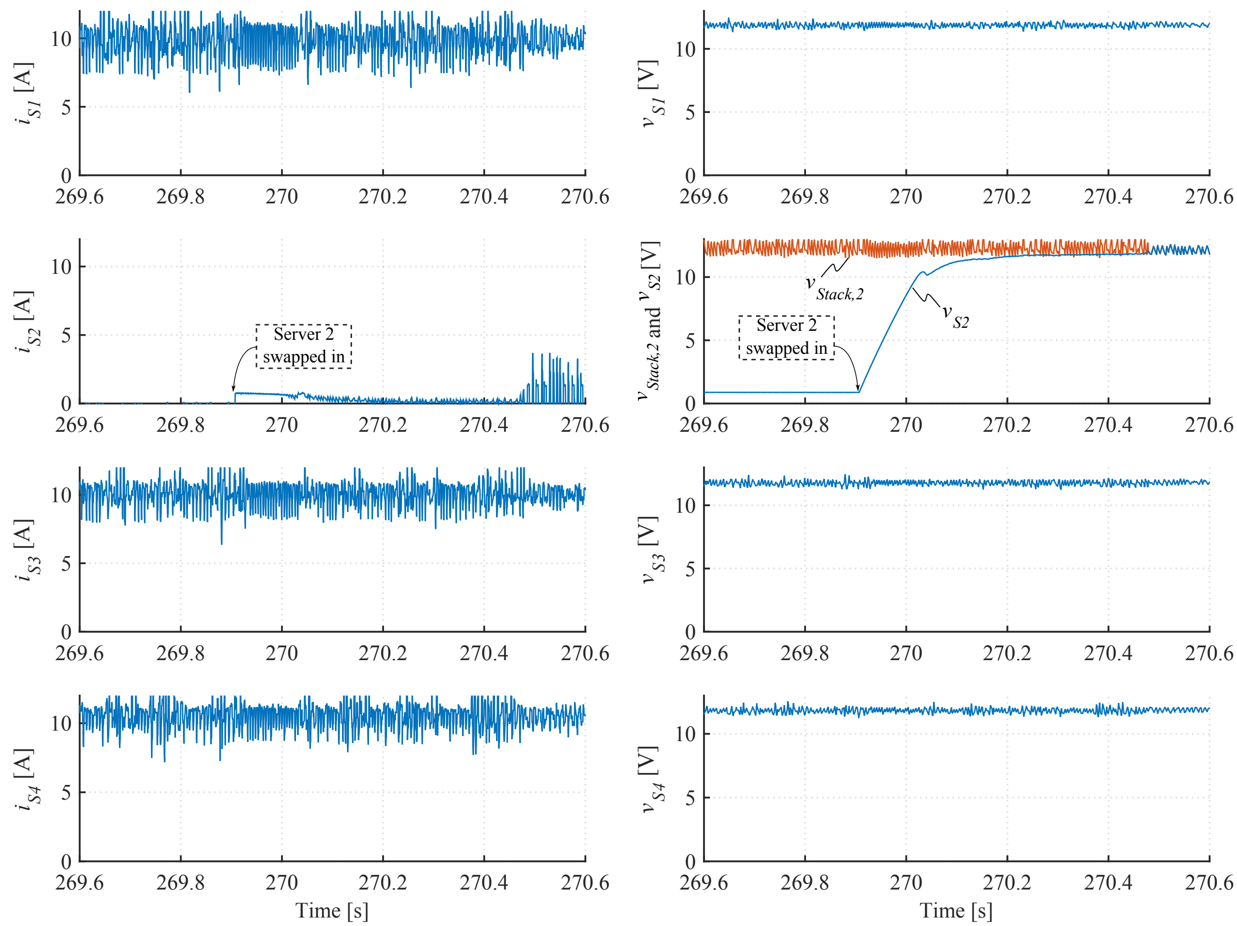
(a) During swapout.

(b) During swapin.

Figure 5.13: Instantaneous differential currents into the virtual bus node.

within the same hysteresis band as before. The second differential converter thus switched to full-load mode, with increased hysteresis bands, while the other differential converters were still able to regulate their server voltages while mostly maintaining their light-load operation mode as before the swapout occurred. Note that the frequency of the server voltage ripple increased slightly during hot-swapped operation. This indicates that the differential converters turn on and off more often than in normal operation, which aligns with increased average power loss during hot-swapped operation.

The instantaneous server waveforms during the swapin of the second server at about 270 s are also illustrated in Figure 5.14. Starting from 269 s, the voltage and current of the second server increased in a controlled manner, similar to the demonstration in Figure 5.9. Following the 270 s, the second server initialized; however, the second differential converter still remained in full-load hysteresis mode since the second server's power consumption during initialization is quite different



(a) Server currents.

(b) Server voltages.

Figure 5.14: Instantaneous server waveforms during swapout.

from that of the remaining servers as they continue the stress test.

The series-stacked architecture separates the processed power from the delivered power. By processing only the power difference throughout the experiment, system-level power conversion efficiency that is higher than the efficiency of the differential converters is achieved. When the servers are almost equally loaded during the stress test (i.e., 90 s $< t <$ 210 s and 330 s $< t <$ 450 s), the differential converters process insignificant amounts of power in the system, yielding above 99% power conversion efficiency. The average power loss distribution while the series-stacked servers are executing the stress test between 90 s and 210 s is also demonstrated in a pie chart in Figure 5.15. Here, power conversion losses are reduced to almost half of the overall losses in the system, while the other half is basically shared as conduction loss between the hot-swapping circuit and the cabling.

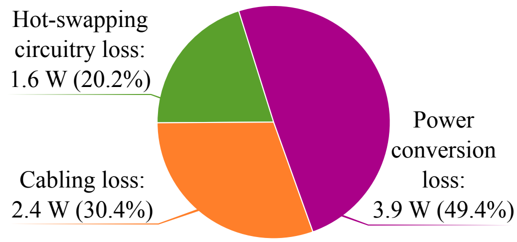
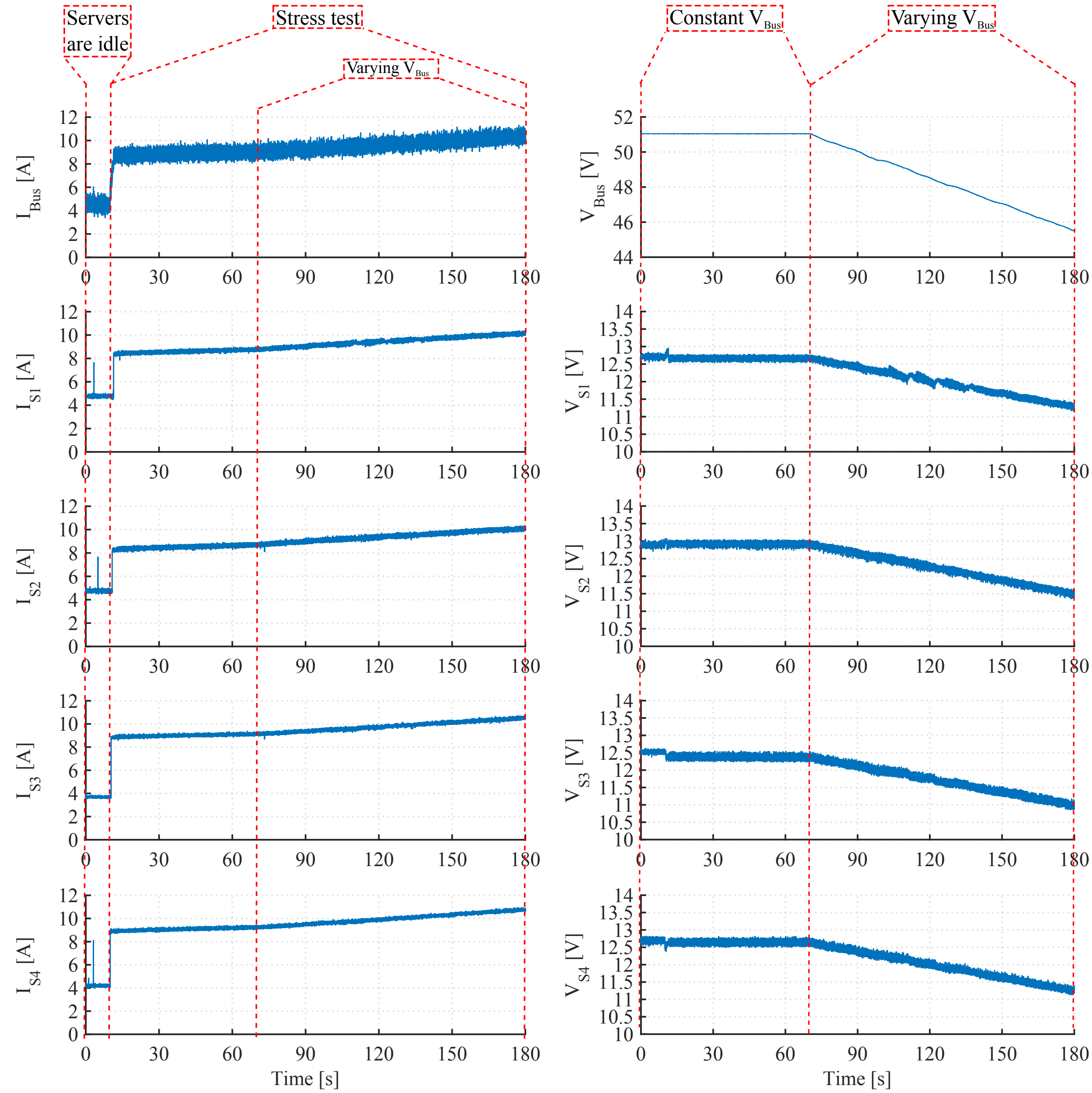


Figure 5.15: A pie chart of $P_{loss,sys}$ distribution during the stress test 90 s $< t <$ 210 s. During this time interval, an average of 472.8 W was delivered to the servers.

5.4.2 Test II: Decaying bus voltage

In this test, after successful initialization of the series-stacked servers as explained in Section 5.4.1, four servers stayed idle for 10 s, followed by execution of a computational load generated by the standard Linux "stress" utility for 170 s. During the first minute of the stress test, the bus voltage was kept constant, and then the bus voltage was manually decreased by using the programmable output of the HP 6674A dc power supply.

Annotated plots include 10 ms window averaged voltage and current waveforms of the servers and the bus, and also the voltage waveform of the virtual bus, are given in Figures 5.16 and Figure 5.17 for the entire experiment. As shown in Figure 5.16(b) and Figure 5.17, series-stacked server voltages and the virtual bus voltage were successfully regulated within a 10.5 V-13 V range, depending on whether the bus voltage was constant or varying. While bus voltage was constant (i.e., 0 s $< t <$ 70 s), server voltages were regulated to constant values near 13 V since the reference voltage input to the control algorithm was constant as well. On the other hand, while bus voltage was varying (i.e., 70 s $< t <$ 180 s), the reference voltage input to the control algorithm varied. This resulted in variation of server voltages and the virtual bus voltage. Since the voltage variation was within the allowed range of the input server voltage, operation of the series-stacked servers was maintained. As can also be seen in Figure 5.16(a), server currents vary, depending on whether the bus voltage was constant or varying, since each server was a constant power load regardless of its input voltage. While the bus voltage was constant (i.e., 0 s $< t <$ 70 s), the server currents were constant at about 5 A when they were idle (i.e., 0 s $< t <$ 10 s), or at about 8.5 A when they executed the stress test (i.e., 10 s $< t <$ 70 s). On the other hand, while the bus voltage was varying (i.e., 70 s $< t <$ 180 s), server currents increased as their input voltages decreased following the reference input to the control algorithm. Average input and output power of the system, average power loss in the system, and system-level efficiency for the various time intervals of the experiment are given



(a) Current waveforms.

(b) Voltage waveforms.

Figure 5.16: Measured server and bus waveforms during the experiment. Measured data is 10 ms window averaged for better illustration of the entire test on a single plot.

in Table 5.6. Recall that the power measurements provided in Table 5.6 carry 0.16% uncertainty.

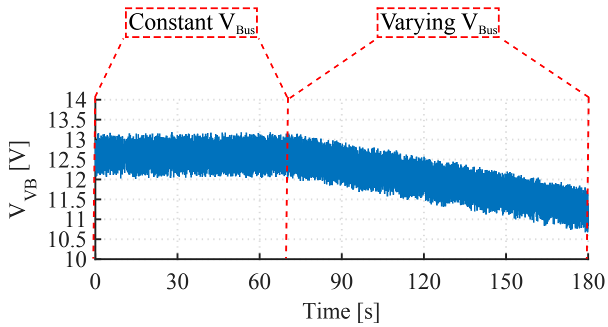


Figure 5.17: Measured virtual bus voltage during the experiment. Measured data is 10 ms window averaged for better illustration of the entire test on a single plot.

Table 5.6: Breakdown of average input and output powers, efficiency, and average power loss during the experiment

	Servers are idle	Stress Test		Overall
Time Interval [s]	Constant V_{Bus} $0 < t < 10$	Constant V_{Bus} $10 < t < 70$	Varying V_{Bus} $70 < t < 180$	$0 < t < 180$
P_{in} $[W]$	226.0	444.8	463.0	443.8
P_{out} $[W]$	221.7	441.9	460.4	441.0
P_{loss} $[W]$	4.3	2.9	2.6	2.8
η [%]	98.1%	99.3%	99.4%	99.4%

CHAPTER 6

SINGLE-PHASE AC TO DC POWER CONVERSION

As mentioned in Chapter 2, due to dc voltage supply needs of servers, data centers must employ power converters to rectify the ac grid voltage at some point in the power conversion architecture. In data centers, the ac voltage may be available at various different voltage levels, at 50 or 60 Hz frequency and as three phases or a single phase. This chapter focuses on single-phase ac to dc power conversion from universal voltage (approximately 90 to 240 V_{RMS} at 50/60 Hz) that is delivered to the rack or blade level.

6.1 Motivation

For ac to dc power conversion at the rack or blade level, the commonly preferred topology steps up a rectified ac voltage in order to achieve power factor correction and twice-line frequency energy buffering. However, the ultimate goal of the power delivery architecture is to generate a well-regulated low dc voltage for digital circuits. Figure 6.1 shows an example conventional data center power delivery architecture which was mentioned in Chapter 2. Here, it is depicted again with annotated voltage levels throughout the power delivery chain to facilitate discussion.

In the conventional power delivery architecture depicted in Figure 6.1, utility scale 50/60 Hz transformers and power distribution units provide single-phase ac power (e.g., 240 V_{RMS}) to the server racks. The single-phase ac voltage is rectified at the rack level by a diode bridge or active rectifier, and then boosted up to a higher dc voltage (e.g., 400 V) for power factor correction (PFC) and twice-line frequency energy buffering. The high dc voltage is then stepped back down to a lower dc voltage (e.g., 48 V) to be delivered to server blades through a dc bus. A recent trend in data center power delivery applications is to place the uninterruptible power supply (UPS) or battery backup units at the dc bus in the rack, which is also illustrated in Figure 6.1. In this architecture, achieving PFC using a boost-type front end converter requires a dc voltage which is significantly higher than the final load voltage in data center applications. Although the high dc voltage in

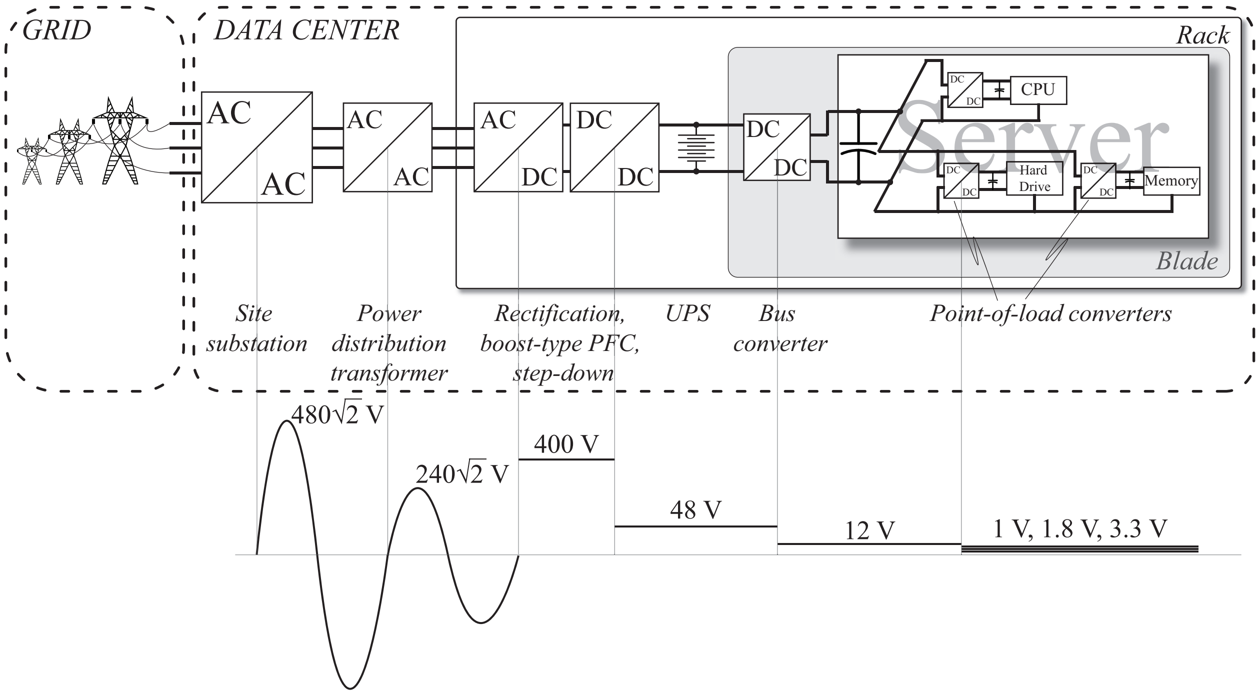


Figure 6.1: An example of conventional power delivery in data centers, illustrating major power conversion stages with annotated voltage levels at each stage.

the front end converter also creates an effective environment to buffer twice-line frequency energy using low-cost and energy dense electrolytic capacitors, the front end converter must be followed by a high conversion ratio dc-dc converter in order to step down the high dc voltage to a lower dc level before the power is distributed throughout the server motherboard.

A single-stage solution in this application could have various advantages. First, stepping up the voltage for PFC and twice-line frequency energy buffering, and then stepping down the voltage for power distribution on the motherboard, is a counterproductive approach since the final loads are at various low dc voltage levels that are much below the high dc voltage created to achieve PFC. In addition, a two-stage solution requires both power stages to be optimized, implemented and tested separately. Furthermore, the power is being processed twice, limiting system-level efficiency and increasing total power converter footprint. Recently many research efforts have focused on the efficiency and power density improvements of boost-type PFC converters, and high voltage step-down converters. For instance, in recent literature a carefully optimized boost-type PFC converter has an efficiency curve ranging from 97.7% to 98.8% and a power density of 220 W/in^3 [34]. A carefully optimized 400 V to 48 V dc-dc converter has a peak efficiency of 94.5% and a power density of 164 W/in^3 [40]. Combining these two stages would yield a best-case 93.4% efficiency and

94 W/in^3 power density. On the other hand, commercial products achieve 92% typical efficiency and 140 W/in^3 power density [130] for boost PFC, and 93.6% peak efficiency and 258 W/in^3 power density [131] for 400 V to 48 V dc-dc conversion. Combining these two stages would yield a best-case 86.1% efficiency and 90.8 W/in^3 power density. Note that these converter efficiencies and power densities do not include twice-line frequency energy buffering.

This work seeks to leverage a 48 V UPS, which is a large energy storage component in a data center power delivery system, to handle twice-line frequency energy buffering at the dc bus, and to explore potential efficiency and power density improvements when a single-stage power converter topology is employed in data center applications to rectify 240 V_{RMS} to 48 V dc. Therefore, the motivation of this work is to explore a single-stage solution to perform PFC in single-phase 240 V_{RMS} ac to 48 V dc power conversion.

6.2 Buck-type power factor correction

In grid-connected power supplies, PFC is often employed to meet power quality requirements mandated by standards such as IEC/EN 61000-3-2 and EnergyStar. Although a data center power distribution system may not be connected directly to the ac grid, power quality in a data center facility is still an important consideration. The 80 Plus certification [132], which is a voluntary program aimed to encourage more efficient power supply units for computer applications, is also widely accepted in data center power delivery architectures. Although 80 Plus certification was originally intended for power conversion efficiency, currently its highest tier (e.g., 80 Plus Titanium) requires above 0.95 power factor for 20-100% of rated load [132].

There are many control techniques and power converter topologies for single-phase PFC operation [133]; however, the boost-type PFC converter is preferred in high power and high voltage applications since it can offer unity power factor and a sufficiently high intermediate voltage to buffer twice-line frequency energy in an effective way. On the other hand, a buck-type converter in a single-phase ac to dc application (depicted in Figure 6.2) can also perform PFC [134–137], though unity power factor is not possible. This limitation is due to the nature of the converter, i.e., when the input voltage is less than the output voltage; the buck converter cannot draw current from its input terminal. Buck-type PFC thus trades off bucking a high ac input voltage directly to a desired low dc output voltage with some ac input current distortion that limits achievable power factor. Note that in Figure 6.2, a full-bridge rectifier is assumed to be used to create the

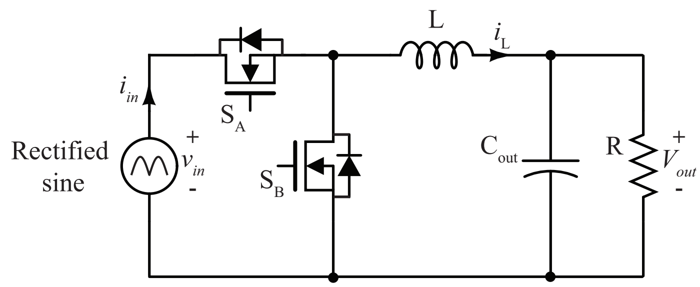


Figure 6.2: Buck converter.

rectified sine voltage source from an ac source in order to prevent the output voltage from being discharged through the body diode of the high side switch when the input voltage is below the output voltage. Ideal operation waveforms of a buck converter achieving best case power factor are illustrated in Figure 6.3 for a rectified ac input of 240 V_{RMS} at 60 Hz and a fixed dc output voltage of 48 V, for an example case of 10 A average output current. Assuming ideal control and filtering, the inductor current and the duty cycle of the buck converter that achieve best case power factor are also illustrated in Figure 6.3 to facilitate the discussion.

Since a buck converter cannot supply power to its load when the input voltage is less than the output voltage, the ac input current conduction angle, shown as α in Figure 6.3, is less than 180deg. As it is also apparent in the duty ratio plot in Figure 6.3, while the ac line voltage is less than the output voltage, the duty ratio (i.e., the on-time of S_A) is zero, which indicates that the buck-type PFC converter is disabled at the beginning and end of each ac half-cycle. During this time, the requisite load energy is provided by the output capacitor, which is assumed to be infinitely large in the ideal waveforms shown in Figure 6.3. In addition, further examination of Figure 6.3 shows that the start and end of α are determined by the instantaneous input and output voltage. Therefore, the limit of achievable power factor depends on the input and output voltage, although higher input current offers higher power factor. In this work, the target input voltage range is 90 V_{RMS} to 240 V_{RMS} and the target output voltage is 48 V. These conditions, assuming a fixed output voltage and filtered input current, limit the theoretically achievable PF to 0.9967 for 90 V_{RMS} and 0.9988 for 240 V_{RMS}.

In practice, because of PFC control limitations, output voltage ripple and input filtering, power factor is expected to be lower than the theoretical limit. Any PFC control method has limitations due to controller bandwidth and energy which results in input current distortions, reducing the achievable power factor. Further discussion of buck-type PFC control and its limitations is provided

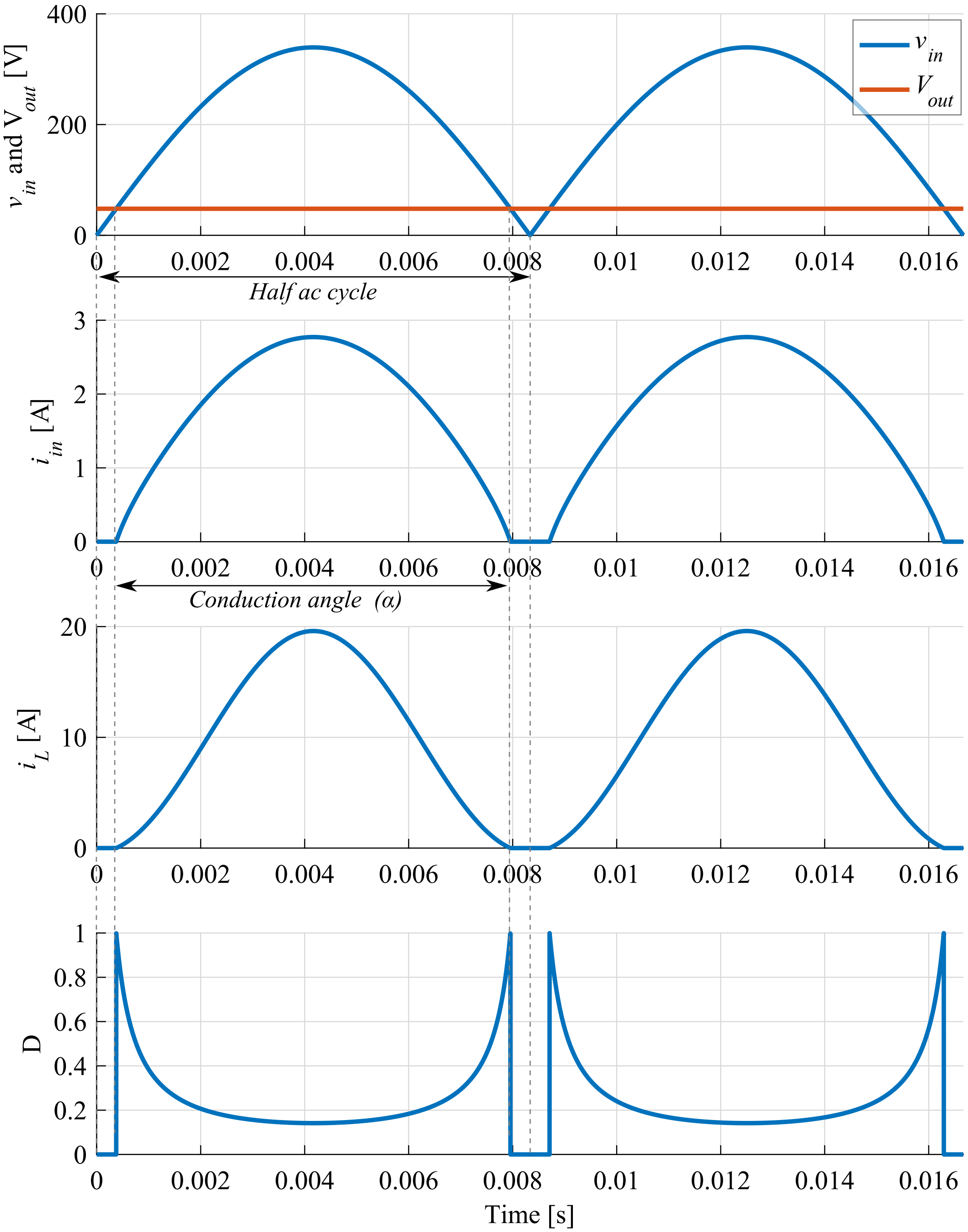


Figure 6.3: The input voltage and current, inductor current, and duty cycle in a buck PFC converter for a full ac line cycle at 60 Hz, with a 10 A average output current, assuming ideal control and filtering.

in Chapters 7 and 8. Here, output voltage ripple and input filtering effects on a buck PFC converter are discussed. The output voltage ripple in a buck PFC converter may result in input current displacement, reducing the achievable power factor. Since a buck converter cannot provide power

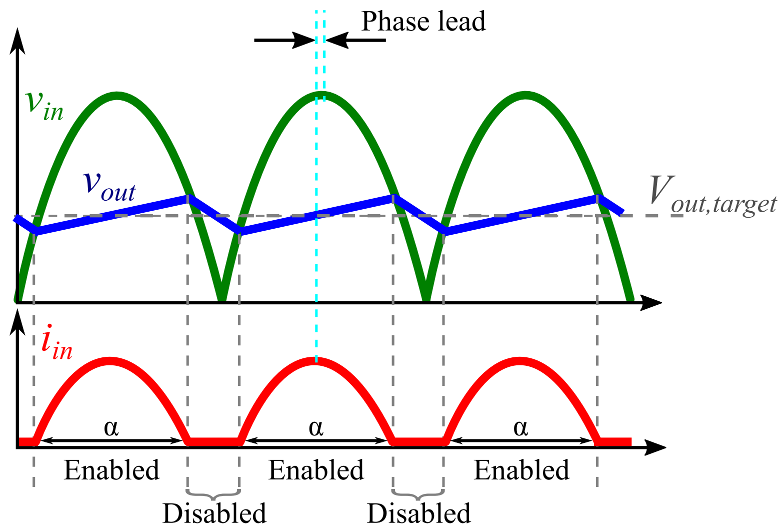


Figure 6.4: Phase shift of input current due to output voltage ripple.

to its output and must be disabled when the input voltage is less than the output voltage, the output capacitor provides the requisite load energy at the beginning and end of each ac half-cycle. The output voltage of the buck converter will thus decrease when the converter is disabled during PFC operation. The rate of voltage decrease is directly proportional to load current, and inversely proportional to output capacitance. This voltage decrease causes an earlier turn-on in the next ac half-cycle, as depicted by the representative waveforms of input voltage and current, and output voltage without twice-line frequency energy buffering ripple in Figure 6.4. Similarly, since the converter must both compensate for the voltage decrease and provide the requisite energy to the output when enabled, the output voltage must reach a value above the reference value, so that the average output voltage over the full ac cycle is equal to the reference value. This causes an earlier turn-off as depicted in Figure 6.4. The voltage decrease when the converter is disabled can be reduced by employing a higher capacitance at the converter output. For example, less than half a degree of input current displacement requires at most 5% decrease from 48 V when the converter is disabled. At 500 W constant output power, at least 3.3 mF of output capacitance is needed to limit the output voltage ripple within ±5% of 48 V. The output voltage ripple displaces the input current, further reducing the power factor. The input filter is another reason for input current displacement in PFC converters, reducing the achievable power factor. Design of a PFC converter input filter requires careful design trade-offs in order to minimize input current displacement [138]. In a buck PFC where the input current ripple at the switching frequency must be substantially attenuated, input filtering may severely disrupt the phase angle of the input current and reduce the power factor. In summary, practical limitations of control, nonnegligible output voltage ripple, and input filtering impose limitations on achievable power factor in a buck-PFC converter.

In addition to a power factor limitation, a few other important points regarding buck converter PFC operation can be highlighted by inspecting Figure 6.3. Recall that input and inductor currents in Figure 6.3 are plotted for an example of 10 A average output current (i.e., 480 W output power at 48 V) and assuming perfect filtering. In a practical implementation of such an operating condition, the inductor current also exhibits current ripple which is given by

$$\Delta i_L = \frac{(1-D)V_{out}}{Lf_{sw}}, \tag{6.1}$$

where f_{sw} is the switching frequency. This means that the inductor in the circuit must have saturation current well above 20 A, and also must withstand the full input voltage as it is directly connected to the input when the high-side switch (S_A in Figure 6.2) is closed. Such an inductor is potentially quite large, and limits converter power density by dominating converter volume. The conventional way of reducing the inductor requirement by increasing f_{sw} is challenging, as the high output current translates to semiconductors with low resistance, and typically fairly large parasitic capacitances that limit the practical switching frequency. In addition, the ideal duty cycle in Figure 6.3 is below 20% for more than half of operating time, which also occurs when the inductor current is above the average output current. This means that the buck converter needs to perform high voltage step-down conversion when the highest power is also transferred from input to output, which is the least efficient operating condition for a buck converter. On the other hand, relatively more efficient operating points of a buck converter (i.e., $0.4 < D < 1$) occur during times of lower power transfer, only performing moderate voltage conversion at relatively low current. These challenges of employing the buck converter as a PFC converter, combined with limited achievable power factor, have made the buck converter less commonly employed in single-phase PFC applications.

In order to improve achievable power factor, a low voltage boost stage can be integrated after the inductor in the buck converter as depicted in Figure 6.5. Such a configuration is commonly referred to as a four-switch noninverting buck-boost converter or a cascaded buck-boost converter in the literature, and has been used to perform PFC in ac-dc power conversion [139–143]. The cascaded buck-boost converter consists of a buck stage, a shared inductor, and a boost stage as depicted in Figure 6.5. The operation of this converter can be divided into two modes, as shown in Figure 6.6,

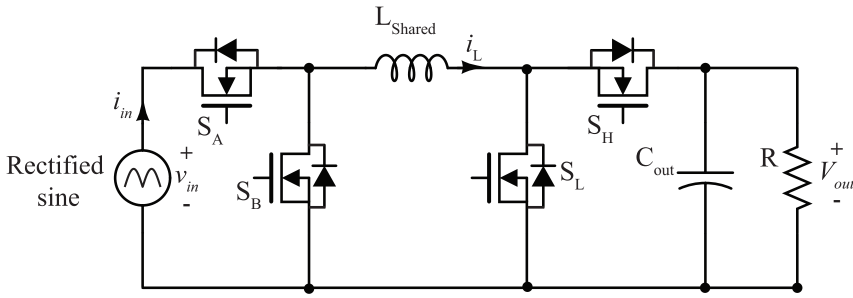


Figure 6.5: An ac-dc converter with noninverting buck-boost converter.

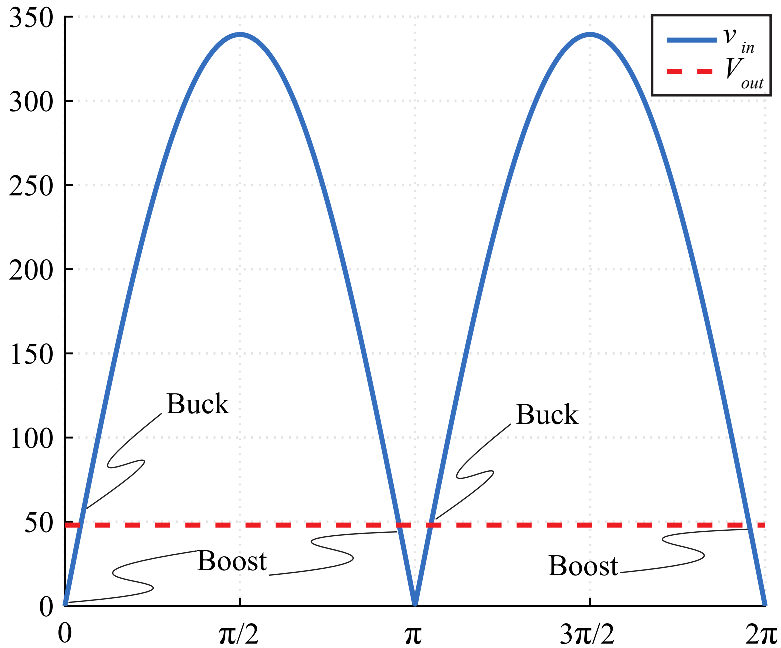


Figure 6.6: Operation modes of the noninverting buck-boost converter for ac-dc rectification.

to achieve unity power factor. Switches S_A and S_B, and S_H and S_L, are run in a complementary manner. While the rectified input voltage is lower than 48 V, S_A is kept continuously on (S_B continuously off) while S_H and S_L are modulated for boost operation. On the other hand, while the input voltage is higher than 48 V, S_H is kept continuously on (S_L continuously off) while S_A and S_B are modulated for buck operation. Note that the low-voltage boost stage is only activated when the input voltage is lower than the output voltage, meaning that the input voltage is only boosted to the desired low output dc voltage (i.e., 48 V), rather than a higher dc voltage observed in a conventional boost-type PFC converter. By using the additional boost stage, the achievable power factor of the buck-type PFC converter can be extended at low expense, since the boost stage processes low power at low voltage, and only adds minor conduction loss through its high side switch when not activated. Addition of the boost stage can also mitigate input current phase lead caused by output voltage ripple due to turn-off of the buck converter when the input voltage is less than the output voltage.

Although achievable power factor can be increased by adding a boost stage, buck or four-switch noninverting buck-boost PFC converters still require large inductors and suffer from high voltage step-down at high output current. As mentioned earlier, since converter losses here are dominated by conduction loss due to high output current, switching frequency increases are difficult to realize in practice to reduce the inductor requirement. In order to improve power density of the buck-type PFC converter, the two-level buck stage can be replaced with a flying capacitor multilevel converter stage.

6.3 Flying capacitor multilevel converters

Flying capacitor multilevel (FCML) converters [144] were initially employed in high voltage and high power dc-dc and dc-ac converters where there is no available semiconductor switch that can sustain the required voltage in the converters. Over the years, FCML converters have been explored in numerous applications [101, 145, 146], including envelope tracking [147], power factor correction [148], and renewable energy systems [149]. More recently, with the adoption of wide band-gap semiconductor (GaN) switches, FCML converters have been revisited for non-isolated dc-dc [45,146,150–153], dc-ac [154–157], and ac-dc [36,158,159] power conversion to increase power density of conventional two-level buck and boost topologies. In particular, the seven-level boost PFC in [159] provided 2.2x improvement in power density in comparison to a two-level boost PFC [34] without compromising efficiency. Similar performance improvements could be made feasible by transitioning to a multilevel design in a buck-type PFC. An N level FCML converter employed in a dc-dc voltage step-down application is depicted in Figure 6.7, which is used to briefly explain the operation and key advantages of FCML converters.

Operation of an FCML converter can be summarized as follows. In Figure 6.7, switch pairs S_{iA} and S_{iB}, where $i = 1, 2, ..., (N-1)$, are driven by complementary PWM signals with a duty ratio of $D = V_{out}/V_{in}$ at an equal switching frequency f_{sw}, as in a conventional two-level synchronous buck converter. Assuming floating capacitors (also called flying capacitors) $C_{fly,j}$, where $j = 1, 2, ..., (N-2)$, are large enough to be treated as constant voltage sources during a switching period ($T_{sw} = 1/f_{sw}$), phase shifting the PWM signals [160] that drive two consecutive switch

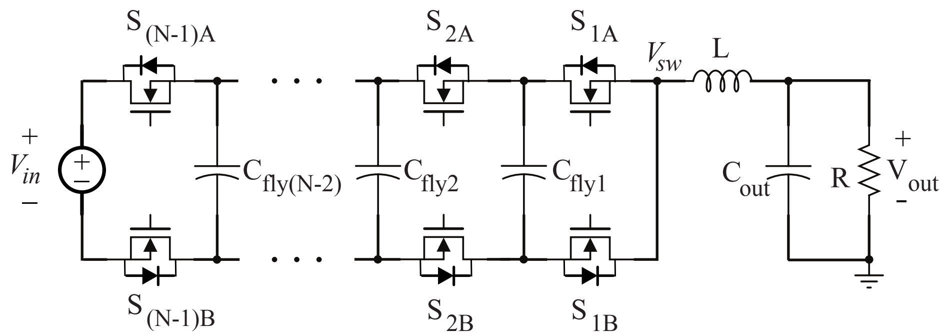


Figure 6.7: N-level FCML converter, configured as a buck converter.

pairs by $360°/(N-1)$ enforces equal charge and discharge times on the flying capacitors. In steady-state, when operated with properly phase shifted PWM signals, capacitors $C_{fly,j}$ are charged to $(1-\frac{j}{(N-1)}) \times V_{in}$, which is commonly known as the natural flying capacitor voltage balancing property of FCML converters [161]. By controlling D of individual switch pairs, N different voltage levels ($j \times V_{in}/(N-1)$, where $j = 0...(N-2)$), can be achieved at the switching node, at an effective frequency of $(N-1) \times f_{sw}$. The output switching node voltage (V_{sw}) is then filtered by filter inductor (L) and output capacitor (C_{out}) to achieve the desired output voltage ($V_{out} = D \times V_{in}$).

The key advantages of FCML converters include

- Reduced switch voltage stress: In an N-level FCML converter, each switch needs to be rated only for $V_{in}/(N-1)$. This enables the use of low voltage, high frequency, and smaller footprint switches.

- Multiple voltage levels at the switching node: Depending on D, V_{sw} can have N different levels (i.e., $k \times V_{in}/(N-1)$, where $k = 0, ..., (N-1)$). This reduces the maximum voltage magnitude that the inductor needs to filter to $V_{in}/(N-1)$.

- Increased frequency at the switching node: The effective switching frequency ($f_{sw,eff}$) observed at the switching node is $(N-1) \times f_{sw}$, where f_{sw} is the switching frequency of the individual switch pairs. This increases the chopped switching node voltage frequency that needs to be filtered without an extreme increase in switching loss of individual switch pairs.

- Reduced filter inductance: Combined with reduced voltage magnitude at the switching node, increased effective switching frequency reduces the required inductance value for desired operation of the converter.

- Capacitive energy conversion: Along with the inductor, the flying capacitors also participate in energy conversion. Since capacitors inherently have 2-3 orders of magnitude higher energy density than inductors [162], total passive component volume can be decreased, yielding improved power density.

- Heat dissipation of semiconductor switches: The total switch loss (switching loss and conduction loss) can be spread out over $2(N-1)$ switches, resulting in wider area for heat dissipation and possible improved thermal management.

- Electromagnetic interference (EMI): Although satisfying the EMI requirements in data center applications is not the focus of this work, an FCML topology generates lower EMI because of reduced dv/dt due to the multilevel waveform. The work in [155] experimentally showed that the FCML converter requires a much smaller EMI filter compared to a conventional two-level buck converter.

Several implementation and operation challenges of FCML converters should be acknowledged. These include flying capacitor voltage balancing, floating gate drives, and commutation loop inductance. Flying capacitor voltage balancing, if not achieved, results in increased voltage stress across the switches, potentially leading to overvoltage failure of the semiconductors. Therefore, flying capacitor voltage dynamics [163–166], active balancing control techniques [167–170], and circuit control methods [45, 145, 169] are thoroughly examined in the literature to better understand the properties of natural balancing and to avoid flying capacitor imbalance. In addition, [171] has shown analytically and experimentally that an FCML with an even number of levels has better natural balancing than one with an odd number of levels when the source impedance and input capacitor are considered.

Floating gate drives in an FCML converter require level shifters and isolated power converters. Supplying isolated power to many floating and isolated gate drives in an FCML converter requires inefficient and bulky circuits which may penalize efficiency and power density improvement. In [172], a well-known bootstrap circuit is modified to provide gate drive power for FCML converters in a compact, efficient, and low-cost way. Parasitic inductance (commonly known as commutation loop inductance in conventional two-level buck converters) exists in FCML converters and may result in significant switch ringing during high dv/dt switching transitions. The commutation loop at each level of an FCML converter must be minimized by careful replacement of flying capacitors and bypass capacitors in order to maintain the benefits of the converter [154, 156, 173].

In addition to implementation challenges, usage challenges of FCML converters include circuit initialization and reliability. The turn-on transient of a (especially buck-type) FCML converter, if not carefully managed, may damage the transistors due to uncertainty of flying capacitor voltages when rated input voltage is instantaneously applied to the input terminal. Such a transient can be controlled using startup circuitry similar to the inrush current limiting circuitry explained in Section 5.1.3. Due to the higher component count of an FCML converter, traditional reliability calculation approaches, in which mean time to failure of a system is dominated mainly by component count in the system, suggest deficient reliability when applied to FCML converters. Presumably, reliability calculation approaches need to be revisited to consider the relatively reduced component stresses and temperature rise, and also system-level opportunities for improved integration, for a fairer reliability analysis FCML converters. Further details of converter operation and an in-depth analysis of key advantages of FCML converters can be found in [8, 162].

The buck-type FCML converter shown in Figure 6.7 is a well-known topology. However, its use in an ac-dc conversion application where the flying capacitor voltages follow the rectified ac line voltage at 50/60 Hz in an ac-dc converter has not been explored previously, and is a key contribution of this work to the field. Note that such operation of an FCML converter is fundamentally different from a boost-type FCML converter in a PFC application, where the flying capacitor voltages are the fractions of the output voltage, which can be considered relatively constant throughout the ac line cycle when a large twice-line frequency energy buffer is present. A thorough search of the relevant literature yielded only [174–176] in which the flying capacitor voltages are subject to follow the ac input voltage at the ac line frequency. In [174–176], a four-level FCML buck converter is employed in an ac-ac application to step down the line voltage by using a fixed duty ratio during the entire line cycle. In this dissertation, an FCML buck converter is tasked to achieve PFC in an ac-dc operation, which results in a unique operating condition as the duty ratio changes at the line frequency. The investigation of flying capacitor voltages under this unique condition is a focus of this dissertation; thus, non-unity power factor operation, where the FCML buck PFC converter disables when the input voltage is below the output voltage, is explored.

CHAPTER 7

BUCK PFC CONTROL

Although an FCML buck converter increases power density in comparison to conventional two-level designs, it also presents challenges in PFC control. In this chapter, existing PFC control techniques for conventional buck converters are summarized, and a new PFC control methodology that is applicable to both conventional two-level and FCML buck topologies is proposed.

7.1 Background

PFC control of the buck converter has been thoroughly analyzed in the literature and several different control methods have been proposed. Most of the existing work in the literature uses a full-bridge rectifier followed by an asynchronous buck converter, in order to simply disable the converter by opening the high side switch when the input voltage is lower than the output voltage. In [134], an asynchronous buck converter performs power factor correction in discontinuous conduction mode by keeping the duty ratio constant throughout the entire ac line cycle. In [177, 178], it is shown that by adding an LC filter to the input of a buck converter, the input capacitor voltage can be forced into discontinuous mode, and a control algorithm to achieve PFC is proposed. Discontinuous input voltage and inductor current mode operations are also combined to achieve PFC in a 1 kW asynchronous buck converter [179]. An averaged small-signal model of an asynchronous buck converter is derived as a function of its output impedance and used in PFC control in [180]. In [137], a universal-input, 80 V output, 94 W buck converter achieved power factor up to 0.96 over its the entire operating range by using a clamped current control methodology that relies on switch current measurement. Critical conduction mode based on inductor current is also leveraged to perform PFC, and constant [181] and variable [182] on time control methods that use inductor current measurement are applied to asynchronous buck converters for notebook charger and LED applications.

As mentioned in Chapter 6, a cascaded buck-boost converter can also be used in PFC applications to increase achievable power factor compared to a conventional buck converter [139]. In the cascaded buck-boost converter, instantaneous input and output voltage determine whether the buck stage or boost stage is modulated, and the duty ratios required to control both stages are calculated separately. In [183], the buck stage is controlled by limiting inductor current within a band to perform PFC control. A current programmed control that offers inrush and overcurrent protection is proposed in [140] and power factor correction is performed without changing the current reference as the converter transitions between the buck and the boost stage. Discontinuous input voltage mode [143], and boundary or critical conduction mode of the inductor current [184], also have been investigated in the cascaded buck-boost topology used as a PFC converter. Recently, a hybrid feedforward control offered seamless transition between its buck and boost stages while performing power factor correction in a cascaded buck-boost converter [57].

Existing PFC control methodologies for buck and cascaded buck-boost converters are not directly applicable to an FCML buck topology because it employs synchronous switch pairs that are controlled with complementary phase shifted PWM signals as explained in Chapter 6. In phase-shifted PWM control, natural balancing of flying capacitor voltages relies on all flying capacitors charging and discharging for the same duration in each switching period. This makes cycle by cycle duty ratio adjustments to limit inductor current within a band at the switching frequency challenging. Moreover, flying capacitor voltages in a FCML topology for PFC are expected to follow the input voltage at 50/60 Hz; therefore, they should not be discontinuous. In this work, a feedforward control, combined with a high bandwidth inner current loop and a slower bandwidth outer voltage loop are used to generate the duty ratio, which is kept constant during each switching period to achieve natural balancing of flying capacitors with phase shifted PWM signals. The rest of this chapter explains the details of the proposed control approach.

7.2 Overview of the proposed control algorithm

In order to focus on the PFC task in the development of the proposed control algorithm, the FCML buck converter is assumed to achieve natural balancing of the flying capacitor voltages, and twice-line frequency energy buffering is assumed to be handled by a capacitor bank at the converter output. Active voltage balancing of the flying capacitors and advanced twice-line frequency energy buffering techniques can be incorporated later if needed. The proposed control algorithm is applicable to any number of levels, including conventional (i.e., two-level) buck converter. The

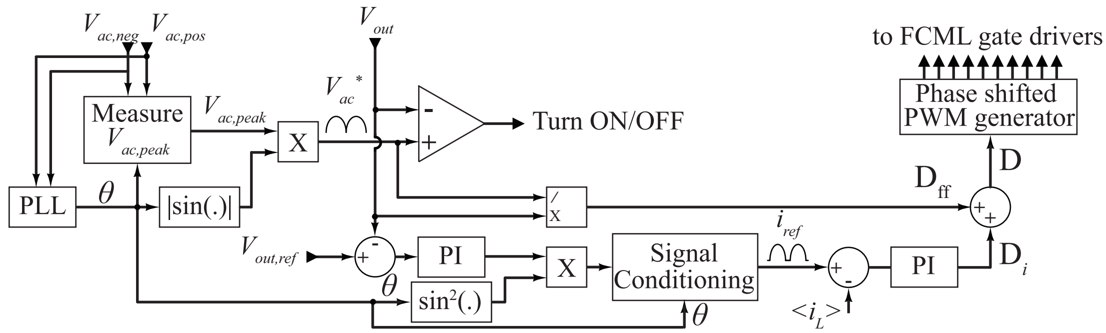


Figure 7.1: High-level control diagram.

control actions are executed as they were for a conventional buck converter, and additional PWM signals for the remaining FCML converter switches are properly phase shifted to achieve natural balancing of the flying capacitor voltages.

A high-level control diagram of the proposed PFC algorithm for the FCML buck converter is illustrated in Figure 7.1. The algorithm is implemented on a 32-bit floating-point microcontroller with a 200 MHz system clock. The 12-bit ADC submodules of the microcontroller are used to sample the input voltage, output voltage, and output (or, average inductor) current, shown as $V_{ac,pos}$, $V_{ac,neg}$, V_{out}, and $< i_L >$ in Figure 7.1, respectively. The control signal (i.e., duty ratio), shown as D in Figure 7.1, is sent to the FCML gate drives through the PWM peripherals of the microcontroller. The ADC and control signal calculation are executed at a sampling frequency matched to the switching frequency. The microcontroller has a trigonometric math unit (TMU) which is used to construct signal equivalents of the input voltage and reference current. Note that by neglecting the phase shifted PWM block, the proposed algorithm can also be applied to a conventional two-level buck converter.

The proposed control algorithm comprises a feedforward term (D_{ff} in Figure 7.1) which provides the ideal duty ratio given the converter operating point, and a multiloop control term (D_i in Figure 7.1) which compensates the nonidealities which are not governed within the feedforward control. The multiloop control consists of a higher bandwidth inner current loop which tracks a desired reference current and a slower bandwidth voltage loop which provides an amplitude for the reference current to achieve PFC. Other supporting functions of the proposed control algorithm include a phase-locked loop to synchronize the converter with the ac input voltage, and a comparator to determine whether the PFC algorithm and the converter must be enabled or disabled.

7.2.1 PLL

A phase-locked loop (PLL) based adaptive notch filter is used to synchronize the converter with the ac input voltage. As shown in Figure 7.1, the PLL control block uses $V_{ac,pos}$ and $V_{ac,neg}$ to extract the phase angle of the input ac voltage (denoted as θ in Figure 7.1). Once the converter is locked to the ac input voltage, the peak value of the input voltage can be calculated to adapt the control algorithm for universal input voltage. The peak value of the input voltage and the phase angle are also used to construct a distortion- and noise-free replica of the input voltage (denoted as V_{ac}^{*} in Figure 7.1) with the help of the TMU.

7.2.2 Comparator

As stated in Chapter 6, an FCML buck converter is unable to perform PFC and deliver power when the input voltage is less than the output voltage. Therefore, a comparator block is needed to compare the replica of the input voltage (V_{ac}^{*}) to the measured output voltage (V_{out}) at every sampling period to determine whether the multiloop and feedforward control laws are executed or the converter is disabled. Here, using the replica of the input voltage (V_{ac}^{*}) instead of the actual input voltage measurement prevents the converter from having a turn-on/off oscillation after it is enabled at every ac half-cycle. In practice, depending on the output current, the sampled input voltage may exhibit a small decline when the converter is enabled. Such a decline, combined with measurement noise, may cause a lower input voltage measurement than the output voltage measurement, and may disable the converter in the next sampling period.

7.2.3 Reference current

A reference signal is needed as an input to the current loop portion of the PFC controller so that the input current is in phase and sinusoidal when the input voltage is higher than the output voltage. As mentioned before, in this work, the high bandwidth inner current loop tracks the average inductor current, which is unlike boost-type PFC converters, at the converter output. Since the input current must be as sinusoidal as possible to achieve good power factor, power flow analysis is needed to identify a reference current for the average inductor current.

Figure 7.2 shows an ac-dc power converter that is connected between a single-phase ac power source and a dc load that includes a twice-line frequency buffering element. In single-phase ac-

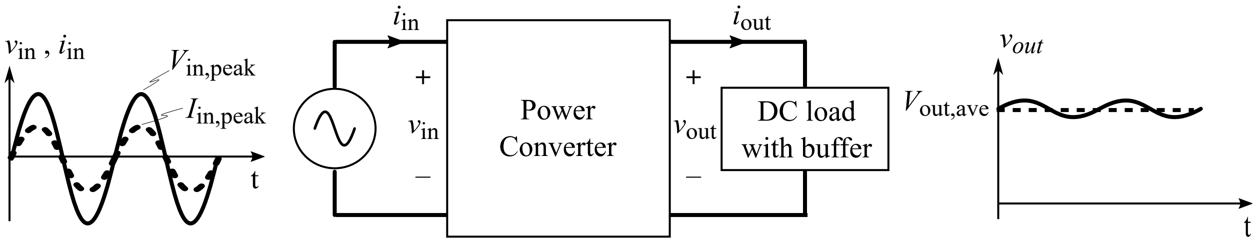


Figure 7.2: A generic ac-dc power converter connected between a single-phase ac input source and a dc load that includes a twice-line frequency energy buffering element.

dc power conversion, unity power factor occurs when the input voltage (v_{in} in Figure 7.2) and current (i_{in} in Figure 7.2) are both sinusoidal and in phase (i.e., $v_{in}(t) = V_{in,peak}sin(2\pi f_{grid}t)$ and $i_{in}(t) = I_{in,peak}sin(2\pi f_{grid}t)$). The instantaneous input power is given by

$$P_{in}(t) = v_{in}(t)i_{in}(t) = V_{in,peak}sin(2\pi f_{grid}t)I_{in,peak}sin(2\pi f_{grid}t). \tag{7.1}$$

In this analysis, it is assumed that the power converter also provides twice-line frequency power buffering, through active or passive means. Under this assumption, although some voltage ripple still exists at the output, its amplitude will be much smaller than the average value of the output voltage. Thus, the output voltage is assumed to be constant in this analysis and $v_{out}(t) \approx V_{out,ave}$. The instantaneous output power is given by

$$P_{out}(t) = v_{out}(t)i_{out}(t) \approx V_{out,ave}i_{out}(t). \tag{7.2}$$

Further assume that the generic ac-dc power converter in Figure 7.2 is ideal. By equating (7.1) and (7.2), a mathematical relationship for the instantaneous output current can be obtained as

$$i_{out}(t) = \frac{v_{in}(t)i_{in}(t)}{v_{out}(t)} = \frac{V_{in,peak}I_{in,peak}}{V_{out,ave}}sin^2(2\pi f_{grid}t). \tag{7.3}$$

Equation (7.3) means that sinusoidal and in-phase input current requires the output current to be proportional to a sine squared waveform that is in phase with the input voltage. In order to compensate for losses that are ignored by equating (7.1) and (7.2), and also to achieve output voltage regulation, a proportionality constant K can be determined by the outer voltage loop. In conclusion, in order to achieve high power factor by tracking the output current (i.e., the average inductor current in a buck converter), the inner current loop reference is given by

$$i_{ref}(t) = \begin{cases} K sin^2(2\pi f_{grid} t), & \text{if } v_{in}(t) > v_{out}(t) \\ 0, & \text{otherwise.} \end{cases} \tag{7.4}$$

Once the PLL provides the angle of the input voltage, the TMU of the microcontroller can calculate the sine squared term in (7.4).

7.2.4 Feedforward control

Feedforward control is an effective method to improve control performance by reducing the effects of disturbances in PFC applications, and is often preferred in boost-type PFC converters [141,185]. Here, feedforward is used to estimate the ideal duty ratio by using circuit equations that govern converter behavior. As mentioned before, the control algorithm is developed by approximating FCML buck converter dynamics with conventional buck converter dynamics. Thus, in order for the inductor current to follow the reference current derived above, the following first-order equation must be satisfied:

$$L\frac{\mathrm{d}i_L}{\mathrm{d}t} = v_{in}D - v_{out}. \tag{7.5}$$

Given the reference current i_{ref} and target output voltage $V_{out,ref}$, (7.5) can be rewritten as

$$L\frac{\mathrm{d}i_{ref}}{\mathrm{d}t} = v_{in}D - V_{out,ref}. \tag{7.6}$$

In order to obtain a feedforward term, (7.6) can be reorganized as

$$D = \frac{L}{v_{in}}\frac{\mathrm{d}i_{ref}}{\mathrm{d}t} + \frac{V_{out,ref}}{v_{in}}, \tag{7.7}$$

which can be implemented in a digital controller as

$$D_{ff} = \frac{L}{v_{in}}\frac{\Delta i_{ref}}{\Delta t} + \frac{V_{out,ref}}{v_{in}}. \tag{7.8}$$

Under ideal conditions, (7.8) should suffice to achieve PFC control. However, sensitivities and uncertainties in the converter itself and sensing hardware require multiloop feedback control to track the reference current and voltage. Nevertheless, the feedforward term D_{ff} given by (7.8) can be calculated by the microcontroller since all but the inductance value (L) information is available through the microcontroller internal states and sensing peripherals. Δi_{ref} can be calculated by storing the reference current value from the previous sampling time, Δt is the sampling period, a signal replica of v_{in} is constructed by using the PLL block, and $V_{out,ref}$ is given as a reference voltage to the multiloop control to achieve load regulation. The true value of L depends on inductor current, temperature, and various other operating parameters which change with operating point. This is another reason to include a multiloop feedback loop to track the reference current and voltage.

In [141], a feedforward term similar to (7.7) is derived for boost PFC converters and called "complete feedforward". As mentioned before, one key feature of an FCML buck converter is a reduced inductor requirement for a given switching frequency. In "complete feedforward" for the buck PFC converter given by (7.7), the impact of the derivative term is attenuated as the inductance gets smaller, resulting in poor reference current tracking performance. Since multiloop feedback control is employed in this work to compensate for uncertainties in the system, the derivative term may be omitted from (7.7), which simplifies (7.8) to

$$D_{ff} = \frac{V_{out,ref}}{v_{in}}. \tag{7.9}$$

In [141], a feedforward term that is similar to (7.9) is also derived for boost PFC converters and called "partial feedforward". This work uses (7.9) to calculate the feedforward term, and relies on multiloop feedback control to track the reference current and voltage. Note that in the actual implementation of (7.9), the signal replica of the input voltage (v_{in}) is used.

7.2.5 Multiloop feedback control

Multiloop control of buck converters, which consists of a fast inner current loop and a slower voltage loop, is well-known [186]. Both the inner loop and the outer loop employ proportional and integral (also known as PI, or lag, or type 1) compensation, and are tuned by using a converter small-signal model. In multiloop feedback control of a converter, the outer loop uses the reference voltage and measured output voltage to provide a reference current for the inner loop. The inner loop uses the reference current and measured average inductor current to track the reference current.

Multiloop feedback control is used in the proposed PFC control. The dynamic behavior of the

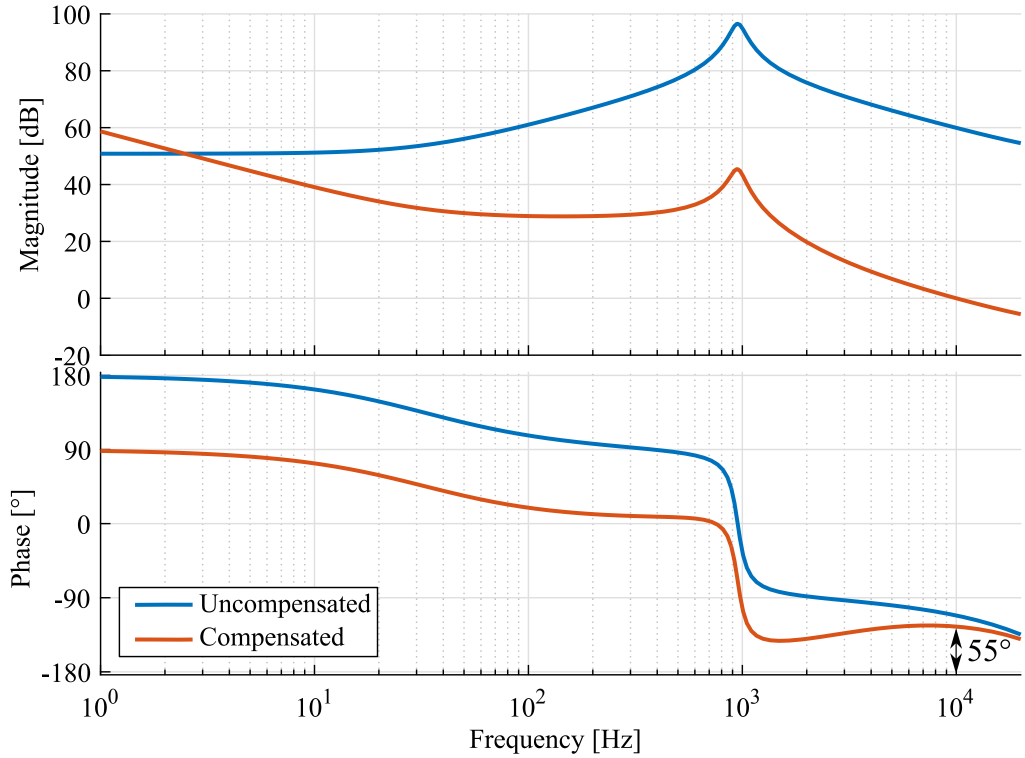


Figure 7.3: The uncompensated and PI compensated loop gain of $(G_{id}(z))$.

FCML buck converter is approximated with the dynamic behavior of a conventional buck converter. (This approximation is experimentally validated by comparing the frequency response of a six-level FCML to a conventional (two-level) buck converter. The comparison results can be found in Appendix C.) Thus, a buck converter small signal model is used to tune the PI compensator parameters. Although the small signal model is not completely appropriate for large signal (i.e., PFC) operation, in this work the feedforward term (i.e., (7.9)) brings the converter near an ideal operating point by providing the expected conversion ratio between input and output. Multiloop feedback control which is tuned by using a small-signal model only compensates for nonidealities and uncertainties around the operating point provided by the feedforward term.

Since the proposed control algorithm is implemented using digital control, discrete time modeling of the buck converter and direct-digital design of the PI compensator are preferred in this work. Interested readers can refer to [186] for complete details of discrete time modeling of a synchronous buck converter (Section 3.2.1 in [186]), and multiloop feedback control compensator design of a synchronous buck converter (Section 4.2.3 in [186]).

Converter specifications for digital modeling and compensator design are given in Table 7.1. The PI compensator for the inner current loop is designed for the discrete time duty ratio to inductor

Table 7.1: Parameters used for multiloop feedback control compensator tuning

Variable	Value
Input Voltage	340 V ($\approx$ 240 V_{RMS})
Output Voltage	48 V
Output Power	400 W
Switching Frequency	80 kHz
Filter Inductor	5.6 μH
Output Capacitor	5 mF

current transfer function ($G_{id}(z)$) of a conventional buck converter. The PI compensator for the inner current loop is designed to have 55° phase margin at 10 kHz crossover frequency (one eighth of the switching frequency). The uncompensated and PI compensated loop gain of ($G_{id}(z)$) are plotted in Figure 7.3. The PI compensator for the outer voltage loop is evaluated with the inner current loop closed. Separating the cut-off frequencies of the inner and outer loops by three to four orders of magnitude is common practice. Therefore, the PI compensator for the outer voltage loop is designed at 10 Hz crossover frequency.

CHAPTER 8

EXPERIMENTAL STUDY OF AN FCML BUCK PFC CONVERTER

Numerous ac-dc and dc-dc experiments were performed to validate performance of the FCML buck converter and control. This chapter describes the experimental studies performed and discusses the results. During the experiments, various converter parameters and operation points were tested. Although not repeatedly underlined throughout the experimental study, the control parameters in the proposed PFC control algorithm are tuned accordingly as converter parameters and operating points change.

8.1 Prototype FCML buck converter for single-phase ac-dc power conversion

The hardware prototype used in this work consists of two main power stages: an active rectifier, and an N-level buck converter. The active rectifier stage is a straightforward implementation with four low $R_{DS(on)}$ MOSFETs that can withstand the ac line voltage, and associated gate drive circuitry. However, the design space for the N-level buck converter is quite broad. A Monte Carlo optimization was pursued to provide good balance between power density and efficiency. Interested readers can refer to [154] for an example of the Monte Carlo optimization method and the loss models used in this work.

As mentioned before, the FCML buck converter needs to operate over universal input voltage and 48 V output voltage. The highest voltage that the converter must withstand occurs at 240 V_{RMS} input voltage, which is the operation condition considered when designing the hardware prototype. The number of levels in the FCML buck converter is chosen as six, not only because of prior work [171] which demonstrated that an even number of levels has better natural balancing of the flying capacitor voltages, but also because a six-level design yields 68 V maximum voltage stress at the ac line peak, enabling the use of 100 V semiconductor switches with adequate margin. Considering the high output current requirements of buck-type PFC converters, GaN transistors

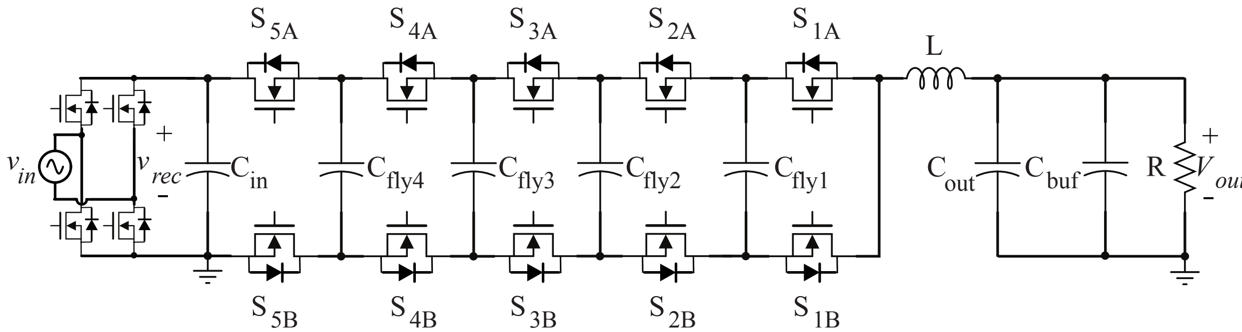


Figure 8.1: The six-level FCML buck converter schematic for a single-phase PFC application.

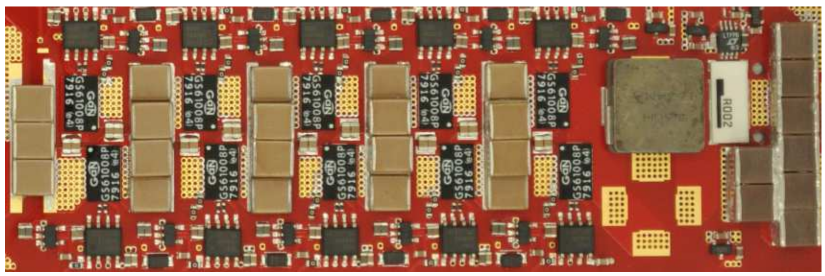


Figure 8.2: A close-up photograph of the FCML buck stage of the hardware prototype. (Actual size.)

are preferred to achieve high power density and low conduction loss despite the dynamic on-state resistance phenomena in present power GaN transistors [187]. The filter inductor and flying capacitor values are chosen to allow 8.5 A average output current at 80 kHz switching frequency, yielding approximately 400 W maximum output power. Gate drive circuitry for the floating transistors in the FCML buck converter is energized using a cascaded bootstrap scheme [172]. The key components used in the multilevel hardware prototype are listed in Table 8.1. The schematic is given in Figure 8.1.

Care must be taken in converter layout to maximize power density while maintaining a reasonable thermal profile and low commutation loop inductance. Proper heat sink placement on top-cooled devices is often challenging due to uneven component heights. In this work, GaN transistors from GaN Systems, which offer bottom cooled devices, are preferred, and all components are placed on the top side of a four-layer printed circuit board (PCB). Minimizing the commutation loop inductance in the FCML buck converter layout is crucial. As in a two-level buck converter, commutation loop parasitic inductance exists between complementary switch pairs and flying capacitors in an FCML converter. In order to minimize this parasitic inductance, small footprint decoupling capacitors are connected in parallel with larger footprint higher capacitance flying capacitors, but

Table 8.1: Key components of the FCML buck-type PFC hardware prototype

Stage	Component	Manufacturer & Part Number	Details
Active rectifier	Transistors	ST Microelectronics STL57N65M5	650 V, 61 mΩ
	Gate driver	Fairchild Semiconductor FAN73932	
	Bootstrap diode	Rohm Semiconductor RFN1L6S	
	Digital isolator	Silicon Labs SI8423	Optional
FCML buck	Transistors	GaN Systems GS61008P	100 V, 7 mΩ
	Gate driver	Silicon Labs SI8271GB-IS	Isolated, single channel
	Flying capacitor	TDK C5750X6S	450 V, 2.2 μF, 6 in parallel per level
	Decoupling capacitor	TDK C2012X7T	450 V, 47 nF, 4 in parallel per level
	Inductor	Vishay ILHP5050EZER	5.6 μH
	Output capacitor	TDK CGA9N3X7S	100 V, 10 μF, 16 in parallel
	Input capacitor	TDK C5750X6S	450 V, 2.2 μF, 3 in parallel
Cascaded bootstrap	Bootstrap diode	Vishay Semiconductors VS-2EFH02HM3	0.75 V, 2 A
	Bootstrap capacitor	TDK C1608X5R	25 V, 10 μF
	LDO	Texas Instruments LP2985	6.1 V
Digita controller	Microcontroller	Texas Instruments F28377D	
	Level shifters	Texas Instruments SN74LV4T125PWR	
Current sensing	Differential amplifier	Linear Technology LT1999	50 V/V
	Voltage reference	Texas Instruments REF3012	1.2 V
	Sense resistor	Ohmite FC4L	2 mΩ
Logic voltage	LDO	Microchip Technology MCP1700	5 V

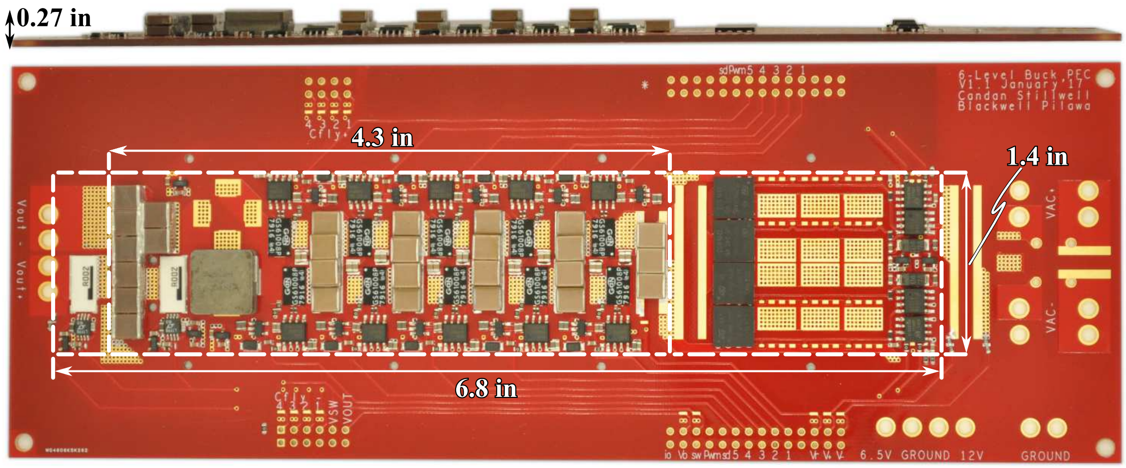


Figure 8.3: Top and side views of the hardware prototype.

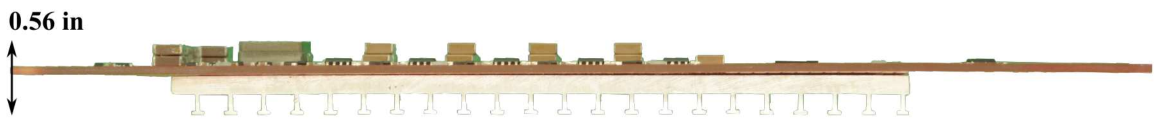


Figure 8.4: Side view of the hardware prototype with attached heat sink.

are placed close to the floating switch pairs in order to achieve the smallest possible commutation loop area. Similar component placement to minimize commutation loop inductance in an FCML converter design can be found in [153–156, 173]. To minimize the hardware prototype box volume, a low profile inductor is used to match the height of the flying capacitors that are stacked in two rows. The FCML buck stage alone, shown in Figure 8.2, has a box volume of 1.63 in^3. The hardware prototype, side and top views of which are shown in Figure 8.3, fits in a box of volume of 2.57 in^3. Experimental results which are provided in this work are achieved with a 0.29 in tall heat sink, placed on the bottom of the PCB. A side view of the converter with the heat sink attached is provided in Figure 8.4. The box volume of the hardware prototype with the heat sink is 5.33 in^3, yielding a 75 W/ in^3 power density. Note that the heat sink is not optimized for this converter or for its heat transfer requirements. The PCB layouts of the prototype FCML buck converter are provided in Appendix D.

A microcontroller that can output phase shifted PWM signals at reasonable frequency and resolution is preferred to control the hardware prototype. 12-bit ADC submodules of this microcontroller sample the voltages through resistive dividers. For current measurement, two high bandwidth dif-

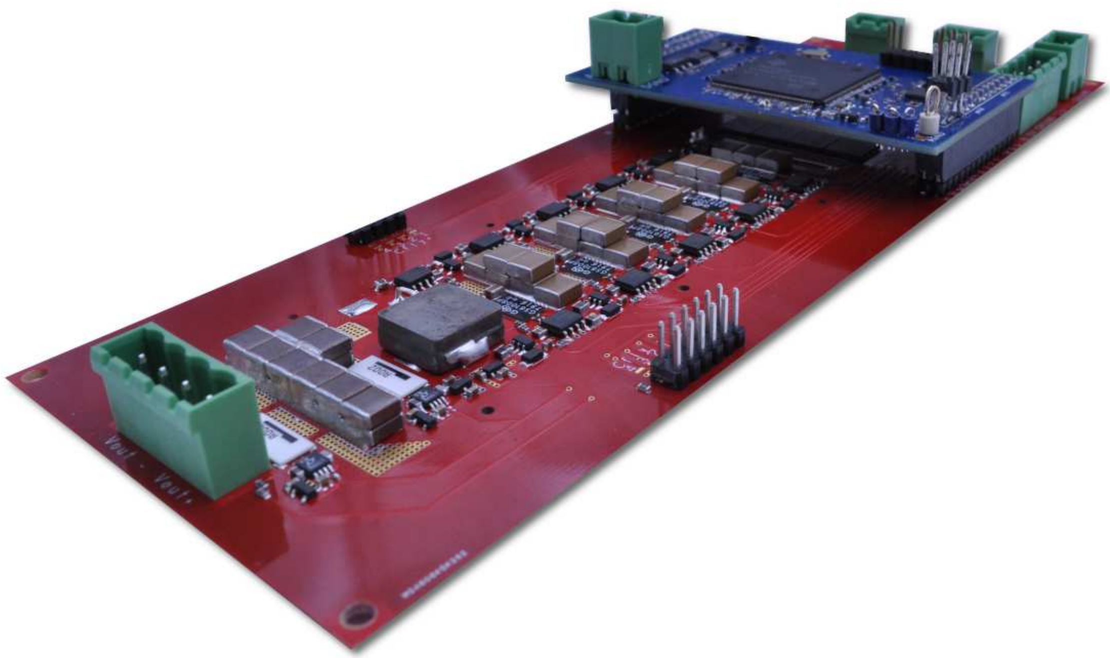


Figure 8.5: Hardware prototype with attached microcontroller.

ferential amplifiers with shunt resistors are used to measure both inductor and output current. Since the output voltage is less than 60 V, the current measurement can be done on the high side without violating common mode voltage capability of commercially available differential amplifiers. The hardware prototype with the microcontroller connected can be seen in Figure 8.5.

As mentioned before, this work focuses on the PFC front end converter and twice-line frequency energy buffering is assumed to be provided separately by the UPS. Therefore, for the experimental work, a large electrolytic capacitor bank (annotated as C_{buf} in Figure 8.1) on the order of millifarads is added to the converter output to mimic 48 V UPS behavior. The large capacitor bank is not part of the PFC front end for purposes of power density calculation.

8.2 Experimental setup

The experimental setup consisted of a programmable ac voltage source, Pacific Power Source 112-AMX; a programmable dc voltage source, Keysight N8937A; two programmable electronic loads, one Agilent 6060B and one Chroma 63803; an ac power analyzer, Keysight PA2201A; a power meter, Yokogawa WT3000; and various high power resistors. Two dedicated dc power supplies, HP 6632A, provide auxiliary power to the hardware prototype at 6.5 V and 12 V.

Code Composer Studio from Texas Instruments was used to program and debug the microcontroller. A JTAG emulator, Spectrum Digital XDS100V2, established communication between the computer and the microcontroller. When running the converter, a SeaISO USB isolator, Sealevel ISO-1-OEM, isolated the computer ground from the converter ground.

Converter waveforms were captured with a mixed signal oscilloscope, Tektronix MSO4034. The ac power analyzer was used to capture input and output waveforms and measure the input and output values such as power factor and efficiency in ac-dc operation. The power meter is used to measure efficiency in dc-dc operation. Both the ac power analyzer and the power meter have 0.05% measurement accuracy. The input voltage and flying capacitor voltages were measured with a data acquisition unit (National Instruments PXIe-1078 chassis, PXIe-4300 16-bit analog input module, and PXIe-4300B terminal block) at 25000 samples per second to study voltage balancing. Note that this sampling frequency does not capture switching frequency ripple on the flying capacitor voltages. Thermal images were captured with FLIR T420.

In addition to the hardware prototype described in Section 8.1, two 15 μH inductors and one 1 Ω high power resistor were added in series between the source and the hardware prototype during the experimental work to set the source impedance and to aid FCML converter startup.

The microcontroller code used in this experimental study is provided in Appendix E.

8.3 Six-level FCML buck converter in dc-dc operation

The six-level FCML buck converter was operated as a dc-dc converter to test its performance at various operating points over the rectified ac cycle. The input voltage, duty ratio and the output current of the converter were manually adjusted to create the operating points that the converter will run during ac-dc conversion. The converter efficiency at selected operating points is given in Figure 8.6.

Flying capacitor voltage balancing and heat dissipation performance of the hardware prototype were also tested in dc-dc operation. The six-level FCML buck converter achieves excellent natural voltage balancing and thermal profile in dc-dc operation. The switch node voltage and the ac coupled inductor current at the peak input voltage and inductor current can be seen in Figure 8.7(a). Also, note that at this dc-dc operating point, the FCML buck stage (shown in Figure 8.2) provides 16 A at 48 V from 340 V input, yielding 228 W/in^3 power density including the heat sink. A thermal

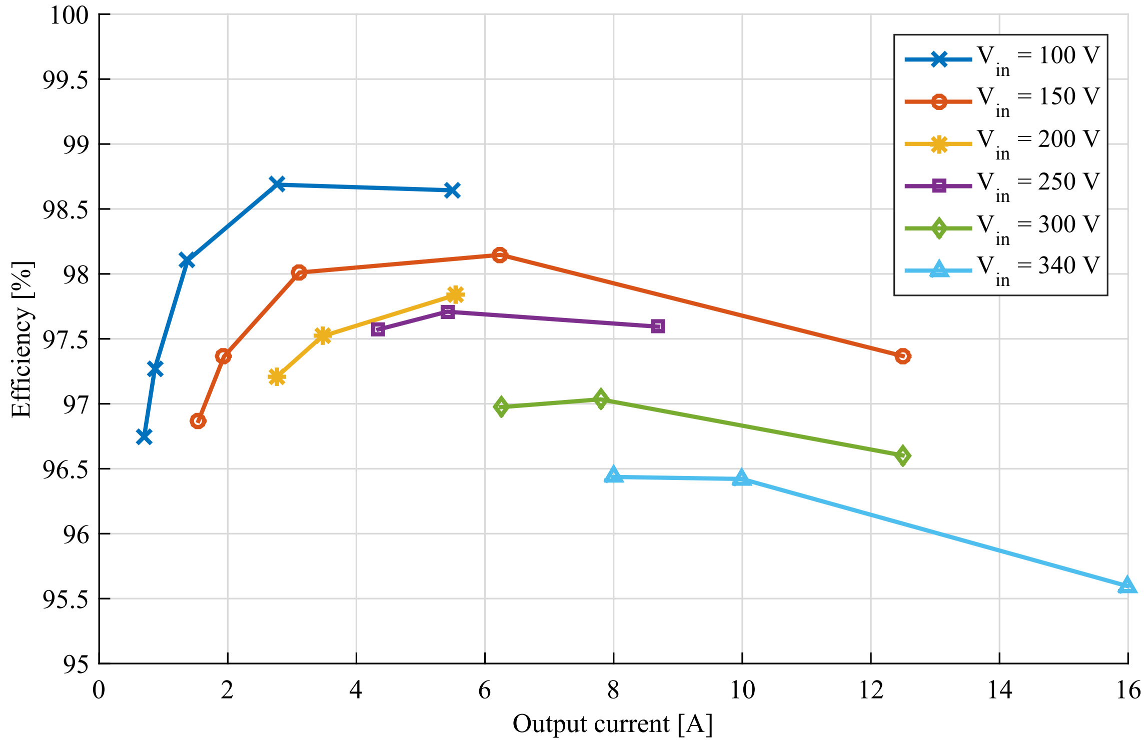
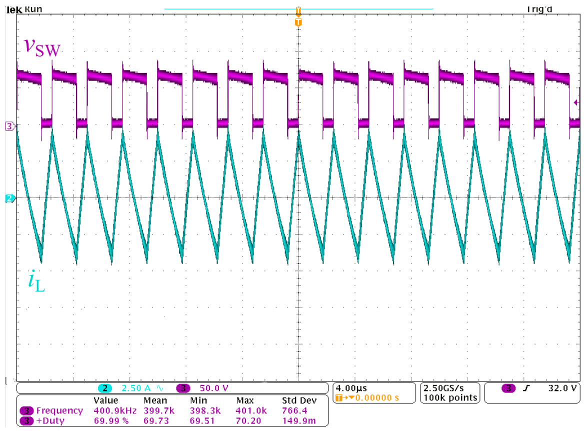


Figure 8.6: Efficiency of the FCML buck converter in dc-dc operation.

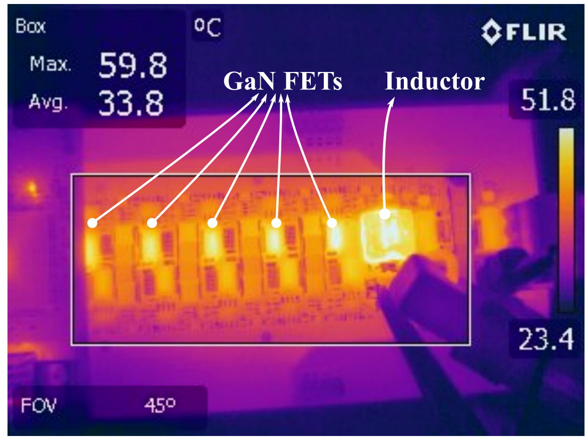
image of the converter for this dc-dc operating point is provided in Figure 8.7(b). Additional experimental results that demonstrate dc-dc operation of the six-level FCML converter can be found in Appendix F.1.

8.4 Verification of the proposed PFC control on a conventional (two-level) buck converter

The proposed PFC control algorithm was first applied to a conventional two-level buck converter in order to validate the combined feedforward and multiloop feedback control approach. The hardware prototype shown in Figure 8.3 was configured as a two-level buck converter using the same transistors, gate drives and analog sensing circuitry. Since each transistor is rated for 100 V, the input and output voltage were scaled down by four times, to preserve the current conduction angle for PFC operation. The output power was also scaled down to 50 W since this experiment focused on control validation. The filter inductor was replaced with an off-board 50 μH inductor to approximately match the current ripple in a six-level FCML converter. The output capacitor was adjusted both to have enough twice-line frequency energy buffering at the lower output voltage,



(a) Switch node voltage and ac coupled inductor current.



(b) Thermal image of the dc-dc stage

Figure 8.7: Six-level FCML buck converter dc-dc operation at $V_{in} = 340$ V, $V_{out} = 48$ V and $I_{out} = 16$ A.

Table 8.2: Updated components and specifications for the two-level configuration of the hardware prototype

Specification	New Value
Input Voltage	60 V_{RMS}
Output Voltage	12 V
Output Power	50 W
Component	**New Value**
Filter Inductor	50 μH
Output Capacitor	10 mF

and to provide the requisite load energy when the input voltage is less than the output voltage. The updated specifications and parts are summarized in Table 8.2. The proposed control algorithm was tuned for the updated specifications in Table 8.2.

The input voltage and current, and inductor current are given in Figure 8.8. This operation point achieved 0.9880 power factor. The PFC performance observed in Figure 8.8 validated the feedforward and multiloop feedback control approach.

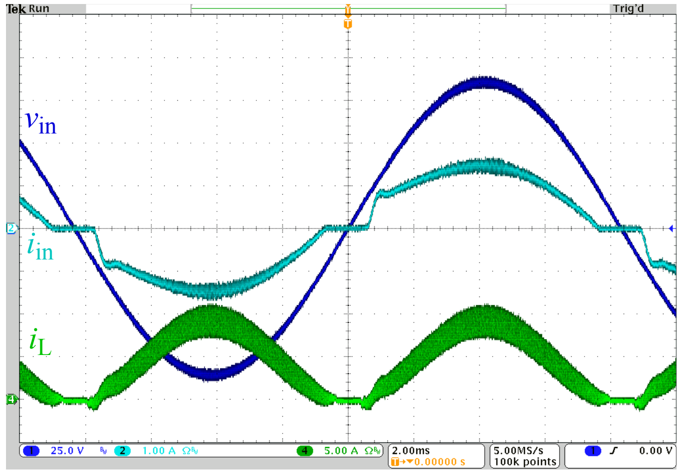


Figure 8.8: Input voltage and current, and inductor current, of the conventional (two-level) buck converter for PFC operation. $V_{in} = 60\ \mathrm{V_{RMS}}$, $V_{out} = 12$ V and $I_{out,ave} = 4.5$ A.

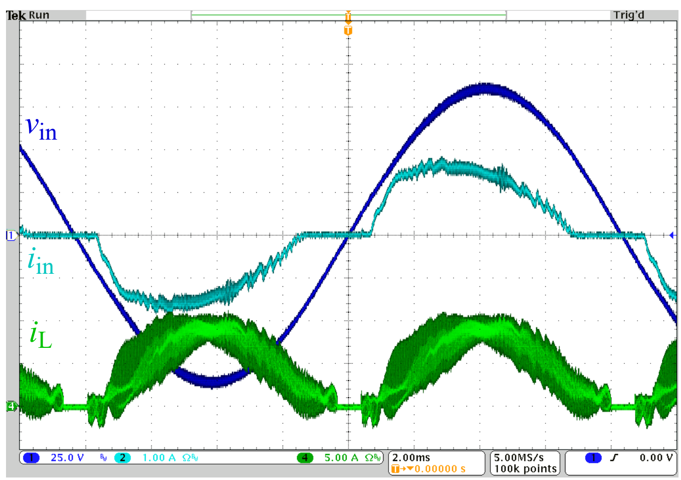
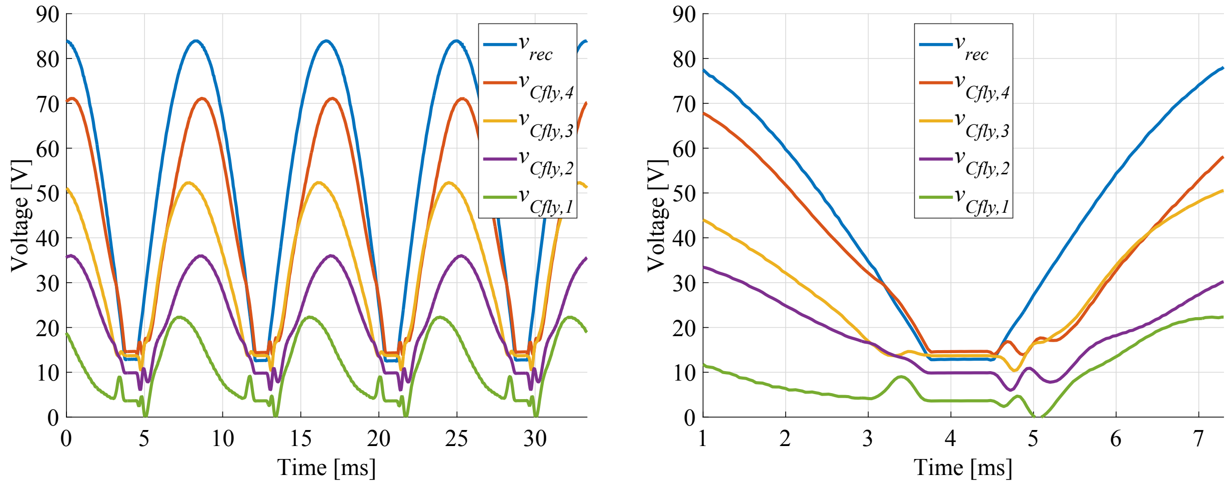


Figure 8.9: Input voltage and current, and inductor current, of the six-level buck converter for PFC operation. $V_{in} = 60\ \mathrm{V_{RMS}}$, $V_{out} = 12$ V and $I_{out,ave} = 4.5$ A.

8.5 Verification of the proposed PFC control on a six-level FCML buck converter

Next, the proposed PFC control algorithm was applied to the six-level FCML buck converter prototype.

Initially, in order to see the natural balancing of flying capacitor voltages with a 60 Hz ac input, the input and output voltage were scaled down by four times, to preserve the current conduction angle for PFC operation, and output power was set to 50 W. All components of the hardware prototype are as listed in Table 8.1, except that the electrolytic output capacitance is 10 mF. This assures enough twice-line frequency energy buffering at the lower output voltage and requisite load



(a) For two ac line cycle

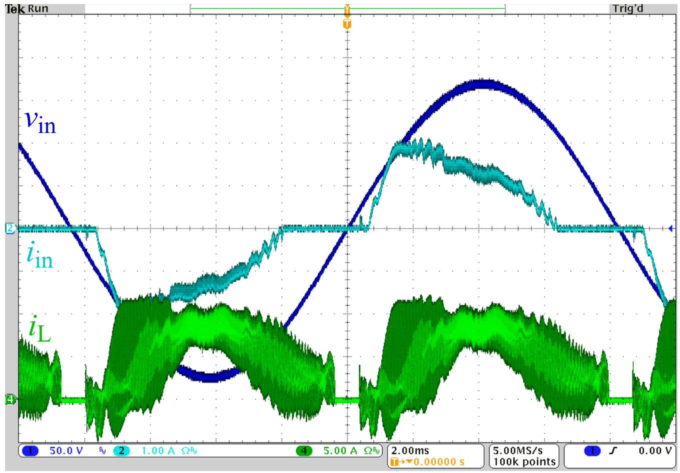
(b) During converter turn-on and -off

Figure 8.10: Flying capacitor voltages at $V_{in} = 60$ $\mathrm{V_{RMS}}$, $V_{out} = 12$ V and $I_{out,ave} = 4.5$ A.

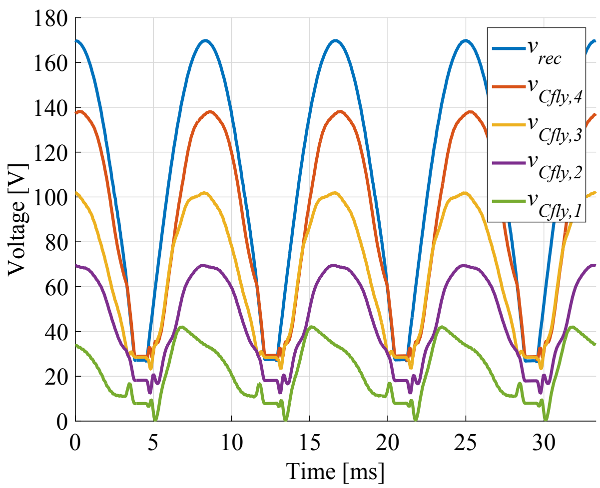
energy for 50 W output power during the time interval when the input voltage is less than the output voltage. The PI compensator parameters were updated by targeting the same 55° phase margin at 10 kHz to reflect changes to the hardware and operating point.

The input voltage and current, and inductor current, are given in Figure 8.9. The flying capacitor voltages are given in Figure 8.10 for two full ac line cycles and during converter turn-on and -off. This operating point achieved 0.9324 power factor.

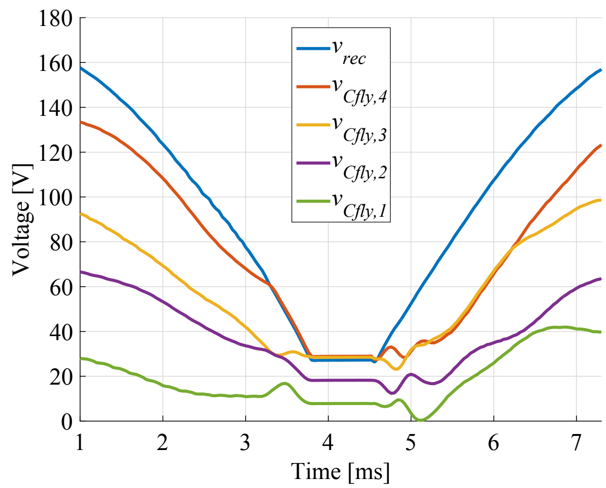
As shown in Figure 8.10(a), where the flying capacitor voltages are given for two full ac line cycles, they are not close to their expected values which are specific fractions of the rectified input voltage. At the beginning and end of the conduction angle, where the rectified input voltage has the highest $\frac{\mathrm{d}v}{\mathrm{d}t}$ and the duty ratio change is the fastest across the complete conduction angle, flying capacitor voltage balance was not maintained, as can be seen in Figure 8.10(b). In addition, the poor balance at the end of the conduction angle results in flying capacitor voltages at uncontrolled levels just before the converter was disabled. This resulted in a nonideal initial condition for the flying capacitor voltages at the beginning of the conduction angle in the next ac line cycle. Thus, the flying capacitor voltages oscillated after the converter was enabled. Nevertheless, voltage imbalances on flying capacitor voltages at the beginning and end of the conduction angle do not violate the switch voltage ratings even at rated input voltage. For safe circuit operation (i.e., in order not to violate the voltage rating of the transistors), the flying capacitor voltages must be well-balanced at the input voltage peak. However, flying capacitor voltage imbalance led to additional current harmonics on the inductor current, which ultimately translated to the input current shape



(a) Input voltage and current, and inductor current.



(b) Flying capacitor voltages for two ac line cycle



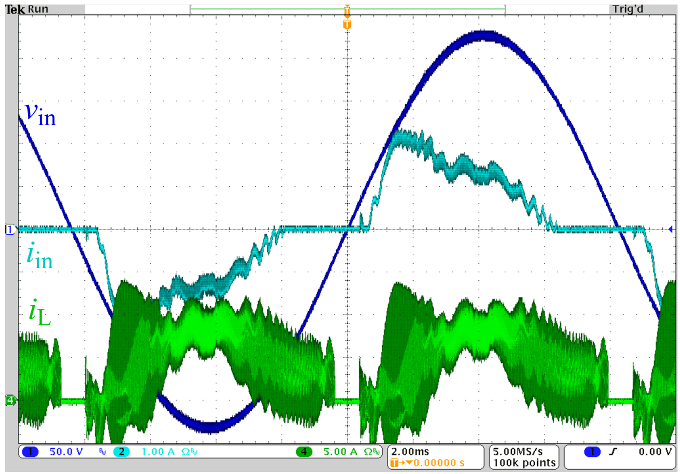
(c) Flying capacitor voltages during turn-on and -off

Figure 8.11: Six-level buck converter for PFC operation at $V_{in} = 120\ \text{V}_{\text{RMS}}$, $V_{out} = 24$ V and $I_{out,ave} = 4.5$ A.

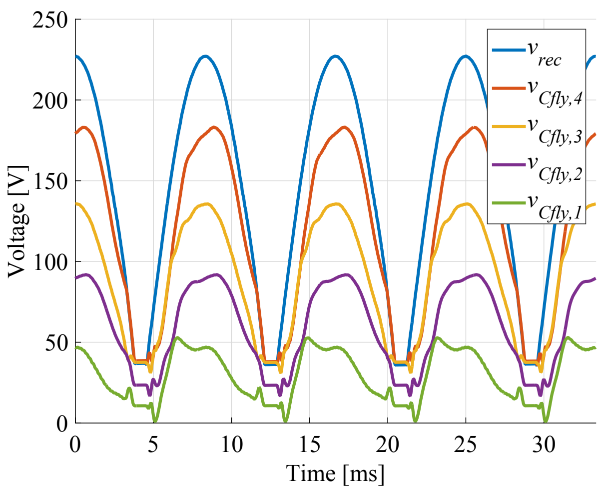
shown in Figure 8.9.

Although the flying capacitor voltages shown in Figure 8.10(a) for 60 V_{RMS} input voltage were not well balanced, the resulting voltage stress at the input voltage peak (for instance when t = 0 s in Figure 8.10(a)) seemed reasonable. In order to see how the flying capacitor voltage balancing performance scales to higher input voltages, the input voltage was gradually increased by preserving the voltage conversion ratio and current conduction angle in the 240 V_{RMS} to 48 V case. Here, only 120 V_{RMS} to 24 V and 160 V_{RMS} to 32 V results are provided.

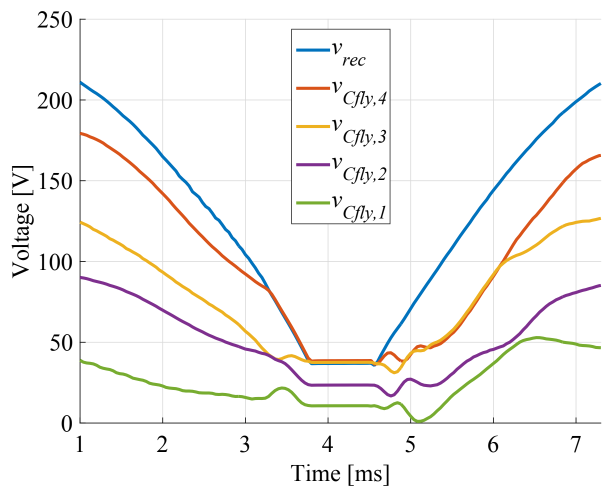
The results of 120 V_{RMS} to 24 V experiment are given in Figure. 8.11. This operating point achieved 0.8684 power factor. The results of 160 V_{RMS} to 32 V experiment are given in Figures 8.12.



(a) Input voltage and current, and inductor current.



(b) Flying capacitor voltages for two ac line cycle.

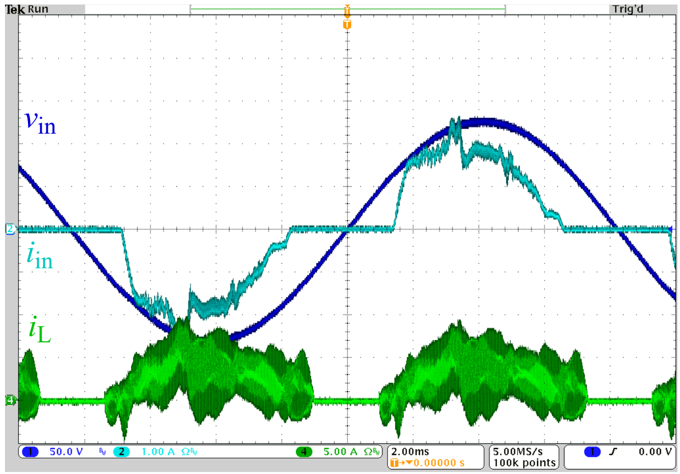


(c) Flying capacitor voltages during converter turn-on and -off.

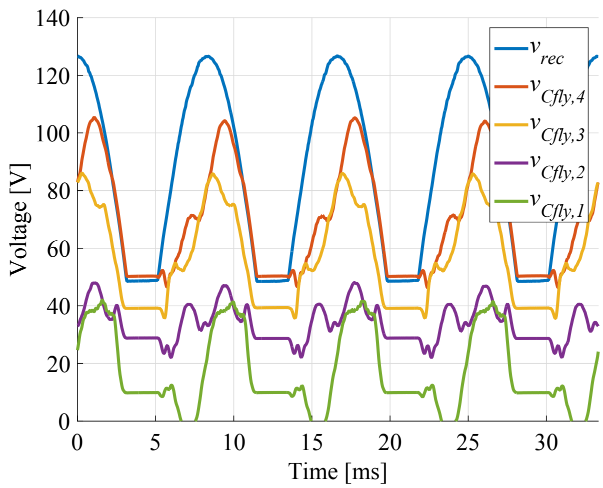
Figure 8.12: Six-level buck converter for PFC operation at $V_{in} = 160$ V$_{\text{RMS}}$, $V_{out} = 32$ V and $I_{out,ave} = 4.5$ A.

This operating point achieved 0.8411 power factor.

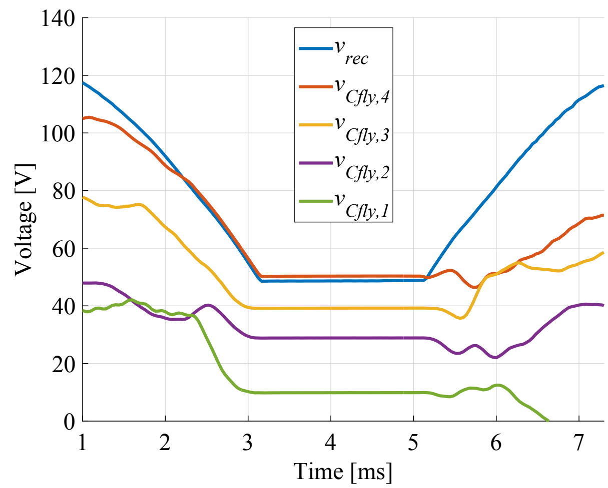
While the current conduction angle was the same in 60 V$_{\text{RMS}}$, 120 V$_{\text{RMS}}$, and 160 V$_{\text{RMS}}$ experiments, higher input voltage resulted in lower power factor. As the input voltage increased, a slight input current displacement can be observed; however, the lower power factor was mainly caused by increased input current distortion in Figures 8.11(a), and 8.12(a) for 120 V$_{\text{RMS}}$ and 160 V$_{\text{RMS}}$, respectively. On the other hand, Figures 8.11(b) and 8.12(b) show that the balancing behavior in the flying capacitor voltages is similar to the 60 V$_{\text{RMS}}$ input voltage case, and arguably worse around peak voltage values. As shown in Figure 8.11(c) and 8.12(c), poor voltage balancing during converter turn-on and -off remained similar to the 60 V$_{\text{RMS}}$ input voltage case. In the 160 V$_{\text{RMS}}$ to



(a) Input voltage and current, and inductor current



(b) Flying capacitor voltages for two ac line cycle.



(c) Flying capacitor voltages during converter turn-on and -off.

Figure 8.13: Six-level buck converter for PFC operation at $V_{in} = 90$ V_{RMS}, $V_{out} = 48$ V and $I_{out,ave} = 4.5$ A.

32 V experiment, the highest switch voltage stress was measured as 56.80 V, whereas with perfect flying capacitor voltage balancing, it should have been 45.3 V. Therefore, higher input voltages were not tested. Instead, the hardware prototype was tested for the low-line universal voltage condition given 48 V output. The PI compensator parameters were updated by targeting 55° phase margin at 10 kHz cut-off frequency to reflect changes in the operating point.

The results of 90 V_{RMS} to 48 V experiments are given in Figure 8.13. This operating point achieved 0.9080 power factor.

As shown in Figure 8.13(a), the inductor and input current shapes with the proposed PFC control have also degraded. Also, Figure 8.13(b) and Figure 8.13(c) show that the balancing behavior in the flying capacitor voltages depends on the current conduction angle, and gets significantly worse as the conduction angle decreases. At this point, it became clear that the flying capacitor voltage balancing must be better understood to improve PFC performance of the prototype hardware and proposed control algorithm.

8.6 Flying capacitor voltage balancing in ac-dc buck conversion

The experimental results in the Section 8.5 suggested that flying capacitor voltage balancing must be enhanced to further pursue an FCML buck converter in PFC applications. There are several approaches to improve voltage balancing. Active balancing techniques proposed in the literature mainly target dc-ac [167, 168] or dc-dc [169, 170] applications, where flying capacitor voltages are balanced around steady voltages. Such techniques require switched node voltage, flying capacitor voltages, or inductor current to be measured, often at higher sampling frequencies than the switching frequency, imposing further practical challenges. Application of active balancing techniques when the operating voltages for the flying capacitors change at 60 Hz has not been reported. It should be also noted that the expected duty ratio change in ideal operating conditions, as depicted in Figure 6.3, has the highest rate of change at the beginning and end of the current conduction angle. An appropriate active balancing technique for this application, especially at the beginning and end of the current conduction angle, must compensate for voltage imbalance considerably faster than the input voltage and duty ratio rate of change, which is not apparent even if existing active balancing techniques are implemented. Therefore, methods to improve natural voltage balancing of the flying capacitors, such as reducing the time constant of flying capacitor voltage balancing dynamics, and changing the phase shift direction of the gate drive signals, are investigated experimentally in this work.

8.6.1 Time constant of flying capacitor voltage balancing dynamics

FCML converter voltage balancing dynamics were analyzed in time domain for three- [163], four- [164], five- [165] and six-level [166] converters. The analysis and exact results for each converter are mathematically complicated. Here, only the relationships between the component values and operating parameters are provided.

A flying capacitor voltage, as provided for many different cases in [163–166], can be summarized by

$$v_C(t) = v_{C,nom} + \exp(-t/\tau)g(t), \tag{8.1}$$

where $v_{C,nom}$ is the nominal voltage of each flying capacitor (i.e., a fraction of the input voltage), τ is the damping time constant of the flying capacitor voltage dynamics, and $g(t)$ is a function of coupled flying capacitor voltages and oscillatory charge transfers between unbalanced flying capacitors. Similar to the flying capacitor voltage, the damping time constant τ, as provided for many different cases in [163–166], has the following parameter dependencies:

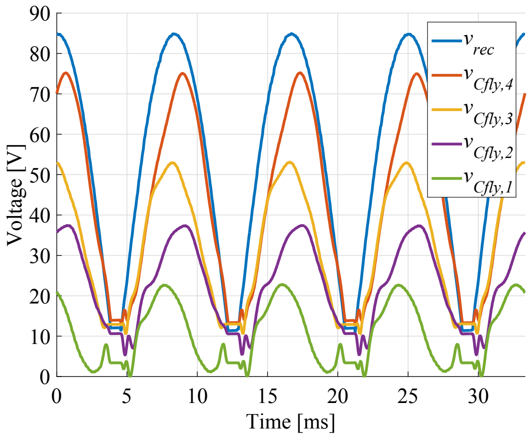
$$\tau \propto L^2, f_{sw}^2, C_{fly}, \frac{1}{R}, h(D, N), \tag{8.2}$$

where L is the inductor value, f_{sw} is the transistor switching frequency, C_{fly} is the flying capacitor value, R is the load, and $h(.)$ is a nonlinear function that depends on the duty ratio D and the number of levels N.

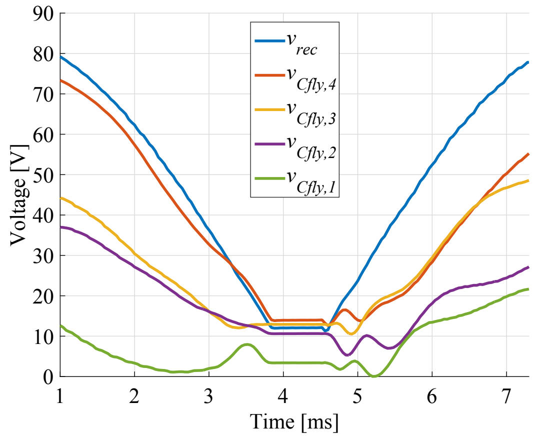
According to (8.1), the dynamic behavior of the flying capacitor voltages decays with a time constant τ, which is related to circuit parameters by (8.2). From (8.1) and (8.2), natural balancing is faster for smaller L, C_{fly}, and f_{sw}, but larger R. It is acknowledged that (8.1) and (8.2) govern flying capacitor voltage dynamics when the input voltage is constant, which is not the case in the ac-dc applications. Here, the relations summarized by (8.1) and (8.2) are used to accelerate natural balancing of the flying capacitor voltages by reducing τ to achieve better balancing at the peak input voltage, where voltage balancing is needed for safe converter operation.

The relations summarized by (8.1) and (8.2) were validated for ac-dc PFC application using the six-level hardware prototype with various L and C_{fly} values at various f_{sw} values. As shown in (8.2), C_{fly} is linearly related, while L and f_{sw} are quadratically related, and to accelerate the natural balancing, L, C_{fly} and f_{sw} should be reduced. Although many different L, C_{fly} and f_{sw} values were tested, only key results are reported here. In the remaining experimental results, unless otherwise is noted $V_{in} = 60$ V_{RMS}, $V_{out} = 12$ V and $I_{out,ave} = 4.5$ A.

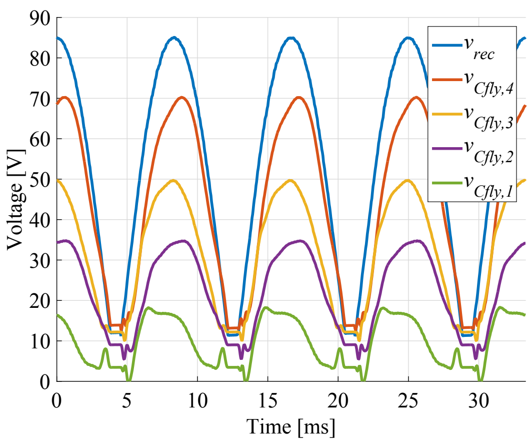
In order to observe natural balancing behavior, the six-level FCML converter in PFC operation was first tested by changing the flying capacitor values between 4×2.2 μF and 8×2.2 μF. The flying capacitor voltages for selected C_{fly} values are given in Figure 8.14.



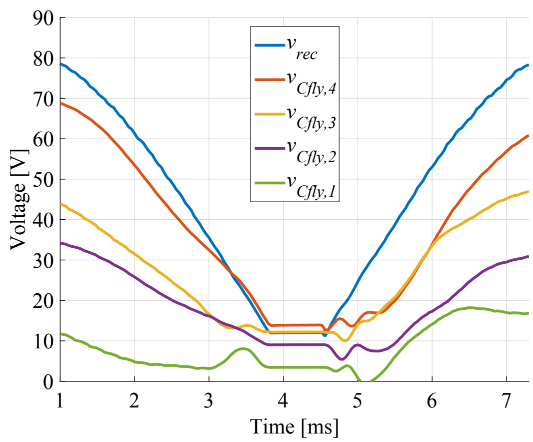
(a) For two ac line cycle. $C_{fly} = 8 \times 2.2\ \mu$F



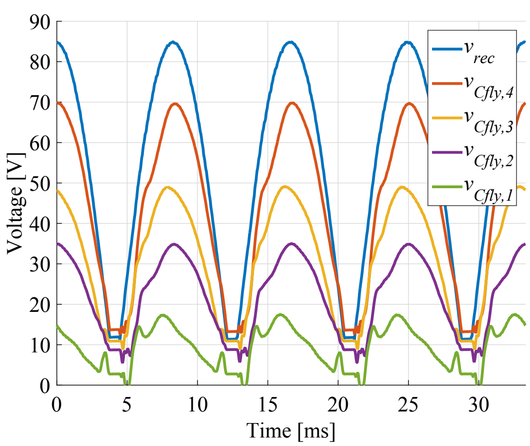
(b) During converter turn-on and -off. $C_{fly} = 8 \times 2.2\ \mu$F



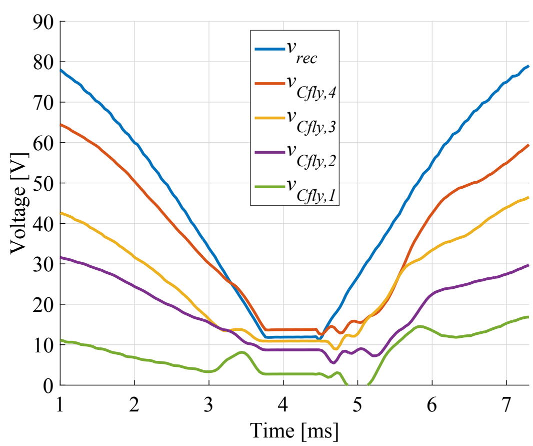
(c) For two ac line cycle. $C_{fly} = 6 \times 2.2\ \mu$F



(d) During converter turn-on and -off. $C_{fly} = 6 \times 2.2\ \mu$F



(e) For two ac line cycle. $C_{fly} = 4 \times 2.2\ \mu$F



(f) During converter turn-on and -off. $C_{fly} = 4 \times 2.2\ \mu$F

Figure 8.14: Flying capacitor voltages of six-level buck converter in PFC operation for different C_{fly} values.

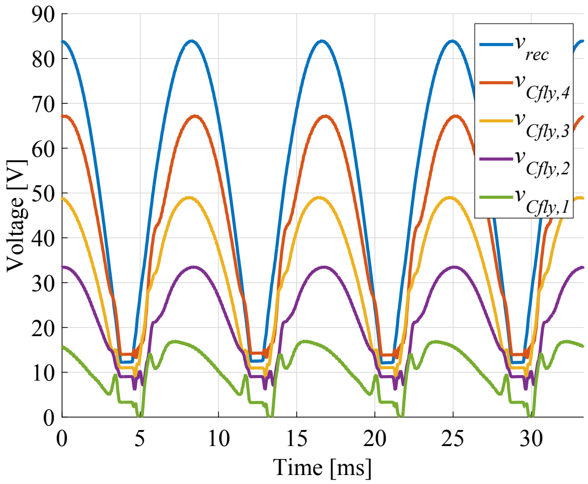
As is apparent in Figure 8.14, C_{fly} changes the balancing behavior of flying capacitors in PFC operation as they charge and discharge at twice-line frequency. According to (8.2), τ should reduce (or the natural balancing should accelerate) as C_{fly} is reduced from 8×2.2 μF to 4×2.2 μF. A visual comparison of Figure 8.14(a), 8.14(c), and 8.14(e) shows that the peak voltages of $C_{fly,1}$ through $C_{fly,4}$ better align with the peak voltage of V_{rec} as C_{fly} reduces. Also, Figure 8.14(b), 8.14(d), and 8.14(f) show that the damping time constant becomes smaller as C_{fly} reduces and flying capacitor voltages tend to approach their expected values sooner after the converter is enabled. With reduced C_{fly}, it is evident in Figure 8.14 that the flying capacitor voltages are still far from their expected values, and further acceleration of natural balancing is necessary. As mentioned before, C_{fly} is linearly related to τ, while L and f_{sw} are quadratically related. Therefore, further acceleration of natural balancing was investigated by adjusting f_{sw} and L while keeping C_{fly} constant at 6×2.2 μF, as described in the remainder of this chapter.

In order to further accelerate natural balancing, the six-level FCML converter in PFC operation was tested by reducing f_{sw} to 40 kHz (i.e., half of the previous switching frequency which should accelerate natural balancing by 4 times). The flying capacitor voltages for this test are given in Figure 8.15(a) and 8.15(b).

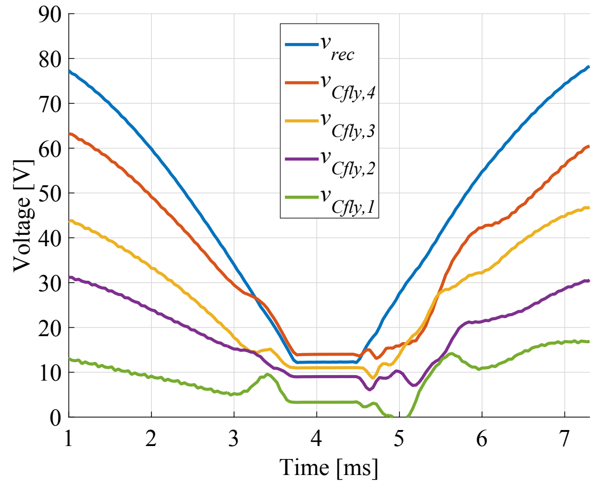
As can be seen in Figure 8.15(a), natural balancing of the flying capacitor voltages was accelerated, yielding better alignment around the input voltage peak compared to the flying capacitor voltages in Figure 8.10. The voltage imbalance during the beginning and end of the current conduction angle still exists, as shown in Figure 8.15(b), but is reduced to a certain extent in comparison to the imbalance in Figure 8.10(b).

According to (8.2), natural balancing can also be accelerated by reducing L. The six-level FCML converter in PFC operation was tested by reducing L to 2.8 μH at a switching frequency of 80 kHz. The flying capacitor voltages for this test are given in Figure 8.15(c) and Figure 8.15(d). Following (8.2), this test should result in flying capacitor voltage behavior similar to the results in Figure 8.15(a) and Figure 8.15(b), since the effective τ is the same in both tests. Close examination of Figure 8.15(a) versus Figure 8.15(c), and Figure 8.15(b) versus Figure 8.15(d) shows that this is indeed the case.

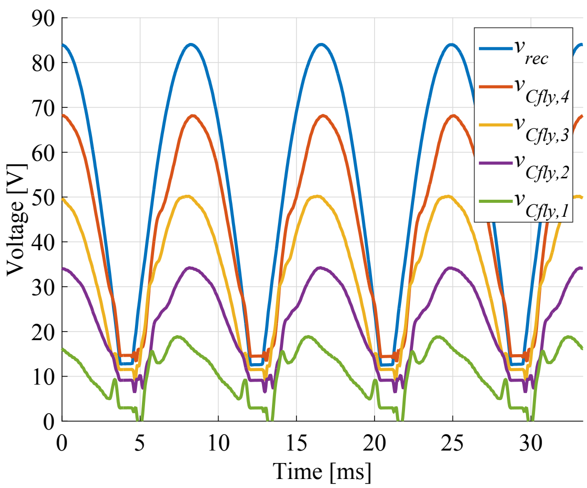
Although a choice of smaller τ by reducing either f_{sw} or L accelerated natural balancing, the flying capacitor voltages still did not reasonably follow appropriate fractions of the rectified input voltage. Also, due to the rapid duty ratio change (as depicted in Figure 6.3) in the beginning



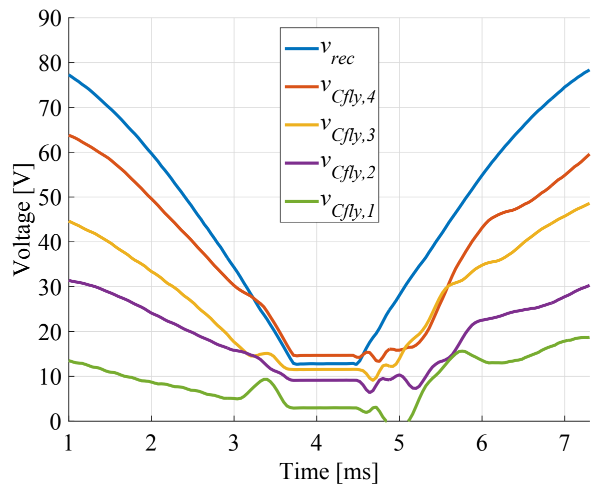
(a) For two ac line cycle. $f_{sw} = 40$ kHz, $L = 5.6$ μH



(b) During converter turn-on and -off. $f_{sw} = 40$ kHz, $L = 5.6$ μH



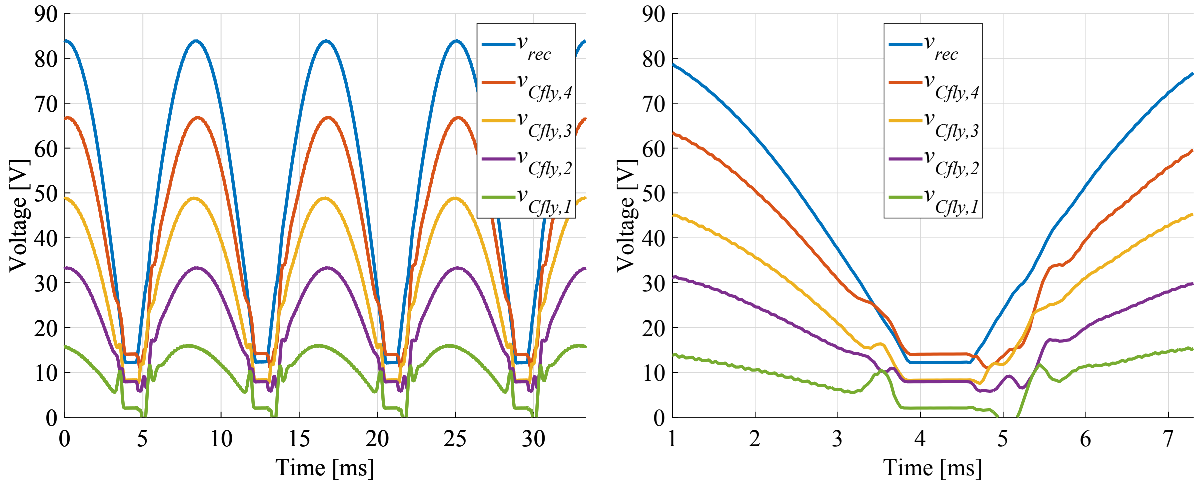
(c) For two ac line cycle. $f_{sw} = 80$ kHz, $L = 2.8$ μH



(d) During converter turn-on and -off. $f_{sw} = 80$ kHz, $L = 2.8$ μH

Figure 8.15: Flying capacitor voltages of six-level buck converter in PFC operation for different f_{sw} and L values. $C_{fly} = 6 \times 2.2$ μF.

and end of the conduction angle, the six-level FCML converter goes through the highest four duty ratio ranges (i.e., $1<D<0.2$) when the duty ratio has the highest rate of change. However, the converter operates in the lowest duty ratio range (i.e., $0<D<0.2$) during a substantial portion of each half line cycle. Therefore, the time constant given by (8.2) is expected to have limited impact at the beginning and end of the current conduction period, unless reduced drastically. This is also apparent in Figures 8.14(b), 8.14(d), 8.14(f), 8.15(b), 8.15(d), among which the change in τ was limited to a few times. Therefore, τ was even further reduced by updating the hardware prototype with $L = 2.6$ μH and $f_{sw} = 40$ kHz, which represents a factor of 16 reduction compared to the first



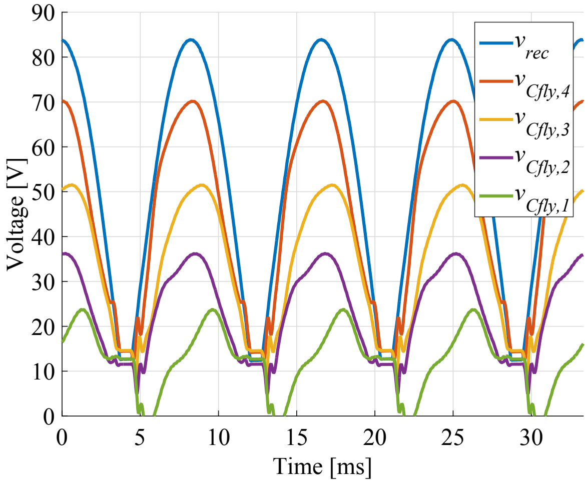
(a) For two ac line cycle.

(b) During converter turn-on and -off.

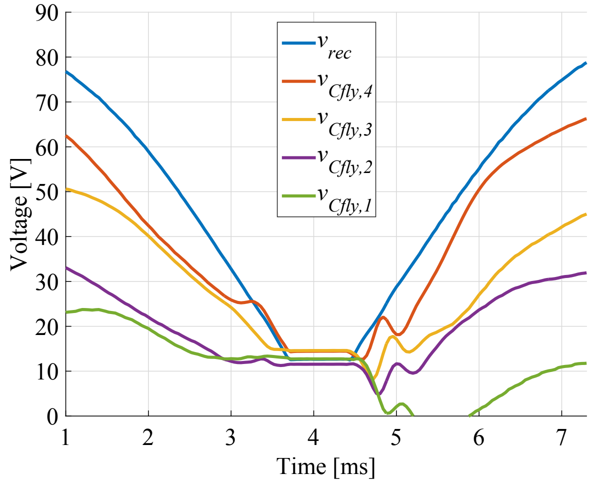
Figure 8.16: Flying capacitor voltages of six-level buck converter in PFC operation.
$C_{fly} = 6 \times 2.2\ \mu$F, $L = 2.8\ \mu$H, f_{sw} =40 kHz.

result where $C_{fly} = 6 \times 2.2\ \mu$F, $L = 5.8\ \mu$H and $f_{sw} = 80$ kHz. The flying capacitor voltages of the updated hardware prototype are given in Figure 8.16.

As expected, natural balancing of the flying capacitor voltages was further accelerated, yielding better alignment around the input voltage peak, as can be seen in Figure 8.16(a). However, voltage imbalance during the beginning and end of the current conduction angle still exists, as shown in Figure 8.16(b). More importantly, reduction in voltage imbalance during the beginning and end of the current conduction angle was still limited compared to the case with f_{sw} = 40 kHz, L = 5.8 μH (as shown in Figure 8.15(b)), or the case with f_{sw} = 80 kHz, L = 2.8 μH (as shown in Figure 8.15(d)). This implies that the available improvement at the beginning and end of the conduction angle by accelerating natural balancing has reached its limit, and other methods should be pursued to enhance flying capacitor voltage behavior during the beginning and end of the current conduction angle. Nevertheless, the updated hardware prototype achieves reasonable voltage balancing at the rectified input voltage peak, where voltage balancing was necessary for safe converter operation at rated input voltage. It should also be noted that the reduction of both switching frequency and inductance value yielded a significantly increased current ripple on the inductor, which increased losses. In general, active balancing methods that enable switching frequency and inductance values to be determined by efficiency and power density targets, rather than by natural balancing time constants, are certainly desirable.



(a) For two ac line cycle.



(b) During converter turn-on and -off.

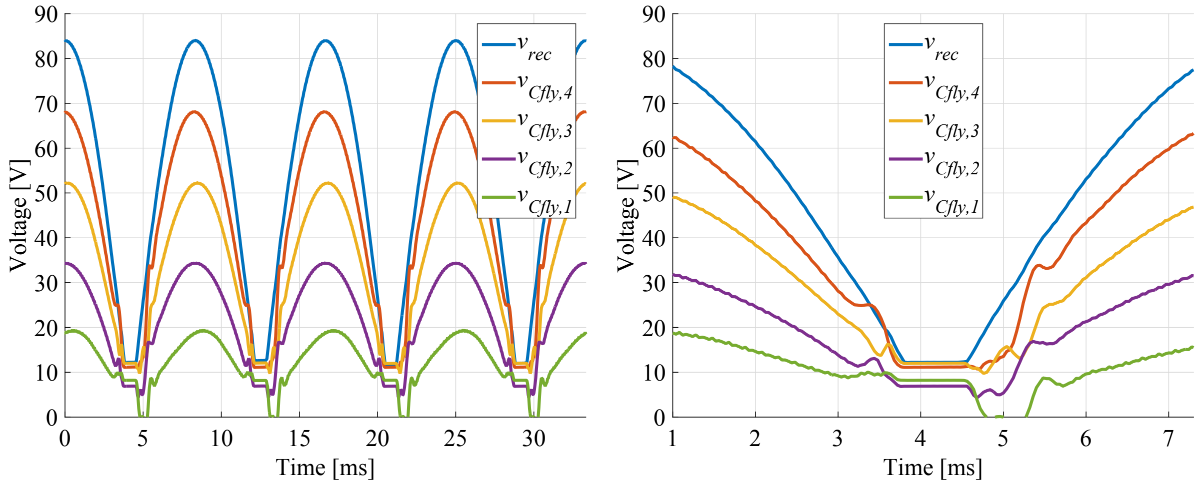
Figure 8.17: Flying capacitor voltages of six-level buck converter in PFC operation. $C_{fly} = 6 \times 2.2\ \mu$F, f_{sw} = 80 kHz, L = 5.6 μH, phase-shift direction: Lag.

8.6.2 Phase-shift direction

Until now, the preferred phase-shift direction for the gate driving signals is known as *lead* (i.e., referring to the transistor order in Figure 8.1, S_{5A} leads S_{4A}, S_{4A} leads S_{3A}, S_{3A} leads S_{2A}, S_{2A} leads S_{1A}). The phase-shift direction can also *lag* (i.e., referring to the transistor order in Figure 8.1, S_{5A} lags S_{4A}, S_{4A} lags S_{3A}, S_{3A} lags S_{2A}, S_{2A} lags S_{1A}). Lead and lag phase shifts result in inverse switching states. This reverses the charge and discharge order of the flying capacitors. Although lead or lag modulation does not affect the value of τ according to [165], it affects the oscillation order of flying capacitor voltages as natural balancing occurs. Interested readers can refer to [165, 188] for further explanation of lead versus lag modulation. Experimental results are reported here for ac-dc operation to investigate the effect of phase shift direction on natural balancing.

Flying capacitor voltages in the six-level FCML converter operated with lag phase shift were tested for L = 5.6 μH and f_{sw} = 80 kHz. Results are given in Figure 8.17. Lag phase shift severely affected the flying capacitor voltages at the rectified input voltage peak as shown in Figure 8.17(a). Compared to Figure 8.10(a), where the same converter specifications operated with lead phase shift, lag phase shift resulted in worse voltage balancing at input voltage peak.

Flying capacitor voltages of the six-level FCML converter operated with lag phase shift were also tested when natural balancing is accelerated by reducing L to 2.8 μH and switching frequency to 40 kHz. Flying capacitor voltages for this test are given in Figure 8.18. Accelerated natural balancing shown in Figure 8.18 mitigated the undesired difference between lead and lag phase shift



(a) For two ac line cycle.

(b) During converter turn-on and -off.

Figure 8.18: Flying capacitor voltages of six-level buck converter in PFC operation.
$C_{fly} = 6 \times 2.2$ μF, $f_{sw} = 40$ kHz, L = 2.8 μH, phase-shift direction: Lag.

between Figures 8.10 and 8.17, and did not severely affect the flying capacitor voltages compared to Figure 8.16. As can be seen in Figure 8.18(b), lag phase shift significantly affected voltage oscillation at the beginning and voltage imbalance at the end of the current conduction angle, compared to Figure 8.16(b), where the converter is operated with the same specifications except for lead phase shift.

In conclusion, experimental results showed that phase-shift direction has a nonnegligible effect on flying capacitor voltages in ac-dc operation as they follow the rectified input voltage at 120 Hz.

8.6.3 Impact on input and inductor current shaping

So far, the experimental work in this chapter has prioritized improving natural balancing of flying capacitor voltages in order to be able to run the six-level buck converter at rated input voltage (i.e., 240 V_{RMS}). Various flying capacitance, switching frequency, inductance, and phase shift directions, and their several combinations were explored experimentally. However, their impact on input current shape has not been reported to focus on the behavior of flying capacitor voltages as the rectified input voltage varies at 120 Hz. Current shaping performance of the converter with selected key parameters is reported here.

First, the converter was operated with lag phase shift at $f_{sw} = 80$ kHz and L = 5.6 μH. The input voltage and current, and inductor current, for this configuration can be seen in Figure 8.19.

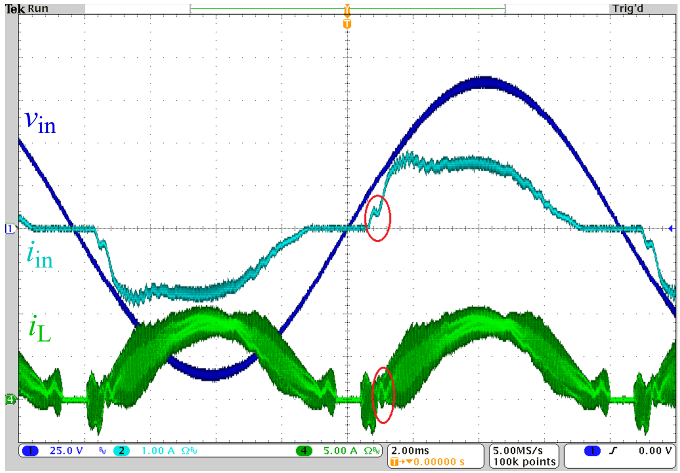
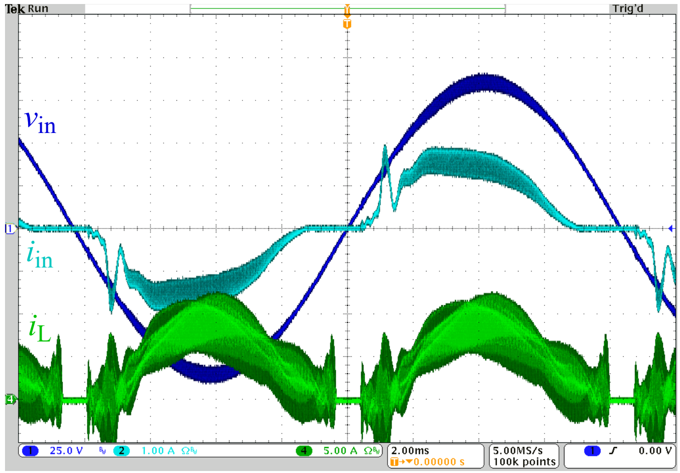


Figure 8.19: The input and output voltage, current, and power of the six-level buck converter for PFC operation. $C_{fly} = 6 \times 2.2\ \mu$F, $f_{sw} = 80$ kHz, L $= 5.6\ \mu$H, phase-shift direction: Lag.

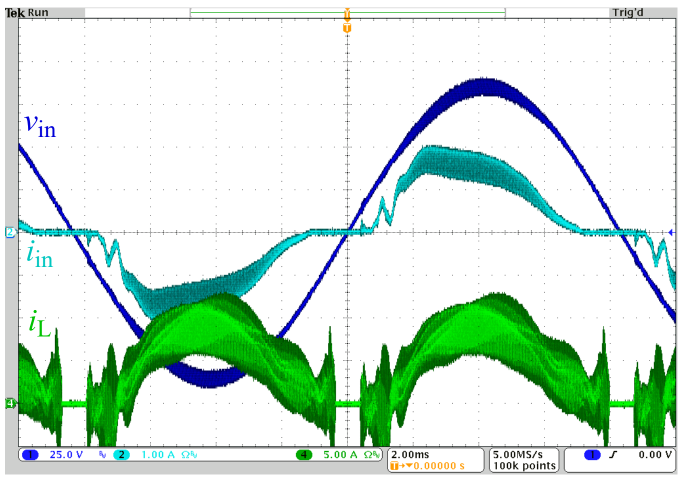
This operating point achieved 0.9331 power factor. In comparison to Figure 8.9, where the same converter is operated with lead phase shift, the input current in Figure 8.19 exhibited a small but noticeable cusp. Cusp distortion on input current is typical of zero-crossing distortion and occurs due to the limited voltage across the inductor to drive the inductor current to follow the reference in boost-type single-phase PFC converters [189]. In the buck-type PFC considered in this work, a cusp is expected following converter turn-on, after the input voltage exceeds the output voltage, instead of right after the input voltage zero crossing. Although not clearly visible on the inductor current plot due to excessive ripple, the cusps are annotated on both the inductor current and the input current in Figure 8.19.

Next, the converter was operated with lead and lag phase shift at $f_{sw} = 40$ kHz and L $= 2.8\ \mu$H. The input voltage and current, and inductor current of the converter under these configurations, can be seen in Figure 8.20. Lead phase shift achieved 0.9252 power factor, and lag phase shift achieved 0.9288 power factor.

Both Figure 8.20(a) and Figure 8.20(b) exhibit larger cusps in the input current, even though the converter employs a smaller inductance. This behavior is counterintuitive since smaller inductor should have mitigated cusp distortion, as is generally the case in boost type PFC converters [189]. However, it should be noted that in Figure 8.20(a) and Figure 8.20(b), the switching frequency is also reduced by half, the same as the inductance. In a buck-type FCML converter, the switch node voltage is a combination of flying capacitor voltages, which are, as experimentally shown in Section 8.6, impacted by the natural balancing time constant and phase shift direction, especially



(a) Phase-shift direction: Lead.



(b) Phase-shift direction: Lag.

Figure 8.20: The input and output voltage, current, and power of the six-level buck converter for PFC operation. $C_{fly} = 6 \times 2.2$ μF, $f_{sw} = 40$ kHz, L $= 2.8$ μH.

during the beginning of the current conduction angle. Presumably, the nonideal switch node voltage due to flying capacitor voltage imbalance has more impact on current shape than the size of the inductor, at least at the beginning of the conduction period.

In conclusion, experimental results showed that phase shift direction, inductance, and switching frequency have less effect on power factor than on flying capacitor voltages. Therefore, the hardware prototype specifications in Table 8.1 were updated with $f_{sw} = 40$ kHz and L $= 2.8$ μH, and the converter was operated with lag phase shift in subsequent experimental work.

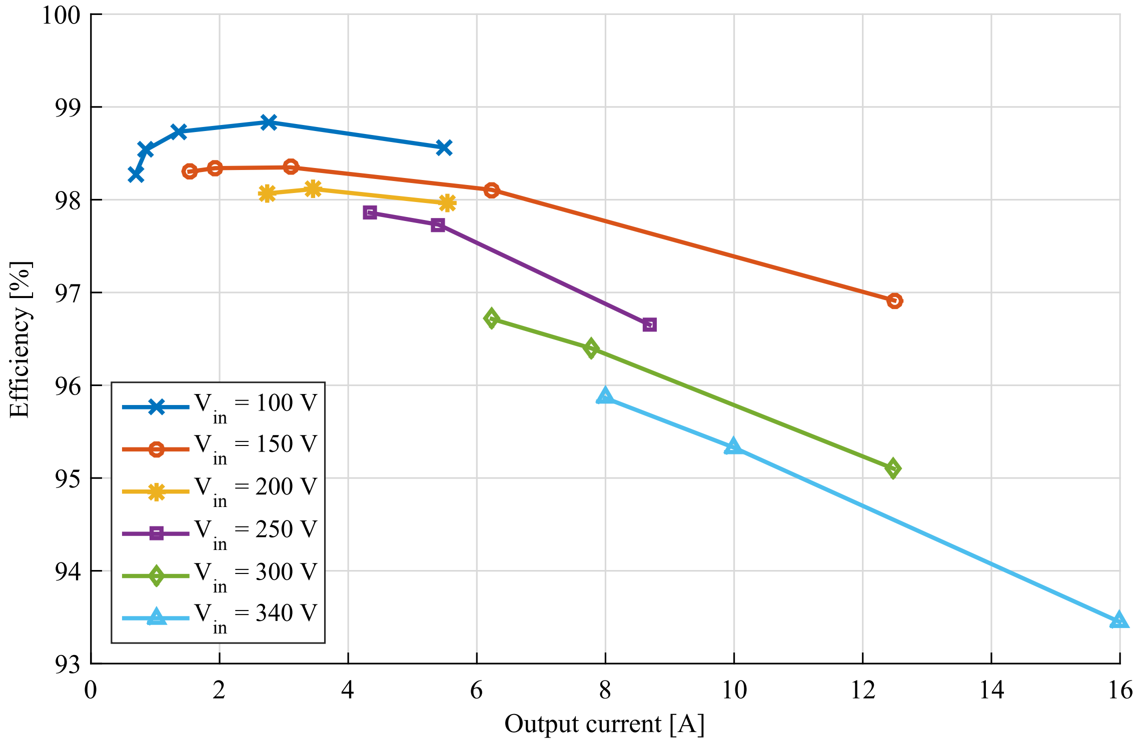
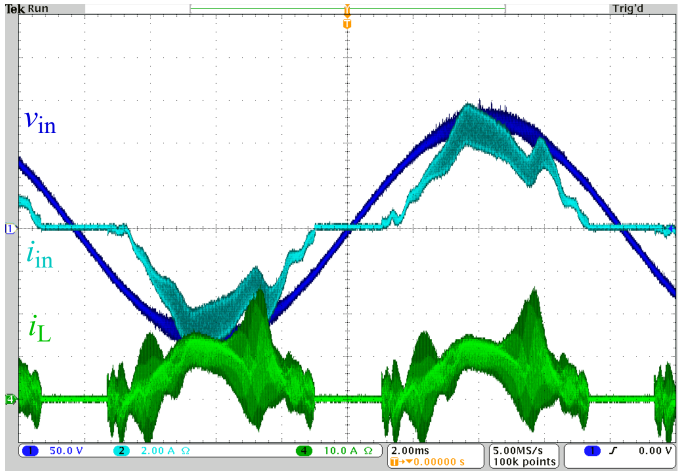


Figure 8.21: Efficiency of the updated FCML buck converter in dc-dc operation.

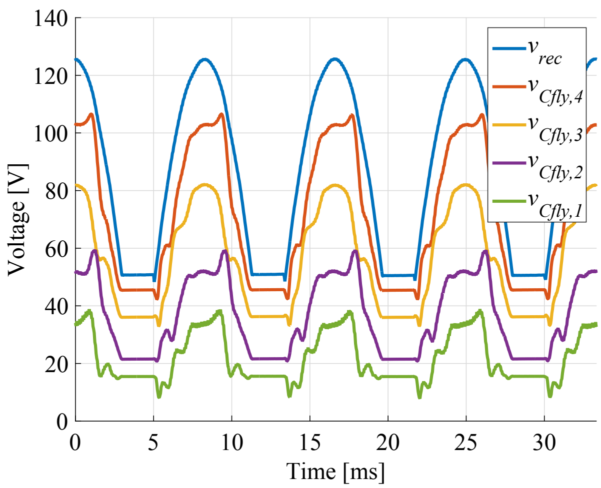
8.7 Six-level FCML converter in universal input ac-dc conversion

Before testing the updated six-level buck converter in universal voltage operation, it was operated as a dc-dc converter to record its performance at various operating points across the rectified ac cycle as well. The input voltage, duty cycle, and output current of the converter were manually adjusted to create operating points that the converter would run during the ac-dc conversion as in Section 8.3. Without further increasing the flying capacitor values, f_{sw} = 40 kHz and L = 2.8 μH limit the average output current to 4.5 A at 240 V_{RMS} input voltage for safe operation of the hardware prototype. This reduces the power density of the hardware prototype with the heat sink to 41 W/in^3. The updated dc-dc converter efficiency is given in Figure 8.21.

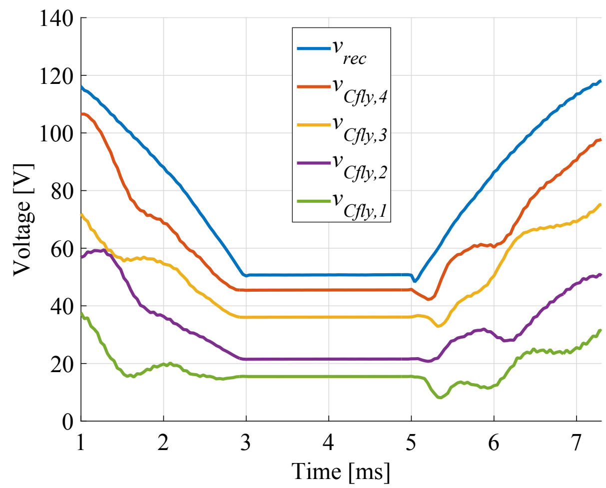
For 90 V_{RMS} input voltage, the six-level buck converter achieved 0.935 power factor and 95.63% power conversion efficiency at rated current. The input voltage and current, and inductor current, are given in Figure 8.22(a). Flying capacitor voltages are given in Figure 8.22(b) for two full ac line cycles. Figure 8.22(c) shows the flying capacitor voltages during converter turn-on and -off. As shown in Figure 8.22(a), inductor and input current shaping performance of the proposed PFC control degraded due to reduced switching frequency and the smaller inductor, compared to results



(a) Input voltage and current, and inductor current



(b) Flying capacitor voltages for two ac line cycle.

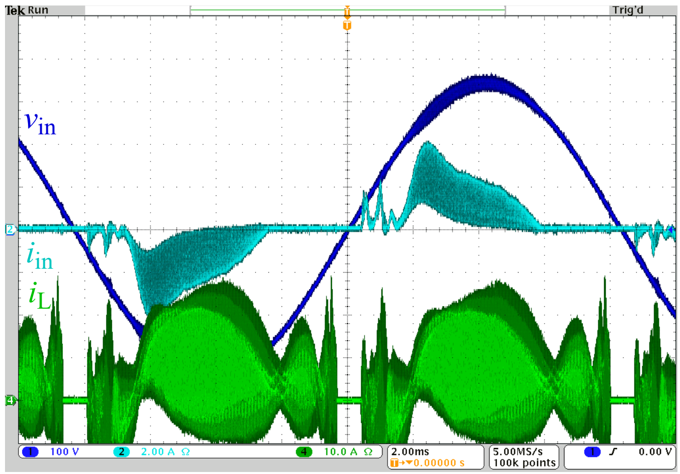


(c) Flying capacitor voltages during converter turn-on and -off.

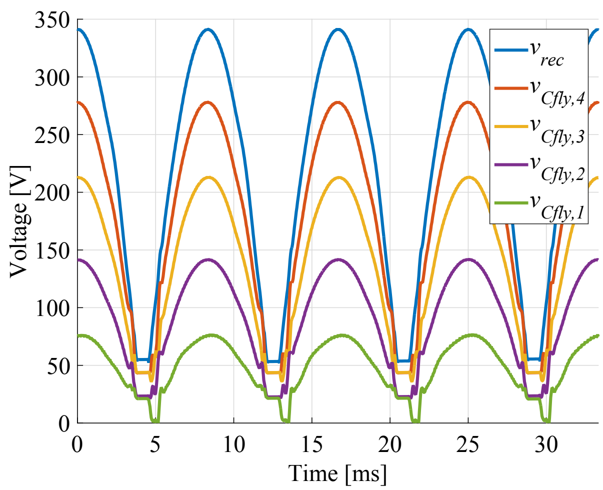
Figure 8.22: Six-level buck converter for PFC operation at $V_{in} = 90$ V_{RMS}, $V_{out} = 48$ V and $I_{out,ave} = 4.5$ A.

at $f_{sw} = 80$ kHz and L $= 5.6$ μH shown in Figure 8.13(a). Figure 8.22(b) and Figure 8.22(c) show that balance in the flying capacitor voltages has improved compared to results at $f_{sw} = 80$ kHz and L $= 5.6$ μH in Figure 8.13(b) and Figure 8.13(c). However, reduced current conduction angle still negatively affected flying capacitor voltage balance. Nevertheless, at low voltage, poor voltage balance did not present a hazardous operating condition for the converter, although it impacted overall efficiency.

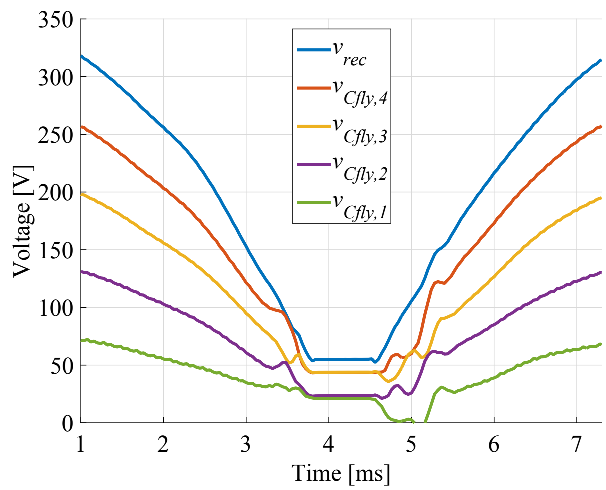
For 240 VRMS input voltage, the six-level buck converter achieved 0.741 power factor and 91.714% power conversion efficiency at rated current. The input voltage and current, and inductor current, are given in Figure 8.23(a). Flying capacitor voltages are given in Figure 8.23(b) for



(a) Input voltage and current, and inductor current



(b) Flying capacitor voltages for two ac line cycle.

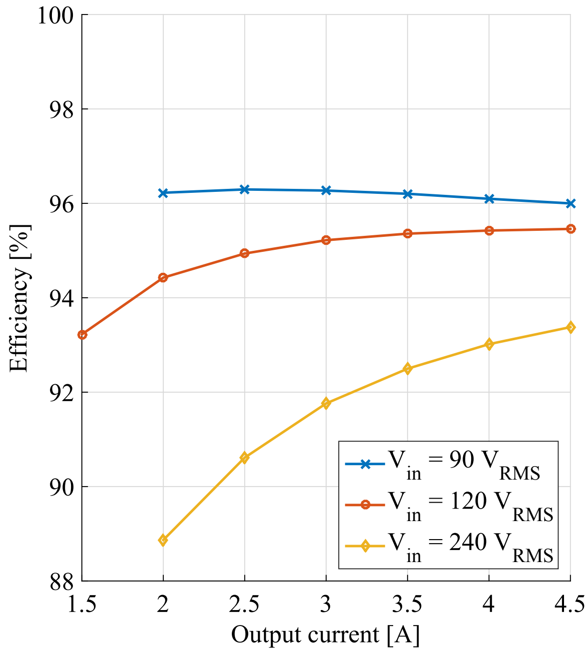


(c) Flying capacitor voltages during converter turn-on and -off.

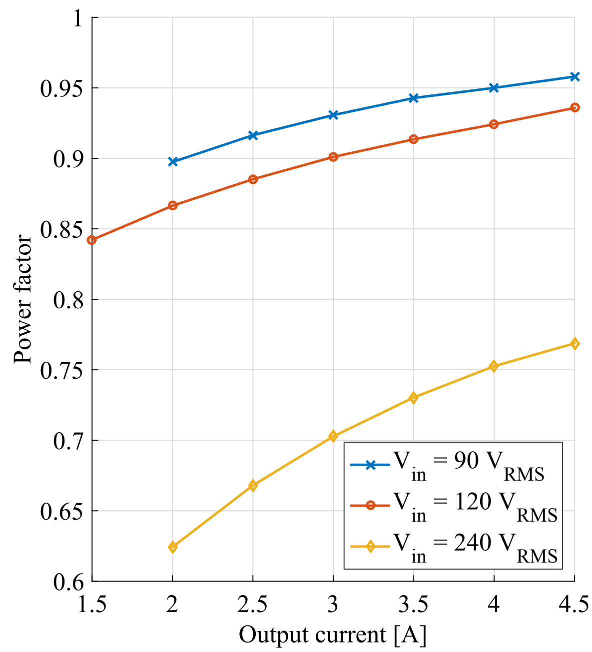
Figure 8.23: Six-level buck converter for PFC operation at $V_{in} = 240$ V_{RMS}, $V_{out} = 48$ V and $I_{out,ave} = 4.5$ A.

two full ac line cycles. Figure 8.23(c) shows the flying capacitor voltages during converter turn-on and -off. As shown in Figure 8.23(a), inductor and input current ripple increased, and the shape performance of the proposed PFC control degraded further due to increased input voltage. Figure 8.23(b) and Figure 8.23(c) show that flying capacitor voltage balance was maintained at rated input voltage.

For 90 V_{RMS}, 120 V_{RMS}, and 240 V_{RMS} input voltages, Figure 8.24(a) and 8.24(b) show output current versus measured ac to dc conversion efficiency and power factor, respectively. As shown in Figure 8.24(a), ac-dc conversion efficiency reduced as the input voltage increases. This is typical of buck-PFC converters since higher input voltage requires larger voltage step-down, and thus lower



(a) Efficiency vs. output current



(b) Power factor vs. output current

Figure 8.24: AC to dc conversion efficiency and power factor at 90, 120 and 240 V_{RMS} input voltage.

efficiency in buck conversion. Similar behavior was also observed in Figure 8.24(b) where power factor reduces as input voltage increases, although lower power factor for higher input voltage in buck-type PFC conversion is an unusual result. In a two-level buck converter, higher input voltage would inherently result in higher power factor because higher input voltage yields longer current conduction angle throughout the ac line cycle. However, in order to achieve reasonably fast natural balancing in the six-level FCML converter, lower inductance was needed, which degraded the current shaping capability of the PFC controller.

CHAPTER 9

CONCLUSION AND FUTURE WORK

9.1 Conclusion

The series-stacked power delivery architecture discussed in this dissertation showed that it is capable of maintaining server operation under various conditions. It is a suitable candidate to replace the bus conversion stage in conventional architectures. By electrically connecting the servers in series, the proposed series-stacked power delivery architecture achieves superior power delivery efficiency.

A hardware prototype that included differential converters and stack initialization circuitry was designed and verified in a real-life scenario that includes startup, hot-swapping, and shutdown of series-stacked servers. In addition, to show the feasibility of 48 V UPS placement at the stack input, the proposed control algorithm was modified to maintain operation of series-stacked servers under varying dc bus voltage. In both cases, more than 99% power delivery efficiency was reported in this dissertation.

The ac-dc front-end power conversion method investigated in this dissertation targets power conversion efficiency and power density improvements by reducing the number of cascaded power stages in single-phase ac-dc power conversion. In data center power delivery applications, the ultimate goal is regulating low dc voltage for digital loads; therefore, an FCML buck PFC converter has been designed and implemented that can provide 48 V from universal input voltage in a single power stage. The FCML topology increases power density by leveraging capacitors along with inductors in the energy conversion process and by reducing the overall required inductor size. However, implementation of an FCML buck converter in a PFC application introduces unique operation scenarios in which the flying capacitor voltages must follow the input voltage at 60 Hz proportionally to ensure proper converter operation. One contribution of this dissertation is experimental exploration of flying capacitor voltage behavior in this unique operation.

A high power density six-level FCML buck converter was designed with GaN transistors to perform the experimental study. A digital PFC control algorithm was developed and verified using a conventional two-level buck converter and a six-level FCML buck converter at low input voltage. In order to achieve natural balancing of flying capacitor voltages at 60 Hz input, both the inductor and the switching frequency, which were initially chosen to achieve high power density for the six-level FCML buck converter, had to be reduced by half. This specification update to accelerate natural balancing of the flying capacitors reduced the power factor, especially as the input voltage increases. Nevertheless, the behavior of flying capacitor voltages as they follow appropriate fractions of a rectified input voltage was successfully evaluated in the experiments.

9.2 Future work

An ultimate goal for future work is to provide a continuous grid-to-server power delivery architecture that cascades an FCML buck PFC converter and a series-stacked architecture in a compact and efficient implementation. If designed as a plug and play solution, future work can also include a comparison to commercially available power converters in actual data centers. This dissertation focuses on efficiency and power density improvements. Reliability analysis of the proposed architectures is a major future research area, since high reliability is as valuable as high efficiency and power density in data centers.

9.2.1 Server-to-virtual bus DPP

Differential converters used in this research were preliminary hardware prototypes designed with off-the shelf components. The design of differential converters can be improved to achieve high power density and efficiency of the converters themselves. Due to the low-voltage nature of differential converters, all analog and digital circuitry and power switches likely can be implemented in an integrated circuit. PCI Express can be used as a single connector for both power and control signals between a motherboard and a differential converter. Through the redesign of differential converters, leveraging recent advancements in planar magnetics and GaN transistors, a high density bidirectional isolated dc-dc converter can be achieved. Power-aware load scheduling algorithms can be extended to balance computational load between the series-stacked servers. Such algorithms reduce the processed power in the system, but also present a guideline for differential converter design. If expected load mismatch between series-stacked servers can be predetermined by software solutions, differential converter design can be optimized to enable lower power rated dc-dc converters.

Data center grade UPS can be incorporated into converter design to test real-life power loss scenarios. In this work, the UPS is assumed to be positioned at the 48 V input; however, a 12 V virtual bus is also a feasible location for UPS integration and should be investigated in the future. The control algorithm can be extended to detect loss and recovery of supply to maintain operation of a series stack under all conditions. Some additional features such as pre-charge of the virtual bus capacitor and automated initialization of a series stack can be integrated into the control algorithm.

The server-to-virtual bus DPP architecture can be built in a standard size server blade that employs commercial processors. Collaborative communities that address efficient power infrastructures for data center applications, such as the Open Compute Project, can be leveraged in server blade design. Using real high performance computing benchmarks, the performance of a server blade with the series-stacked architecture can be compared to that of off-the-shelf server blades that employ conventional power delivery architectures. Such a demonstration could accelerate the adoption of this innovative approach.

9.2.2 Buck-type FCML as a PFC converter

This dissertation showed that achieving natural balancing of flying capacitor voltages in an FCML buck converter used in a single-phase PFC application imposes limits on flying capacitance, inductance, and transistor switching frequency. There are many areas that can be pursued in the future to improve power factor, natural balancing of flying capacitor voltages, and power density of FCML buck converters.

Power factor can be improved by redesigning the input filter, enhancing PFC control and integrating a low voltage boost stage into the FCML buck stage. A higher order input filter can be designed to attenuate input current switching ripple, although such an input filter may introduce input current displacement. Digital control algorithms can be extended to consider and compensate for input current displacement. As mentioned in Section 6.2, to achieve unity power factor, a low voltage boost stage can be added to the buck stage. This addition may help with the input current cusp and voltage imbalance when the buck FCML is enabled at the beginning of a conduction period. Control algorithms that can shape the input current directly instead of the inductor current may improve the power factor achieved in this work. However, such control algorithms may present difficulties when applied to an FCML buck converter that is controlled with phase-shifted PWM signals to achieve natural balancing of the flying capacitor voltages.

Flying capacitor voltages may be controlled using active balancing or pre-charge approaches. Future work can explore cases in which such approaches can have the highest benefit during a line cycle. Accelerated natural balancing enabled the prototype converter to operate at rated voltage by achieving excellent voltage balancing at the input voltage peak. However, accelerated natural balancing was not sufficient to maintain flying capacitor voltage balancing during the beginning and end of the current conduction angle. Perhaps a combination of natural and active balancing can be more beneficial than running an active balancing method over the entire line cycle. Since the natural balancing time constant depends on the number of levels in FCML converters, dynamically adjusting the number of levels may selectively improve voltage balancing. Based on the experience gained during this work, the preferred active balancing, pre-charge or dynamic level selection approach must be quite fast to drive the flying capacitor voltages to their proper values. Such approaches may be challenging to implement, especially at the beginning of the current conduction angle where the input voltage rate of change is the highest.

The hardware prototype can be redesigned to achieve higher power density while achieving power factor correction by considering natural balancing limitations presented in this work. Previous literature showed that the number of levels has an impact on the natural balancing time constant. High power density optimization should consider the trade-off between the inductor, switching frequency, and time constant as a function of the number of levels to design the densest converter possible. Additional components and functionality such as EMI filters, startup routines, and redundancy are also needed so that a single-stage FCML buck converter can be employed in data center applications.

APPENDIX A

DESIGN FILES OF PROTOTYPE DPP HARDWARE

This appendix contains printed circuit board (PCB) layouts of the prototype DPP hardware.

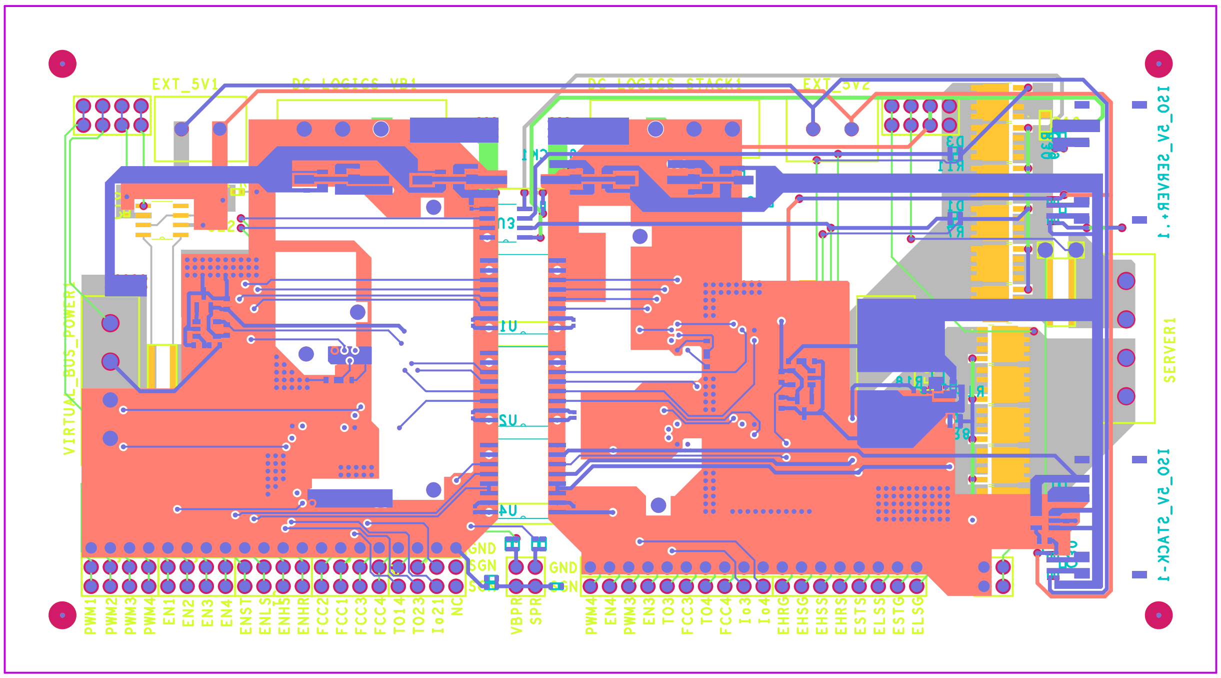


Figure A.1: PCB layout of prototype DPP hardware: All layers, silkscreens and solder masks.

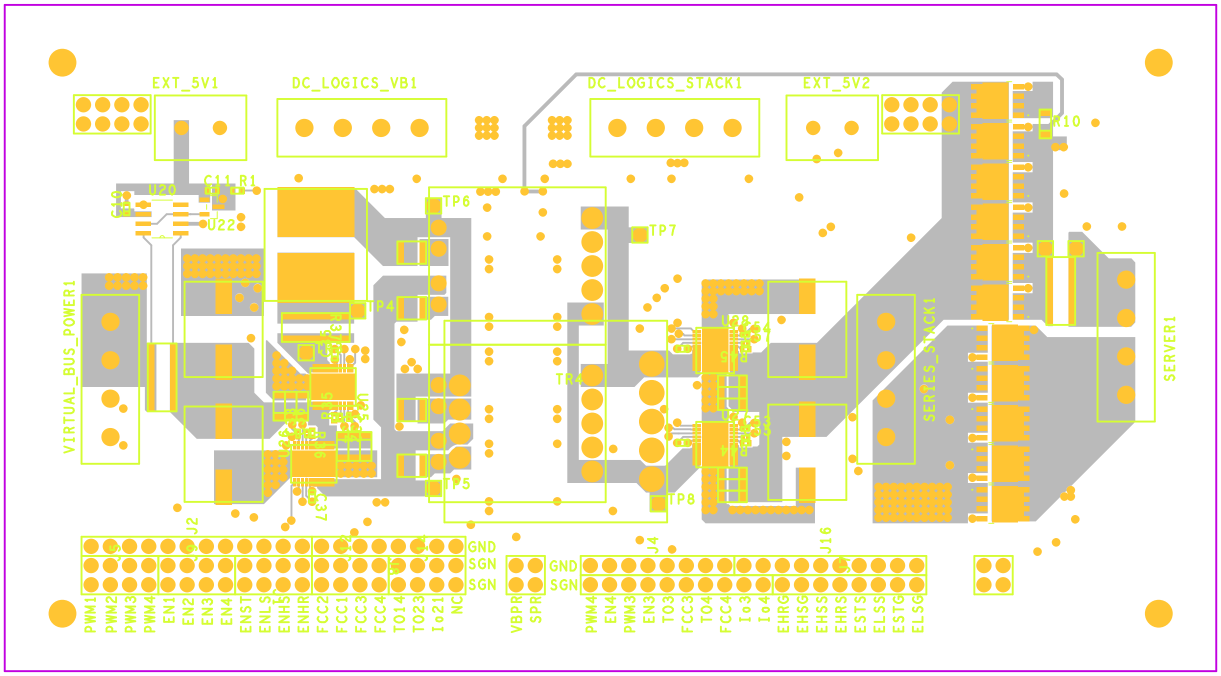


Figure A.2: PCB layout of prototype DPP hardware: Top layer, silkscreen and solder mask.

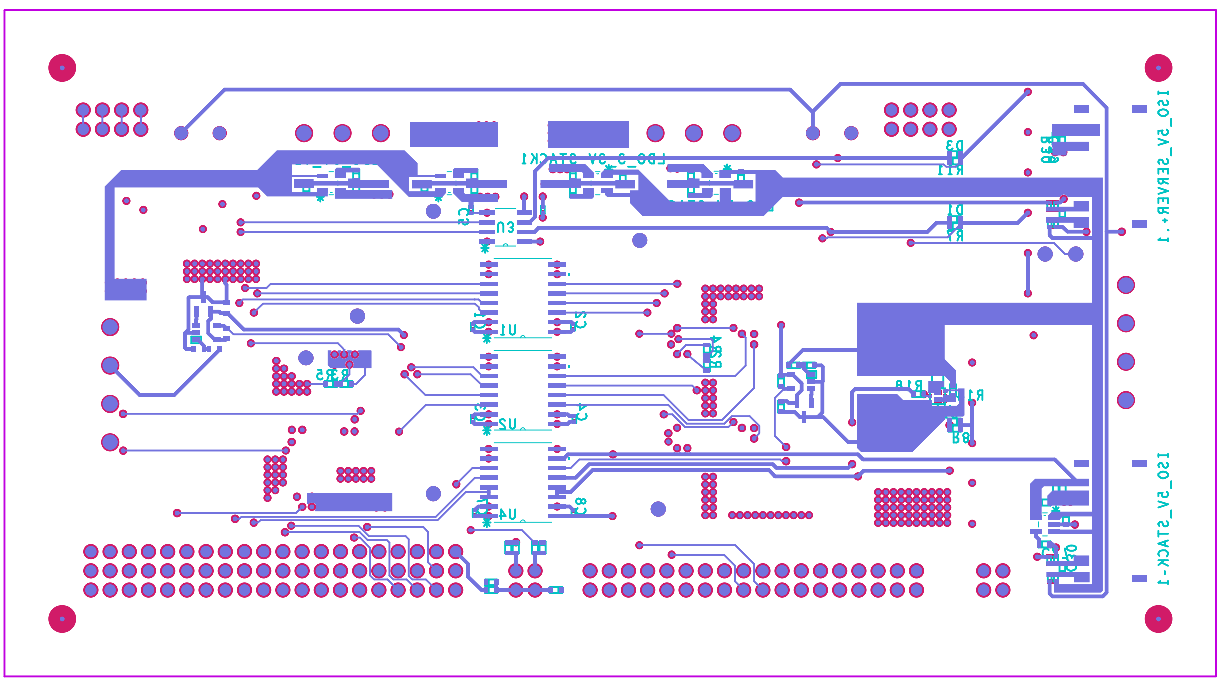


Figure A.3: PCB layout of prototype DPP hardware: Bottom layer, silkscreen and solder mask.

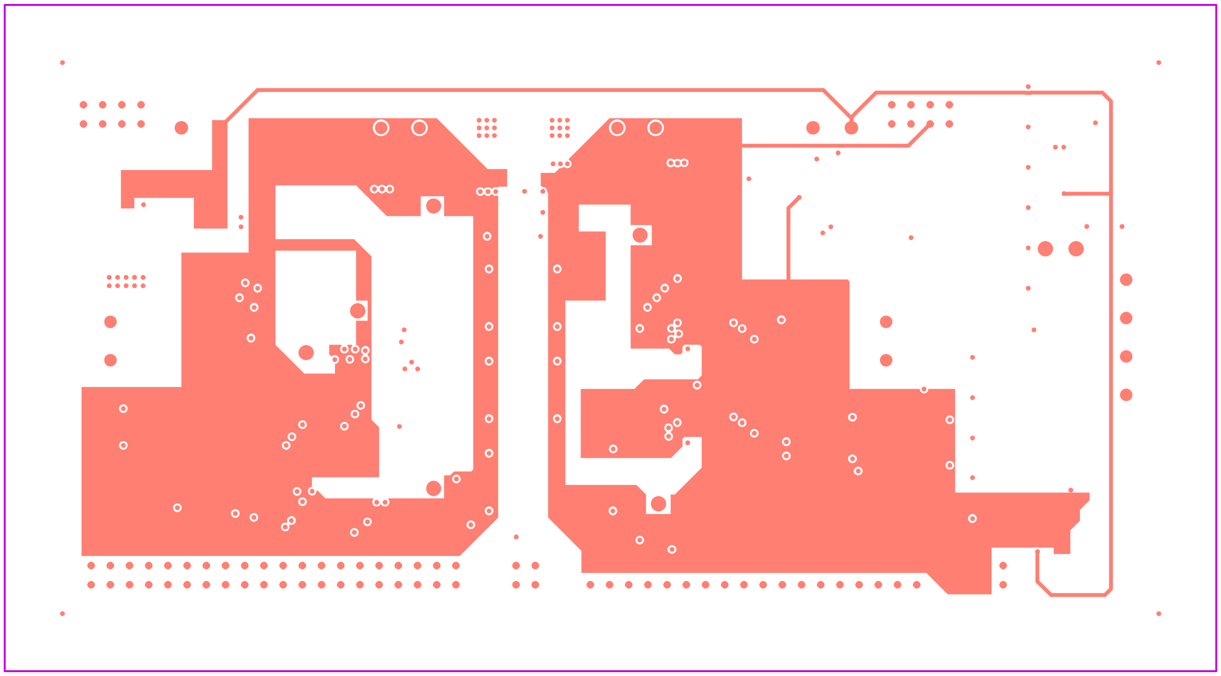


Figure A.4: PCB layout of prototype DPP hardware: Ground layer.

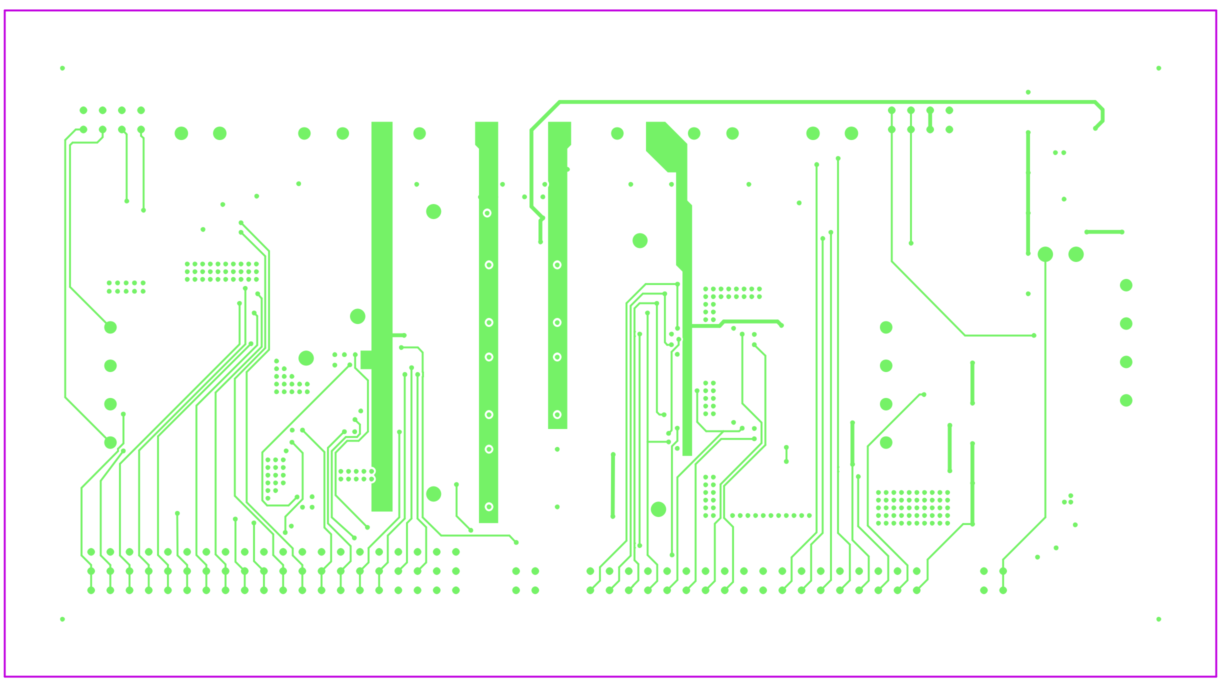


Figure A.5: PCB layout of prototype DPP hardware: Signal layer.

APPENDIX B

MICROCONTROLLER CODE USED IN SERVER-TO-VIRTUAL BUS DPP EXPERIMENTAL STUDY

This appendix contains microcontroller code used in server-to-virtual bus DPP experimental study.

Listing B.1: main.c

```
#include "DSP28x_Project.h"     // Device Header file and Examples Include File

// Prototype statements for functions found within this file.
void Initialize_GPIOs(void);

void Adc_Config(void);

void EPwm_Config(void);
void EPwm1_Config(void);
void EPwm2_Config(void);
void EPwm3_Config(void);
void EPwm4_Config(void);
void EPwm5_Config(void);
void EPwm6_Config(void);
void EPwm7_Config(void);
void EPwm8_Config(void);

void Disable_All_Converters(void);
void Reset_All_Converters(void);
void Update_All_Enables(void);
void Update_All_Phase_Shifts(void);
void Update_HotSwap(void);
void Update_Stack(void);
void Calculate_Voltages(void);
void Determine_Phi_n_Dir(void);
void Swap_All_In(void);
void Swap_All_Out(void);
void Swap_One_In(void);
void Swap_One_Out(void);


interrupt void adc_isr(void);
//interrupt void cpu_timer0_isr(void);

// Global variables used in this file
Uint16 adc_count = 0;        // adc counter
Uint16 adc_divider = 10;       // adc divider for averaging n measurements
Uint16 adc_i = 0;          // adc measurement index
Uint16 adc_ready_flag = 0;

Uint16 swap_one_counter = 0;
Uint16 swap_one_multip = 500;
Uint16 swap_one_in_flag = 0;
```

```
Uint16 swap_one_out_flag = 0;

Uint16 swap_all_counter = 0;
Uint16 swap_all_multip = 500;
Uint16 swap_all_in_flag = 0;
Uint16 swap_all_out_flag = 0;

int16 VVB[10]; //must be same as adc_divider
int16 V1[10];
int16 V2[10];
int16 V3[10];
int16 V4[10];

int16 VVB_ave;
int16 V1_ave;
int16 V2_ave;
int16 V3_ave;
int16 V4_ave;

int16 V_vb;
int16 V_s1;
int16 V_s2;
int16 V_s3;
int16 V_s4;

int16 V_vb_err;
int16 V_s1_err;
int16 V_s2_err;
int16 V_s3_err;
int16 V_s4_err;

int16 V_1;
int16 V_2;
int16 V_3;
int16 V_4;

int16 V_s1_hys_highest_ = -78;//2357;//2370;
int16 V_s1_hys_high_ = -35;//2400;
int16 V_s1_hys_low_ = -15;//2420;
int16 V_s1_hys_low = 17;//2452;
int16 V_s1_hys_high = 35;//2470;
int16 V_s1_hys_highest = 76;//2511;//2492;

int16 V_s2_hys_highest_ = -87;//2292;//2317;
int16 V_s2_hys_high_ = -44;//2335;
int16 V_s2_hys_low_ = -14;//2365;
int16 V_s2_hys_low = 16;//2395;
int16 V_s2_hys_high = 39;//2418;
int16 V_s2_hys_highest = 92;//2471;//2460;

int16 V_s3_hys_highest_ = -85;//2386;//2410;
int16 V_s3_hys_high_ = -41;//2430;
int16 V_s3_hys_low_ = -22;//2449;
int16 V_s3_hys_low = 19;//2490;
int16 V_s3_hys_high = 39;//2510;
int16 V_s3_hys_highest = 87;//2558;//2544;

int16 V_s4_hys_highest_ = -84;//2366;//2386;
int16 V_s4_hys_high_ = -43;//2407;
int16 V_s4_hys_low_ = -20;//2430;
int16 V_s4_hys_low = 20;//2470;
```

```
int16 V_s4_hys_high = 43;//2493;
int16 V_s4_hys_highest = 82;//2532;//2508;

int16 V_vb_hys_highest_ = -207;//2275;// 2334;
int16 V_vb_hys_high_ = -124;//2358;//2376;
int16 V_vb_hys_low_ = -20;//2462;
int16 V_vb_hys_low = 21;//2503;
int16 V_vb_hys_high = 124;//2606;//2538;
int16 V_vb_hys_highest = 207;//2689;//2580;

int16 state_s1 = 0;
int16 state_s1_1 = 0;
int16 state_s2 = 0;
int16 state_s2_1 = 0;
int16 state_s3 = 0;
int16 state_s3_1 = 0;
int16 state_s4 = 0;
int16 state_s4_1 = 0;
int16 state_vb = 0;
int16 state_vb_1 = 0;

Uint16 phi_1 = 0;         // Set Phase, phi ~= - desired_phase / 180 * period , where period is
    defined below, fine tuning may be needed
Uint16 phi_2 = 0;         // Set Phase, phi ~= - desired_phase / 180 * period , where period is
    defined below, fine tuning may be needed
Uint16 phi_3 = 0;         // Set Phase, phi ~= - desired_phase / 180 * period , where period is
    defined below, fine tuning may be needed
Uint16 phi_4 = 0;         // Set Phase, phi ~= - desired_phase / 180 * period , where period is
    defined below, fine tuning may be needed

//DPP#1 is connected to server1
//DPP#2 is connected to server2
//DPP#3 is connected to server3
//DPP#4 is connected to server4

Uint16 phi_1_x = 5;     // phi_1 value for current injection to server
Uint16 phi_1_x_prime = 11;  // phi_1 value for current injection to virtual bus
Uint16 phi_1_xx =  27;    // phi_1 value for current injection to server
Uint16 phi_1_xx_prime = 30; // phi_1 value for current injection to virtual bus

Uint16 phi_2_x = 5;     // phi_2 value for current injection to server
Uint16 phi_2_x_prime = 11;  // phi_2 value for current injection to virtual bus
Uint16 phi_2_xx = 25;     // phi_2 value for current injection to server
Uint16 phi_2_xx_prime = 29;   // phi_2 value for current injection to virtual bus

Uint16 phi_3_x = 5;     // phi_3 value for current injection to server
Uint16 phi_3_x_prime = 11;  // phi_3 value for current injection to virtual bus
Uint16 phi_3_xx = 27;     // phi_3 value for current injection to server
Uint16 phi_3_xx_prime = 30;   // phi_3 value for current injection to virtual bus

Uint16 phi_4_x = 5;     // phi_4 value for current injection to server
Uint16 phi_4_x_prime = 11;  // phi_4 value for current injection to virtual bus
Uint16 phi_4_xx = 27;     // phi_4 value for current injection to server
Uint16 phi_4_xx_prime = 30; // phi_4 value for current injection to virtual bus

Uint16 dir_1 = 0;     // DPP direction; 0 = VB to server(pri2sec), 1 = server to VB (sec2pri)
Uint16 dir_2 = 0;     // DPP direction; 0 = VB to server(pri2sec), 1 = server to VB (sec2pri)
Uint16 dir_3 = 0;     // DPP direction; 0 = VB to server(pri2sec), 1 = server to VB (sec2pri)
Uint16 dir_4 = 0;     // DPP direction; 0 = VB to server(pri2sec), 1 = server to VB (sec2pri)

Uint16 enable_1_1 = 0;    // Enable DPP1 pri switches, initially disabled
```

```c
Uint16 enable_1_2 = 0;    // Enable DPP1 sec switches, initially disabled
Uint16 enable_2_1 = 0;    // Enable DPP2 pri switches, initially disabled
Uint16 enable_2_2 = 0;    // Enable DPP2 sec switches, initially disabled
Uint16 enable_3_1 = 0;    // Enable DPP3 pri switches, initially disabled
Uint16 enable_3_2 = 0;    // Enable DPP3 sec switches, initially disabled
Uint16 enable_4_1 = 0;    // Enable DPP3 pri switches, initially disabled
Uint16 enable_4_2 = 0;    // Enable DPP3 sec switches, initially disabled

Uint16 st_1 = 0;      // Enable Stack 1, initially disabled
Uint16 st_2 = 0;      // Enable Stack 2, initially disabled
Uint16 st_3 = 0;      // Enable Stack 3, initially disabled
Uint16 st_4 = 0;      // Enable Stack 4, initially disabled

Uint16 high_side_high_res_server1 = 0;  // Enable high side high resistance path for server1, initially disabled
Uint16 high_side_low_res_server1 = 0; // Enable high side low resistance path for server1, initially disabled
Uint16 low_side_server1 = 0;      // Enable low side for server1, initially disabled

Uint16 high_side_high_res_server2 = 0;  // Enable high side high resistance path for server1, initially disabled
Uint16 high_side_low_res_server2 = 0; // Enable high side low resistance path for server1, initially disabled
Uint16 low_side_server2 = 0;      // Enable low side for server1, initially disabled

Uint16 high_side_high_res_server3 = 0;  // Enable high side high resistance path for server1, initially disabled
Uint16 high_side_low_res_server3 = 0; // Enable high side low resistance path for server1, initially disabled
Uint16 low_side_server3 = 0;      // Enable low side for server1, initially disabled

Uint16 high_side_high_res_server4 = 0;  // Enable high side high resistance path for server1, initially disabled
Uint16 high_side_low_res_server4 = 0; // Enable high side low resistance path for server1, initially disabled
Uint16 low_side_server4 = 0;      // Enable low side for server1, initially disabled

Uint16 closed_loop = 0;
Uint16 disable_all = 0;   // Disable all converters when = 1
Uint16 reset_all = 0;     // Disable all converters and phases and directions when = 1
Uint16 update_enbl = 0;   // Update EPWM Enable Registers when = 1
Uint16 update_ps = 0;   // Update EPWM Phase Shift Registers when = 1
Uint16 update_hotswap = 0;  // Update Hot Swap Registers when = 1
Uint16 update_st = 0;   // Update Stack Registers when = 1
Uint16 stack_enbl = 0;
Uint16 stack_disbl = 0;

Uint16 sampling_time = 100;     // sampling time = sampling_time * 5mus
Uint16 period = 200;    // period = 0.5 * TBCLK / f_desired , where
                // TBCLK = SYSCLKOUT / (HSPCLKDIV   CLKDIV)
                // SYSCLKOUT is selected in DevInit_F2806x.c
                // HSPCLKDIV and CLKDIV are selected in TBCTL register

void main(void)
{
// Step 1. Initialize Syst ipheral Clocks
// This example function is found in the F2806x_SysCtrl.c file.
   InitSysCtrl();

// Step 2. Initialize GPIO:
   InitEPwm1Gpio();   // Initialize GPIO pins for ePWMs
```

```c
    InitEPwm2Gpio();    // These functions are in the F2806x_EPwm.c file
    InitEPwm3Gpio();
    InitEPwm4Gpio();
    InitEPwm5Gpio();
    InitEPwm6Gpio();
    InitEPwm7Gpio();
    InitEPwm8Gpio();
    Initialize_GPIOs();  // For Enable signals and Debugging Toggle Pin

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

// Initialize the PIE control registers to their default state.
// The default state is all PIE interrupts disabled and flags
// are cleared.
// This function is found in the F2806x_PieCtrl.c file.

    InitPieCtrl();

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

// Initialize the PIE vector table with pointers to the shell Interrupt
// Service Routines (ISR).
// This will populate the entire table, even if the interrupt
// is not used in this example.  This is useful for debug purposes.
// The shell ISR routines are found in F2806x_DefaultIsr.c.
// This function is found in F2806x_PieVect.c.
    InitPieVectTable();

// Interrupts that are used in this example are re-mapped to
// ISR functions found within this file.

     EALLOW;                    // This is needed to write to EALLOW protected register
     PieVectTable.ADCINT1 = &adc_isr;
     EDIS;                      // This is needed to disable write to EALLOW protected registers

// Call ADC configuration
    InitAdc();
    Adc_Config();

// Step 4. Initialize all the Device Peripherals:
// This function is found in F2806x_InitPeripherals.c

    EALLOW;
    SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0;
    EDIS;

    EPwm_Config();

    EALLOW;
    SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;
    EDIS;

// Step 5. User specific code, enable interrupts

// Enable CPU int1 which is connected to CPU-Timer 0
    PieCtrlRegs.PIEIER1.bit.INTx1 = 1;      // Enable INT 1.1 in the PIE (ADCINT1 in PIE)
    IER |= M_INT1;                    // Enable CPU Interrupt 0
```

```
// Enable TINT0 in the PIE: Group 1 interrupt 7 for CPU TImer 0
//   PieCtrlRegs.PIEIER1.bit.INTx7 = 1;

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

// Step 6. infinite for loop
   for(;;)
   {
      if(disable_all == 1)
    {
        Disable_All_Converters();
        disable_all = 0;
        closed_loop = 0;
        st_1 = 0;
        st_2 = 0;
        st_3 = 0;
        st_4 = 0;
        Update_Stack();
        Swap_All_Out();
        Update_HotSwap();
    }
      else if(reset_all == 1)
    {
      Reset_All_Converters();
      reset_all = 0;
      closed_loop = 0;
        st_1 = 0;
        st_2 = 0;
        st_3 = 0;
        st_4 = 0;
        Update_Stack();
        Swap_All_Out();
        Update_HotSwap();
    }
      else if (closed_loop == 0)
      {
        if(swap_all_in_flag == 1)
        {
          if(adc_ready_flag == 1) // this if takes around 5.6 mus
        {
          Swap_All_In();
          Update_HotSwap();
            AdcRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;     // Clear ADCINT1 flag reinitialize for next
    SOC
          PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;       // Acknowledge interrupt to PIE
          adc_ready_flag = 0;
        }
        }
        if(swap_all_out_flag == 1)
      {
        Swap_All_Out();
        Update_HotSwap();
      }
        if(swap_one_in_flag == 1)
        {
          if(adc_ready_flag == 1) // this if takes around 5.6 mus
        {
          Swap_One_In();
```

```c
          Update_HotSwap();
            AdcRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;     // Clear ADCINT1 flag reinitialize for next SOC
          PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;       // Acknowledge interrupt to PIE
          adc_ready_flag = 0;
        }
        }
        if(swap_one_out_flag == 1)
      {
        Swap_One_Out();
        Update_HotSwap();
      }
        if(update_hotswap == 1)
        {
          Update_HotSwap();
          update_hotswap = 0;
        }
        if(update_enbl == 1)
      {
        Update_All_Enables();
        update_enbl = 0;
      }
        if(update_st == 1)
        {
          Update_Stack();
          update_st = 0;
        }
      if(adc_ready_flag == 1) // this if takes around 5.6 mus
      {
        Calculate_Voltages();
        AdcRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;     // Clear ADCINT1 flag reinitialize for next SOC
        PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;       // Acknowledge interrupt to PIE
        adc_ready_flag = 0;
      }
      Update_All_Phase_Shifts();
      }
      else if (closed_loop == 1)
      {
        if(stack_enbl == 1)
        {
          st_1 = 1;
          st_2 = 1;
          st_3 = 1;
          st_4 = 1;
          stack_enbl = 0;
          Update_Stack();
        }
        if(stack_disbl == 1)
      {
        st_1 = 0;
        st_2 = 0;
        st_3 = 0;
        st_4 = 0;
        stack_disbl = 0;
        Update_Stack();
      }
        if(update_st == 1)
      {
        Update_Stack();
        update_st = 0;
      }
```

```
        if(update_hotswap == 1)
      {
        Update_HotSwap();
        update_hotswap = 0;
      }
        if(adc_ready_flag == 1) // this if takes around 10 mus
      {
          if(swap_all_in_flag == 1)
          {
            Swap_All_In();
          Update_HotSwap();
        }
          if(swap_all_out_flag == 1)
        {
          Swap_All_Out();
          Update_HotSwap();
        }
          if(swap_one_in_flag == 1)
          {
            Swap_One_In();
          Update_HotSwap();
          }
          if(swap_one_out_flag == 1)
        {
          Swap_One_Out();
          Update_HotSwap();
        }

          Calculate_Voltages();
        Determine_Phi_n_Dir();
        Update_All_Phase_Shifts();
        Update_All_Enables();
        AdcRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;     // Clear ADCINT1 flag reinitialize for next SOC
        PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;       // Acknowledge interrupt to PIE
        adc_ready_flag = 0;
      }
      }
   } // End of infinite for loop

} // End of Main

interrupt void  adc_isr(void)
{
  if (adc_count >= (sampling_time - adc_divider))
  {
    VVB[adc_i] = AdcResult.ADCRESULT0;
    V4[adc_i] = AdcResult.ADCRESULT1;
    V3[adc_i] = AdcResult.ADCRESULT2;
    V2[adc_i] = AdcResult.ADCRESULT3;
    V1[adc_i] = AdcResult.ADCRESULT4;
    adc_i += 1;
  }
  if (adc_count == sampling_time - 1)
  {
    GpioDataRegs.GPBTOGGLE.bit.GPIO33 = 1;
    adc_ready_flag = 1;
    adc_count = 0;
  }
  else
  {
    AdcRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;     // Clear ADCINT1 flag reinitialize for next SOC
```

```
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;       // Acknowledge interrupt to PIE
    adc_count += 1;
  }
  return;
}

//---------------------------------------------------------------------------
// Control Functions: Below functions controls converter operations
//---------------------------------------------------------------------------

void Disable_All_Converters()
{
  enable_1_1 = 0;
  enable_1_2 = 0;
  enable_2_1 = 0;
  enable_2_2 = 0;
  enable_3_1 = 0;
  enable_3_2 = 0;
  enable_4_1 = 0;
  enable_4_2 = 0;
  Update_All_Enables();
}

void Reset_All_Converters()
{
  enable_1_1 = 0;
  enable_1_2 = 0;
  enable_2_1 = 0;
  enable_2_2 = 0;
  enable_3_1 = 0;
  enable_3_2 = 0;
  enable_4_1 = 0;
  enable_4_2 = 0;
  phi_1 = 0;
  phi_2 = 0;
  phi_3 = 0;
  phi_4 = 0;
  dir_1 = 0;
  dir_2 = 0;
  dir_3 = 0;
  dir_4 = 0;
  Update_All_Enables();
  Update_All_Phase_Shifts();
  state_s1 = 0;
  state_s1_1 = 0;
  state_s2 = 0;
  state_s2_1 = 0;
  state_s3 = 0;
  state_s3_1 = 0;
  state_s4 = 0;
  state_s4_1 = 0;
  state_vb = 0;
  state_vb_1 = 0;
}

void Update_All_Phase_Shifts()
{
  EPwm2Regs.TBPHS.half.TBPHS = phi_1;    // Add phi to watch window and change its value to adjust phase
                         // phi ~= - desired_phase / 180 * period, fine tuning may be needed
  EPwm2Regs.TBCTL.bit.PHSDIR = dir_1;    // Add phase_dir to watch window and change its value to
```

```
    change direction
                    // pahse_dir = 0 epwm1a leading, 1 epwm2a leading

  EPwm4Regs.TBPHS.half.TBPHS = phi_2;    // Add phi to watch window and change its value to adjust
    phase
                    // phi ~= - desired_phase / 180 * period, fine tuning may be needed
  EPwm4Regs.TBCTL.bit.PHSDIR = dir_2;    // Add phase_dir to watch window and change its value to
    change direction
                    // pahse_dir = 0 epwm1a leading, 1 epwm2a leading

  EPwm6Regs.TBPHS.half.TBPHS = phi_3;    // Add phi to watch window and change its value to adjust
    phase
                    // phi ~= - desired_phase / 180 * period, fine tuning may be needed
  EPwm6Regs.TBCTL.bit.PHSDIR = dir_3;    // Add phase_dir to watch window and change its value to
    change direction
                    // pahse_dir = 0 epwm1a leading, 1 epwm2a leading

  EPwm8Regs.TBPHS.half.TBPHS = phi_4;    // Add phi to watch window and change its value to adjust
    phase
                    // phi ~= - desired_phase / 180 * period, fine tuning may be needed
  EPwm8Regs.TBCTL.bit.PHSDIR = dir_4;    // Add phase_dir to watch window and change its value to
    change direction
                    // phase_dir = 0 epwm1a leading, 1 epwm2a leading
}

void Update_All_Enables()
{
  GpioDataRegs.GPBDAT.bit.GPIO50 = enable_1_1;
  GpioDataRegs.GPADAT.bit.GPIO13 = enable_1_2;
  GpioDataRegs.GPADAT.bit.GPIO13 = enable_1_2;

  GpioDataRegs.GPADAT.bit.GPIO12 = enable_2_1;
  GpioDataRegs.GPADAT.bit.GPIO12 = enable_2_1;
  GpioDataRegs.GPADAT.bit.GPIO14 = enable_2_2;
  GpioDataRegs.GPADAT.bit.GPIO14 = enable_2_2;

  GpioDataRegs.GPADAT.bit.GPIO15 = enable_3_1;
  GpioDataRegs.GPADAT.bit.GPIO25 = enable_3_2;
  GpioDataRegs.GPADAT.bit.GPIO25 = enable_3_2;

  GpioDataRegs.GPADAT.bit.GPIO24 = enable_4_1;
  GpioDataRegs.GPADAT.bit.GPIO24 = enable_4_1;
  GpioDataRegs.GPADAT.bit.GPIO27 = enable_4_2;
}

void Update_Stack()
{
  GpioDataRegs.GPADAT.bit.GPIO28 = st_1;
  GpioDataRegs.GPADAT.bit.GPIO28 = st_1;
  GpioDataRegs.GPADAT.bit.GPIO30 = st_2;
  GpioDataRegs.GPADAT.bit.GPIO30 = st_2;
  GpioDataRegs.GPBDAT.bit.GPIO32 = st_3;
  GpioDataRegs.GPBDAT.bit.GPIO32 = st_3;
  GpioDataRegs.GPBDAT.bit.GPIO34 = st_4;
  GpioDataRegs.GPBDAT.bit.GPIO34 = st_4;
}


void Update_HotSwap()
{
  GpioDataRegs.GPADAT.bit.GPIO26 = high_side_high_res_server1;
```

```c
  GpioDataRegs.GPADAT.bit.GPIO26 = high_side_high_res_server1;
  GpioDataRegs.GPADAT.bit.GPIO16 = high_side_low_res_server1;
  GpioDataRegs.GPADAT.bit.GPIO16 = high_side_low_res_server1;
  GpioDataRegs.GPADAT.bit.GPIO18 = low_side_server1;
  GpioDataRegs.GPADAT.bit.GPIO18 = low_side_server1;

  GpioDataRegs.GPADAT.bit.GPIO17 = high_side_high_res_server2;
  GpioDataRegs.GPADAT.bit.GPIO17 = high_side_high_res_server2;
  GpioDataRegs.GPADAT.bit.GPIO19 = high_side_low_res_server2;
  GpioDataRegs.GPADAT.bit.GPIO19 = high_side_low_res_server2;
  GpioDataRegs.GPADAT.bit.GPIO21 = low_side_server2;
  GpioDataRegs.GPADAT.bit.GPIO21 = low_side_server2;

  GpioDataRegs.GPADAT.bit.GPIO23 = high_side_high_res_server3;
  GpioDataRegs.GPADAT.bit.GPIO23 = high_side_high_res_server3;
  GpioDataRegs.GPADAT.bit.GPIO29 = high_side_low_res_server3;
  GpioDataRegs.GPADAT.bit.GPIO29 = high_side_low_res_server3;
  GpioDataRegs.GPADAT.bit.GPIO31 = low_side_server3;
  GpioDataRegs.GPADAT.bit.GPIO31 = low_side_server3;

  GpioDataRegs.GPADAT.bit.GPIO20 = high_side_high_res_server4;
  GpioDataRegs.GPADAT.bit.GPIO20 = high_side_high_res_server4;
  GpioDataRegs.GPADAT.bit.GPIO22 = high_side_low_res_server4;
  GpioDataRegs.GPADAT.bit.GPIO22 = high_side_low_res_server4;
  GpioDataRegs.GPBDAT.bit.GPIO51 = low_side_server4;
  GpioDataRegs.GPBDAT.bit.GPIO51 = low_side_server4;

}

void Calculate_Voltages()
{
  VVB_ave = 0;
  V1_ave = 0;
  V2_ave = 0;
  V3_ave = 0;
  V4_ave = 0;

  for (adc_i = 0; adc_i < adc_divider; adc_i++)
  {
    VVB_ave += VVB[adc_i];
    V1_ave += V1[adc_i];
    V2_ave += V2[adc_i];
    V3_ave += V3[adc_i];
    V4_ave += V4[adc_i];
  }

  V_vb = VVB_ave/adc_divider;
  V_1 = V1_ave/adc_divider;
  V_2 = V2_ave/adc_divider;
  V_3 = V3_ave/adc_divider;
  V_4 = V4_ave/adc_divider;

  V_vb = V_vb;
  V_s4 = V_4;
  V_s3 = 2 * V_3 - V_s4;
  V_s2 = 3 * V_2 - V_s4 - V_s3;
  V_s1 = 4 * V_1 - V_s4 - V_s3 - V_s2;

  //Constant DC bus
  V_s1_err = V_s1 - 2435;
  V_s2_err = V_s2 - 2379;
```

```
  V_s3_err = V_s3 - 2471;
  V_s4_err = V_s4 - 2450;
  V_vb_err = V_vb - 2482;

  //Varying DC bus
  //V_s1_err = V_s1 - V_1;
  //V_s2_err = V_s2 - V_1;
  //V_s3_err = V_s3 - V_1;
  //V_s4_err = V_s4 - V_1;
  //V_vb_err = V_vb - V_1;

  adc_i = 0;
}

void Determine_Phi_n_Dir(void)
{
//state decisions
  if (V_s1_err > V_s1_hys_highest)
  {
    if ((state_s1_1 == 1) || (state_s1_1 == 2))
    {
      state_s1 = 2;
    }
    else if ((state_s1_1 == -1) || (state_s1_1 == -2))
    {
      state_s1 = 0;
    }
    else if (state_s1_1 == 0)
    {
      state_s1 = 1;
    }
  }
  else if (V_s1_err < V_s1_hys_highest_)
  {
    if ((state_s1_1 == -1) || (state_s1_1 == -2))
    {
      state_s1 = -2;
    }
    else if ((state_s1_1 == 1) || (state_s1_1 == 2))
    {
      state_s1 = 0;
    }
    else if (state_s1_1 == 0)
    {
      state_s1 = -1;
    }
  }
  else if (V_s1_err > V_s1_hys_high)
  {
    if ((state_s1_1 == 0) || (state_s1_1 == 1))
    {
      state_s1 = 1;
    }
    else if ((state_s1_1 == -1) || (state_s1_1 == -2))
    {
      state_s1 = 0;
    }
    else if (state_s1_1 == 2)
    {
      state_s1 = 2;
    }
```

```
  }
  else if (V_s1_err < V_s1_hys_high_)
  {
    if ((state_s1_1 == 0) || (state_s1_1 == -1))
    {
      state_s1 = -1;
    }
    else if ((state_s1_1 == 1) || (state_s1_1 == 2))
    {
      state_s1 = 0;
    }
    else if (state_s1_1 == -2)
    {
      state_s1 = -2;
    }
  }
  else if (V_s1_err > V_s1_hys_low)
  {
    if ((state_s1_1 == -1) || (state_s1_1 == -2))
    {
      state_s1 = 0;
    }
    else if (state_s1_1 == 0)
    {
      state_s1 = 0;
    }
    else if (state_s1_1 == 1)
    {
      state_s1 = 1;
    }
    else if (state_s1_1 == 2)
    {
      state_s1 = 2;
    }
  }
  else if (V_s1_err < V_s1_hys_low_)
  {
    if ((state_s1_1 == 1) || (state_s1_1 == 2))
    {
      state_s1 = 0;
    }
    else if (state_s1_1 == 0)
    {
      state_s1 = 0;
    }
    else if (state_s1_1 == -1)
    {
      state_s1 = -1;
    }
    else if (state_s1_1 == -2)
    {
      state_s1 = -2;
    }
  }
  else
  {
    state_s1 = state_s1_1;
  }

  if (V_s2_err > V_s2_hys_highest)
  {
```

```
    if ((state_s2_1 == 1) || (state_s2_1 == 2))
    {
      state_s2 = 2;
    }
    else if ((state_s2_1 == -1) || (state_s2_1 == -2))
    {
      state_s2 = 0;
    }
    else if (state_s2_1 == 0)
    {
      state_s2 = 1;
    }
  }
  else if (V_s2_err < V_s2_hys_highest_)
  {
    if ((state_s2_1 == -1) || (state_s2_1 == -2))
    {
      state_s2 = -2;
    }
    else if ((state_s2_1 == 1) || (state_s2_1 == 2))
    {
      state_s2 = 0;
    }
    else if (state_s2_1 == 0)
    {
      state_s2 = -1;
    }
  }
  else if (V_s2_err > V_s2_hys_high)
  {
    if ((state_s2_1 == 0) || (state_s2_1 == 1))
    {
      state_s2 = 1;
    }
    else if ((state_s2_1 == -1) || (state_s2_1 == -2))
    {
      state_s2 = 0;
    }
    else if (state_s2_1 == 2)
    {
      state_s2 = 2;
    }
  }
  else if (V_s2_err < V_s2_hys_high_)
  {
    if ((state_s2_1 == 0) || (state_s2_1 == -1))
    {
      state_s2 = -1;
    }
    else if ((state_s2_1 == 1) || (state_s2_1 == 2))
    {
      state_s2 = 0;
    }
    else if (state_s2_1 == -2)
    {
      state_s2 = -2;
    }
  }
  else if (V_s2_err > V_s2_hys_low)
  {
    if ((state_s2_1 == -1) || (state_s2_1 == -2))
```

```
    {
      state_s2 = 0;
    }
    else if (state_s2_1 == 0)
    {
      state_s2 = 0;
    }
    else if (state_s2_1 == 1)
    {
      state_s2 = 1;
    }
    else if (state_s2_1 == 2)
    {
      state_s2 = 2;
    }
  }
  else if (V_s2_err < V_s2_hys_low_)
  {
    if ((state_s2_1 == 1) || (state_s2_1 == 2))
    {
      state_s2 = 0;
    }
    else if (state_s2_1 == 0)
    {
      state_s2 = 0;
    }
    else if (state_s2_1 == -1)
    {
      state_s2 = -1;
    }
    else if (state_s2_1 == -2)
    {
      state_s2 = -2;
    }
  }
  else
  {
    state_s2 = state_s2_1;
  }

  if (V_s3_err > V_s3_hys_highest)
  {
    if ((state_s3_1 == 1) || (state_s3_1 == 2))
    {
      state_s3 = 2;
    }
    else if ((state_s3_1 == -1) || (state_s3_1 == -2))
    {
      state_s3 = 0;
    }
    else if (state_s3_1 == 0)
    {
      state_s3 = 1;
    }
  }
  else if (V_s3_err < V_s3_hys_highest_)
  {
    if ((state_s3_1 == -1) || (state_s3_1 == -2))
    {
      state_s3 = -2;
    }
```

```
    else if ((state_s3_1 == 1) || (state_s3_1 == 2))
    {
      state_s3 = 0;
    }
    else if (state_s3_1 == 0)
    {
      state_s3 = -1;
    }
  }
  else if (V_s3_err > V_s3_hys_high)
  {
    if ((state_s3_1 == 0) || (state_s3_1 == 1))
    {
      state_s3 = 1;
    }
    else if ((state_s3_1 == -1) || (state_s3_1 == -2))
    {
      state_s3 = 0;
    }
    else if (state_s3_1 == 2)
    {
      state_s3 = 2;
    }
  }
  else if (V_s3_err < V_s3_hys_high_)
  {
    if ((state_s3_1 == 0) || (state_s3_1 == -1))
    {
      state_s3 = -1;
    }
    else if ((state_s3_1 == 1) || (state_s3_1 == 2))
    {
      state_s3 = 0;
    }
    else if (state_s3_1 == -2)
    {
      state_s3 = -2;
    }
  }
  else if (V_s3_err > V_s3_hys_low)
  {
    if ((state_s3_1 == -1) || (state_s3_1 == -2))
    {
      state_s3 = 0;
    }
    else if (state_s3_1 == 0)
    {
      state_s3 = 0;
    }
    else if (state_s3_1 == 1)
    {
      state_s3 = 1;
    }
    else if (state_s3_1 == 2)
    {
      state_s3 = 2;
    }
  }
  else if (V_s3_err < V_s3_hys_low_)
  {
    if ((state_s3_1 == 1) || (state_s3_1 == 2))
```

```
  {
    state_s3 = 0;
  }
  else if (state_s3_1 == 0)
  {
    state_s3 = 0;
  }
  else if (state_s3_1 == -1)
  {
    state_s3 = -1;
  }
  else if (state_s3_1 == -2)
  {
    state_s3 = -2;
  }
}
else
{
  state_s3 = state_s3_1;
}

if (V_s4_err > V_s4_hys_highest)
{
  if ((state_s4_1 == 1) || (state_s4_1 == 2))
  {
    state_s4 = 2;
  }
  else if ((state_s4_1 == -1) || (state_s4_1 == -2))
  {
    state_s4 = 0;
  }
  else if (state_s4_1 == 0)
  {
    state_s4 = 1;
  }
}
else if (V_s4_err < V_s4_hys_highest_)
{
  if ((state_s4_1 == -1) || (state_s4_1 == -2))
  {
    state_s4 = -2;
  }
  else if ((state_s4_1 == 1) || (state_s4_1 == 2))
  {
    state_s4 = 0;
  }
  else if (state_s4_1 == 0)
  {
    state_s4 = -1;
  }
}
else if (V_s4_err > V_s4_hys_high)
{
  if ((state_s4_1 == 0) || (state_s4_1 == 1))
  {
    state_s4 = 1;
  }
  else if ((state_s4_1 == -1) || (state_s4_1 == -2))
  {
    state_s4 = 0;
  }
```

```
      else if (state_s4_1 == 2)
      {
        state_s4 = 2;
      }
    }
    else if (V_s4_err < V_s4_hys_high_)
    {
      if ((state_s4_1 == 0) || (state_s4_1 == -1))
      {
        state_s4 = -1;
      }
      else if ((state_s4_1 == 1) || (state_s4_1 == 2))
      {
        state_s4 = 0;
      }
      else if (state_s4_1 == -2)
      {
        state_s4 = -2;
      }
    }
    else if (V_s4_err > V_s4_hys_low)
    {
      if ((state_s4_1 == -1) || (state_s4_1 == -2))
      {
        state_s4 = 0;
      }
      else if (state_s4_1 == 0)
      {
        state_s4 = 0;
      }
      else if (state_s4_1 == 1)
      {
        state_s4 = 1;
      }
      else if (state_s4_1 == 2)
      {
        state_s4 = 2;
      }
    }
    else if (V_s4_err < V_s4_hys_low_)
    {
      if ((state_s4_1 == 1) || (state_s4_1 == 2))
      {
        state_s4 = 0;
      }
      else if (state_s4_1 == 0)
      {
        state_s4 = 0;
      }
      else if (state_s4_1 == -1)
      {
        state_s4 = -1;
      }
      else if (state_s4_1 == -2)
      {
        state_s4 = -2;
      }
    }
    else
    {
      state_s4 = state_s4_1;
```

```
}


if (V_vb_err > V_vb_hys_highest)
{
  if ((state_vb_1 == 1) || (state_vb_1 == 2))
  {
    state_vb = 2;
  }
  else if ((state_vb_1 == -1) || (state_vb_1 == -2))
  {
    state_vb = 0;
  }
  else if (state_vb_1 == 0)
  {
    state_vb = 1;
  }
}
else if (V_vb_err < V_vb_hys_highest_)
{
  if ((state_vb_1 == -1) || (state_vb_1 == -2))
  {
    state_vb = -2;
  }
  else if ((state_vb_1 == 1) || (state_vb_1 == 2))
  {
    state_vb = 0;
  }
  else if (state_vb_1 == 0)
  {
    state_vb = -1;
  }
}
else if (V_vb_err > V_vb_hys_high)
{
  if ((state_vb_1 == 0) || (state_vb_1 == 1))
  {
    state_vb = 1;
  }
  else if ((state_vb_1 == -1) || (state_vb_1 == -2))
  {
    state_vb = 0;
  }
  else if (state_vb_1 == 2)
  {
    state_vb = 2;
  }
}
else if (V_vb_err < V_vb_hys_high_)
{
  if ((state_vb_1 == 0) || (state_vb_1 == -1))
  {
    state_vb = -1;
  }
  else if ((state_vb_1 == 1) || (state_vb_1 == 2))
  {
    state_vb = 0;
  }
  else if (state_vb_1 == -2)
  {
    state_vb = -2;
```

```
    }
  }
  else if (V_vb_err > V_vb_hys_low)
  {
    if ((state_vb_1 == -1) || (state_vb_1 == -2))
    {
      state_vb = 0;
    }
    else if (state_vb_1 == 0)
    {
      state_vb = 0;
    }
    else if (state_vb_1 == 1)
    {
      state_vb = 1;
    }
    else if (state_vb_1 == 2)
    {
      state_vb = 2;
    }
  }
  else if (V_vb_err < V_vb_hys_low_)
  {
    if ((state_vb_1 == 1) || (state_vb_1 == 2))
    {
      state_vb = 0;
    }
    else if (state_vb_1 == 0)
    {
      state_vb = 0;
    }
    else if (state_vb_1 == -1)
    {
      state_vb = -1;
    }
    else if (state_vb_1 == -2)
    {
      state_vb = -2;
    }
  }
  else
  {
    state_vb = state_vb_1;
  }

//end of state decisions
//dpp1 decision
  if (state_s1 == 1 && (state_vb == 0 || state_vb == -1 || state_vb == -2 || state_vb == 2))
  {
    enable_1_1 = 1;
    enable_1_2 = 1;
    phi_1 = phi_1_x_prime;
    dir_1 = 1;
  }
  else if (state_s1 == -1 && (state_vb == 0 || state_vb == 1 || state_vb == 2 || state_vb == -2))
  {
    enable_1_1 = 1;
    enable_1_2 = 1;
    phi_1 = phi_1_x;
    dir_1 = 0;
  }
```

```
else if (state_s1 == 2 && (state_vb == 0 || state_vb == -1 || state_vb == -2))
{
  enable_1_1 = 1;
  enable_1_2 = 1;
  phi_1 = phi_1_xx_prime;
  dir_1 = 1;
}
else if (state_s1 == 2 && (state_vb == 1))
{
  enable_1_1 = 1;
  enable_1_2 = 1;
  phi_1 = phi_1_x_prime;
  dir_1 = 1;
}
else if (state_s1 == -2 && (state_vb == 0 || state_vb == 1 || state_vb == 2))
{
  enable_1_1 = 1;
  enable_1_2 = 1;
  phi_1 = phi_1_xx;
  dir_1 = 0;
}
else if (state_s1 == -2 && (state_vb == -1))
{
  enable_1_1 = 1;
  enable_1_2 = 1;
  phi_1 = phi_1_x;
  dir_1 = 0;
}
else if ((state_s1 == 0 && state_vb == 0) || (state_s1 == 1 && state_vb == 1) || (state_s1 == 2 &&
    state_vb == 2) || (state_s1 == -1 && state_vb == -1) || (state_s1 == -2 && state_vb == -2))
{
  enable_1_1 = 0;
  enable_1_2 = 0;
  phi_1 = 0;
  dir_1 = 0;
}
else if (state_s1 == 0 && state_vb == 1)
{
  enable_1_1 = 1;
  enable_1_2 = 1;
  phi_1 = phi_1_x;
  dir_1 = 0;
}
else if (state_s1 == 0 && state_vb == -1)
{
  enable_1_1 = 1;
  enable_1_2 = 1;
  phi_1 = phi_1_x_prime;
  dir_1 = 1;
}
else if (state_s1 == 0 && state_vb == 2)
{
  enable_1_1 = 1;
  enable_1_2 = 1;
  phi_1 = phi_1_xx;
  dir_1 = 0;
}
else if (state_s1 == 0 && state_vb == -2)
{
  enable_1_1 = 1;
  enable_1_2 = 1;
```

```
    phi_1 = phi_1_xx_prime;
    dir_1 = 1;
  }
//end of dpp1 decision
//dpp2 decision
  if (state_s2 == 1 && (state_vb == 0 || state_vb == -1 || state_vb == -2 || state_vb == 2))
  {
    enable_2_1 = 1;
    enable_2_2 = 1;
    phi_2 = phi_2_x_prime;
    dir_2 = 1;
  }
  else if (state_s2 == -1 && (state_vb == 0 || state_vb == 1 || state_vb == 2 || state_vb == -2))
  {
    enable_2_1 = 1;
    enable_2_2 = 1;
    phi_2 = phi_2_x;
    dir_2 = 0;
  }
  else if (state_s2 == 2 && (state_vb == 0 || state_vb == -1 || state_vb == -2))
  {
    enable_2_1 = 1;
    enable_2_2 = 1;
    phi_2 = phi_2_xx_prime;
    dir_2 = 1;
  }
  else if (state_s2 == 2 && (state_vb == 1))
  {
    enable_2_1 = 1;
    enable_2_2 = 1;
    phi_2 = phi_2_x_prime;
    dir_2 = 1;
  }
  else if (state_s2 == -2 && (state_vb == 0 || state_vb == 1 || state_vb == 2))
  {
    enable_2_1 = 1;
    enable_2_2 = 1;
    phi_2 = phi_2_xx;
    dir_2 = 0;
  }
  else if (state_s2 == -2 && (state_vb == -1))
  {
    enable_2_1 = 1;
    enable_2_2 = 1;
    phi_2 = phi_2_x;
    dir_2 = 0;
  }
  else if ((state_s2 == 0 && state_vb == 0) || (state_s2 == 1 && state_vb == 1) || (state_s2 == 2 &&
     state_vb == 2) || (state_s2 == -1 && state_vb == -1) || (state_s2 == -2 && state_vb == -2))
  {
    enable_2_1 = 0;
    enable_2_2 = 0;
    phi_2 = 0;
    dir_2 = 0;
  }
  else if (state_s2 == 0 && state_vb == 1)
  {
    enable_2_1 = 1;
    enable_2_2 = 1;
    phi_2 = phi_2_x;
    dir_2 = 0;
```

```
  }
  else if (state_s2 == 0 && state_vb == -1)
  {
    enable_2_1 = 1;
    enable_2_2 = 1;
    phi_2 = phi_2_x_prime;
    dir_2 = 1;
  }
  else if (state_s2 == 0 && state_vb == 2)
  {
    enable_2_1 = 1;
    enable_2_2 = 1;
    phi_2 = phi_2_xx;
    dir_2 = 0;
  }
  else if (state_s2 == 0 && state_vb == -2)
  {
    enable_2_1 = 1;
    enable_2_2 = 1;
    phi_2 = phi_2_xx_prime;
    dir_2 = 1;
  }
//end of dpp2 decision
//dpp3 decision
  if (state_s3 == 1 && (state_vb == 0 || state_vb == -1 || state_vb == -2 || state_vb == 2))
  {
    enable_3_1 = 1;
    enable_3_2 = 1;
    phi_3 = phi_3_x_prime;
    dir_3 = 1;
  }
  else if (state_s3 == -1 && (state_vb == 0 || state_vb == 1 || state_vb == 2 || state_vb == -2))
  {
    enable_3_1 = 1;
    enable_3_2 = 1;
    phi_3 = phi_3_x;
    dir_3 = 0;
  }
  else if (state_s3 == 2 && (state_vb == 0 || state_vb == -1 || state_vb == -2))
  {
    enable_3_1 = 1;
    enable_3_2 = 1;
    phi_3 = phi_3_xx_prime;
    dir_3 = 1;
  }
  else if (state_s3 == 2 && (state_vb == 1))
  {
    enable_3_1 = 1;
    enable_3_2 = 1;
    phi_3 = phi_3_x_prime;
    dir_3 = 1;
  }
  else if (state_s3 == -2 && (state_vb == 0 || state_vb == 1 || state_vb == 2))
  {
    enable_3_1 = 1;
    enable_3_2 = 1;
    phi_3 = phi_3_xx;
    dir_3 = 0;
  }
  else if (state_s3 == -2 && (state_vb == -1))
  {
```

```
    enable_3_1 = 1;
    enable_3_2 = 1;
    phi_3 = phi_3_x;
    dir_3 = 0;
  }
  else if ((state_s3 == 0 && state_vb == 0) || (state_s3 == 1 && state_vb == 1) || (state_s3 == 2 &&
     state_vb == 2) || (state_s3 == -1 && state_vb == -1) || (state_s3 == -2 && state_vb == -2))
  {
    enable_3_1 = 0;
    enable_3_2 = 0;
    phi_3 = 0;
    dir_3 = 0;
  }
  else if (state_s3 == 0 && state_vb == 1)
  {
    enable_3_1 = 1;
    enable_3_2 = 1;
    phi_3 = phi_3_x;
    dir_3 = 0;
  }
  else if (state_s3 == 0 && state_vb == -1)
  {
    enable_3_1 = 1;
    enable_3_2 = 1;
    phi_3 = phi_3_x_prime;
    dir_3 = 1;
  }
  else if (state_s3 == 0 && state_vb == 2)
  {
    enable_3_1 = 1;
    enable_3_2 = 1;
    phi_3 = phi_3_xx;
    dir_3 = 0;
  }
  else if (state_s3 == 0 && state_vb == -2)
  {
    enable_3_1 = 1;
    enable_3_2 = 1;
    phi_3 = phi_3_xx_prime;
    dir_3 = 1;
  }
//end of dpp3 decision
//dpp4 decision
  if (state_s4 == 1 && (state_vb == 0 || state_vb == -1 || state_vb == -2 || state_vb == 2))
  {
    enable_4_1 = 1;
    enable_4_2 = 1;
    phi_4 = phi_4_x_prime;
    dir_4 = 1;
  }
  else if (state_s4 == -1 && (state_vb == 0 || state_vb == 1 || state_vb == 2 || state_vb == -2))
  {
    enable_4_1 = 1;
    enable_4_2 = 1;
    phi_4 = phi_4_x;
    dir_4 = 0;
  }
  else if (state_s4 == 2 && (state_vb == 0 || state_vb == -1 || state_vb == -2))
  {
    enable_4_1 = 1;
    enable_4_2 = 1;
```

```
    phi_4 = phi_4_xx_prime;
    dir_4 = 1;
  }
  else if (state_s4 == 2 && (state_vb == 1))
  {
    enable_4_1 = 1;
    enable_4_2 = 1;
    phi_4 = phi_4_x_prime;
    dir_4 = 1;
  }
  else if (state_s4 == -2 && (state_vb == 0 || state_vb == 1 || state_vb == 2))
  {
    enable_4_1 = 1;
    enable_4_2 = 1;
    phi_4 = phi_4_xx;
    dir_4 = 0;
  }
  else if (state_s4 == -2 && (state_vb == -1))
  {
    enable_4_1 = 1;
    enable_4_2 = 1;
    phi_4 = phi_4_x;
    dir_4 = 0;
  }
  else if ((state_s4 == 0 && state_vb == 0) || (state_s4 == 1 && state_vb == 1) || (state_s4 == 2 &&
     state_vb == 2) || (state_s4 == -1 && state_vb == -1) || (state_s4 == -2 && state_vb == -2))
  {
    enable_4_1 = 0;
    enable_4_2 = 0;
    phi_4 = 0;
    dir_4 = 0;
  }
  else if (state_s4 == 0 && state_vb == 1)
  {
    enable_4_1 = 1;
    enable_4_2 = 1;
    phi_4 = phi_4_x;
    dir_4 = 0;
  }
  else if (state_s4 == 0 && state_vb == -1)
  {
    enable_4_1 = 1;
    enable_4_2 = 1;
    phi_4 = phi_4_x_prime;
    dir_4 = 1;
  }
  else if (state_s4 == 0 && state_vb == 2)
  {
    enable_4_1 = 1;
    enable_4_2 = 1;
    phi_4 = phi_4_xx;
    dir_4 = 0;
  }
  else if (state_s4 == 0 && state_vb == -2)
  {
    enable_4_1 = 1;
    enable_4_2 = 1;
    phi_4 = phi_4_xx_prime;
    dir_4 = 1;
  }
//end of dpp4 decision
```

```
  state_s1_1 = state_s1;
  state_s2_1 = state_s2;
  state_s3_1 = state_s3;
  state_s4_1 = state_s4;
  state_vb_1 = state_vb;
}

void Swap_One_In(void)
{
  if(swap_one_counter == 2)
  {
    high_side_high_res_server2 = 1;
    high_side_low_res_server2 = 0;
    low_side_server2 = 1;
    swap_one_counter += 1;
  }
  else if (swap_one_counter == 1*swap_one_multip)
  {
    high_side_high_res_server2 = 1;
    high_side_low_res_server2 = 1;
    low_side_server2 = 1;

    swap_one_counter += 1;
  }
  else if (swap_one_counter == 3*swap_one_multip)
  {
    high_side_high_res_server2 = 0;
    high_side_low_res_server2 = 1;
    low_side_server2 = 1;

    swap_one_counter = 1;
    swap_one_in_flag = 0;
  }
  else
  {
    swap_one_counter += 1;
  }
}

void Swap_All_In(void)
{
  if(swap_all_counter == 2)
  {
    high_side_high_res_server1 = 1;
    high_side_low_res_server1 = 0;
    low_side_server1 = 1;

    high_side_high_res_server2 = 1;
    high_side_low_res_server2 = 0;
    low_side_server2 = 1;

    high_side_high_res_server3 = 1;
    high_side_low_res_server3 = 0;
    low_side_server3 = 1;

    high_side_high_res_server4 = 1;
    high_side_low_res_server4 = 0;
    low_side_server4 = 1;

    swap_all_counter += 1;
```

```
  }
  else if (swap_all_counter == 1*swap_all_multip)
  {
    high_side_high_res_server1 = 1;
    high_side_low_res_server1 = 1;
    low_side_server1 = 1;

    high_side_high_res_server2 = 1;
    high_side_low_res_server2 = 1;
    low_side_server2 = 1;

    high_side_high_res_server3 = 1;
    high_side_low_res_server3 = 1;
    low_side_server3 = 1;

    high_side_high_res_server4 = 1;
    high_side_low_res_server4 = 1;
    low_side_server4 = 1;

    swap_all_counter += 1;
  }
  else if (swap_all_counter == 3*swap_all_multip)
  {
    high_side_high_res_server1 = 0;
    high_side_low_res_server1 = 1;
    low_side_server1 = 1;

    high_side_high_res_server2 = 0;
    high_side_low_res_server2 = 1;
    low_side_server2 = 1;

    high_side_high_res_server3 = 0;
    high_side_low_res_server3 = 1;
    low_side_server3 = 1;

    high_side_high_res_server4 = 0;
    high_side_low_res_server4 = 1;
    low_side_server4 = 1;

    swap_all_counter = 1;
    swap_all_in_flag = 0;
  }
  else
  {
    swap_all_counter += 1;
  }
}

void Swap_One_Out(void)
{
  high_side_high_res_server2 = 0;
  high_side_low_res_server2 = 0;
  low_side_server2 = 0;

  swap_one_out_flag = 0;
}

void Swap_All_Out(void)
{
  high_side_high_res_server1 = 0;
  high_side_low_res_server1 = 0;
```

```
  low_side_server1 = 0;

  high_side_high_res_server2 = 0;
  high_side_low_res_server2 = 0;
  low_side_server2 = 0;

  high_side_high_res_server3 = 0;
  high_side_low_res_server3 = 0;
  low_side_server3 = 0;

  high_side_high_res_server4 = 0;
  high_side_low_res_server4 = 0;
  low_side_server4 = 0;

  swap_all_out_flag = 0;
}

//--------------------------------------------------------------------------
// Configuration Functions: Below functions initialize GPIOs and configures ADC and EPWM
//--------------------------------------------------------------------------

void Initialize_GPIOs()
{
  EALLOW; // following registers are protected
//  GPIO-50 (Pin#86 in experimenter board) - PIN FUNCTION = Enable Signal for DPP_1 pri switches
    GpioCtrlRegs.GPBMUX2.bit.GPIO50 = 0;  // 0 = GPIO
    GpioCtrlRegs.GPBDIR.bit.GPIO50 = 1;   // 1 = OUTput,  0 = INput
    GpioDataRegs.GPBCLEAR.bit.GPIO50 = 1; // uncomment if --> Set Low initially
//    GpioDataRegs.GPBSET.bit.GPIO50 = 1; // uncomment if --> Set High initially
//-------------------------------------------------------------------------------
//  GPIO-13 - PIN FUNCTION = Enable Signal for DPP_1 sec switches
    GpioCtrlRegs.GPAMUX1.bit.GPIO13 = 0;  // 0=GPIO
    GpioCtrlRegs.GPADIR.bit.GPIO13 = 1;   // 1=OUTput,  0=INput
    GpioDataRegs.GPACLEAR.bit.GPIO13 = 1; // uncomment if --> Set Low initially
//    GpioDataRegs.GPASET.bit.GPIO13 = 1; // uncomment if --> Set High initially
//-------------------------------------------------------------------------------
//  GPIO-12 - PIN FUNCTION = Enable Signal for DPP_2 pri switches
  GpioCtrlRegs.GPAMUX1.bit.GPIO12 = 0;  // 0 = GPIO
  GpioCtrlRegs.GPADIR.bit.GPIO12 = 1;   // 1 = OUTput,  0 = INput
  GpioDataRegs.GPACLEAR.bit.GPIO12 = 1; // uncomment if --> Set Low initially
//    GpioDataRegs.GPASET.bit.GPIO12 = 1; // uncomment if --> Set High initially
//-------------------------------------------------------------------------------
//  GPIO-14 - PIN FUNCTION = Enable Signal for DPP_2 sec switches
  GpioCtrlRegs.GPAMUX1.bit.GPIO14 = 0;  // 0=GPIO
  GpioCtrlRegs.GPADIR.bit.GPIO14 = 1;   // 1=OUTput,  0=INput
  GpioDataRegs.GPACLEAR.bit.GPIO14 = 1; // uncomment if --> Set Low initially
//    GpioDataRegs.GPASET.bit.GPIO14 = 1; // uncomment if --> Set High initially
//-------------------------------------------------------------------------------
//  GPIO-15 - PIN FUNCTION = Enable Signal for DPP_3 pri switches
  GpioCtrlRegs.GPAMUX1.bit.GPIO15 = 0;  // 0 = GPIO
  GpioCtrlRegs.GPADIR.bit.GPIO15 = 1;   // 1 = OUTput,  0 = INput
  GpioDataRegs.GPACLEAR.bit.GPIO15 = 1; // uncomment if --> Set Low initially
//    GpioDataRegs.GPASET.bit.GPIO15 = 1; // uncomment if --> Set High initially
//-------------------------------------------------------------------------------
//  GPIO-25 - PIN FUNCTION = Enable Signal for DPP_3 sec switches
  GpioCtrlRegs.GPAMUX2.bit.GPIO25 = 0;  // 0=GPIO
  GpioCtrlRegs.GPADIR.bit.GPIO25 = 1;   // 1=OUTput,  0=INput
  GpioDataRegs.GPACLEAR.bit.GPIO25 = 1; // uncomment if --> Set Low initially
//    GpioDataRegs.GPASET.bit.GPIO25 = 1; // uncomment if --> Set High initially
//-------------------------------------------------------------------------------
//  GPIO-24 - PIN FUNCTION = Enable Signal for DPP_4 pri switches
```

```
  GpioCtrlRegs.GPAMUX2.bit.GPIO24 = 0;  // 0 = GPIO
  GpioCtrlRegs.GPADIR.bit.GPIO24 = 1;   // 1 = OUTput,  0 = INput
  GpioDataRegs.GPACLEAR.bit.GPIO24 = 1; // uncomment if --> Set Low initially
//    GpioDataRegs.GPASET.bit.GPIO24 = 1; // uncomment if --> Set High initially
//--------------------------------------------------------------------------------------------
//  GPIO-27 - PIN FUNCTION = Enable Signal for DPP_4 sec switches
  GpioCtrlRegs.GPAMUX2.bit.GPIO27 = 0;  // 0=GPIO
  GpioCtrlRegs.GPADIR.bit.GPIO27 = 1;   // 1=OUTput,  0=INput
  GpioDataRegs.GPACLEAR.bit.GPIO27 = 1; // uncomment if --> Set Low initially
//    GpioDataRegs.GPASET.bit.GPIO27 = 1; // uncomment if --> Set High initially
//--------------------------------------------------------------------------------------------
//  GPIO-33 - PIN FUNCTION = GPIO Testing, Toggle
  GpioCtrlRegs.GPBMUX1.bit.GPIO33 = 0;  // 0=GPIO,  1=I2C-SDA,  2=SYNCI,  3=ADCSOCA
  GpioCtrlRegs.GPBDIR.bit.GPIO33 = 1;   // 1=OUTput,  0=INput
//  GpioDataRegs.GPBCLEAR.bit.GPIO33 = 1; // uncomment if --> Set Low initially
//  GpioDataRegs.GPBSET.bit.GPIO33 = 0;   // uncomment if --> Set High initially
//--------------------------------------------------------------------------------------------
//  GPIO-26 - PIN FUNCTION = Enable Signal for High Side High Res HotSwap - Server1
  GpioCtrlRegs.GPAMUX2.bit.GPIO26 = 0;  // 0=GPIO,  1=I2C-SDA,  2=SYNCI,  3=ADCSOCA
  GpioCtrlRegs.GPADIR.bit.GPIO26 = 1;   // 1=OUTput,  0=INput
  GpioDataRegs.GPACLEAR.bit.GPIO26 = 1; // uncomment if --> Set Low initially
//  GpioDataRegs.GPASET.bit.GPIO26 = 0;   // uncomment if --> Set High initially
//-------------------------------------------------------------------------------------------
//  GPIO-16 - PIN FUNCTION = Enable Signal for High Side Low Res HotSwap - Server1
  GpioCtrlRegs.GPAMUX2.bit.GPIO16 = 0;  // 0=GPIO,  1=I2C-SDA,  2=SYNCI,  3=ADCSOCA
  GpioCtrlRegs.GPADIR.bit.GPIO16 = 1;   // 1=OUTput,  0=INput
  GpioDataRegs.GPACLEAR.bit.GPIO16 = 1; // uncomment if --> Set Low initially
//  GpioDataRegs.GPASET.bit.GPIO16 = 0;   // uncomment if --> Set High initially
//--------------------------------------------------------------------------------------------
//  GPIO-18 - PIN FUNCTION = Enable Signal for Low Side HotSwap - Server1
  GpioCtrlRegs.GPAMUX2.bit.GPIO18 = 0;  // 0=GPIO,  1=I2C-SDA,  2=SYNCI,  3=ADCSOCA
  GpioCtrlRegs.GPADIR.bit.GPIO18 = 1;   // 1=OUTput,  0=INput
  GpioDataRegs.GPACLEAR.bit.GPIO18 = 1; // uncomment if --> Set Low initially
//  GpioDataRegs.GPASET.bit.GPIO18 = 0;   // uncomment if --> Set High initially
//--------------------------------------------------------------------------------------------
//  GPIO-17 - PIN FUNCTION = Enable Signal for High Side High Res HotSwap - Server2
  GpioCtrlRegs.GPAMUX2.bit.GPIO17 = 0;  // 0=GPIO,  1=I2C-SDA,  2=SYNCI,  3=ADCSOCA
  GpioCtrlRegs.GPADIR.bit.GPIO17 = 1;   // 1=OUTput,  0=INput
  GpioDataRegs.GPACLEAR.bit.GPIO17 = 1; // uncomment if --> Set Low initially
//  GpioDataRegs.GPASET.bit.GPIO17 = 0;   // uncomment if --> Set High initially
//-------------------------------------------------------------------------------------------
//  GPIO-19 - PIN FUNCTION = Enable Signal for High Side Low Res HotSwap - Server2
  GpioCtrlRegs.GPAMUX2.bit.GPIO19 = 0;  // 0=GPIO,  1=I2C-SDA,  2=SYNCI,  3=ADCSOCA
  GpioCtrlRegs.GPADIR.bit.GPIO19 = 1;   // 1=OUTput,  0=INput
  GpioDataRegs.GPACLEAR.bit.GPIO19 = 1; // uncomment if --> Set Low initially
//  GpioDataRegs.GPASET.bit.GPIO19 = 0;   // uncomment if --> Set High initially
//--------------------------------------------------------------------------------------------
//  GPIO-21 - PIN FUNCTION = Enable Signal for Low Side HotSwap - Server2
  GpioCtrlRegs.GPAMUX2.bit.GPIO21 = 0;  // 0=GPIO,  1=I2C-SDA,  2=SYNCI,  3=ADCSOCA
  GpioCtrlRegs.GPADIR.bit.GPIO21 = 1;   // 1=OUTput,  0=INput
  GpioDataRegs.GPACLEAR.bit.GPIO21 = 1; // uncomment if --> Set Low initially
//  GpioDataRegs.GPASET.bit.GPIO21 = 0;   // uncomment if --> Set High initially
//--------------------------------------------------------------------------------------------
//  GPIO-23 - PIN FUNCTION = Enable Signal for High Side High Res HotSwap - Server3
  GpioCtrlRegs.GPAMUX2.bit.GPIO23 = 0;  // 0=GPIO,  1=I2C-SDA,  2=SYNCI,  3=ADCSOCA
  GpioCtrlRegs.GPADIR.bit.GPIO23 = 1;   // 1=OUTput,  0=INput
  GpioDataRegs.GPACLEAR.bit.GPIO23 = 1; // uncomment if --> Set Low initially
//  GpioDataRegs.GPASET.bit.GPIO23 = 0;   // uncomment if --> Set High initially
//-------------------------------------------------------------------------------------------
//  GPIO-29 - PIN FUNCTION = Enable Signal for High Side Low Res HotSwap - Server3
  GpioCtrlRegs.GPAMUX2.bit.GPIO29 = 0;  // 0=GPIO,  1=I2C-SDA,  2=SYNCI,  3=ADCSOCA
```

```
  GpioCtrlRegs.GPADIR.bit.GPIO29 = 1;    // 1=OUTput,  0=INput
  GpioDataRegs.GPACLEAR.bit.GPIO29 = 1; // uncomment if --> Set Low initially
//  GpioDataRegs.GPASET.bit.GPIO29 = 0;    // uncomment if --> Set High initially
//-------------------------------------------------------------------------------------
//  GPIO-31 - PIN FUNCTION = Enable Signal for Low Side HotSwap - Server3
  GpioCtrlRegs.GPAMUX2.bit.GPIO31 = 0;  // 0=GPIO,  1=I2C-SDA,  2=SYNCI,  3=ADCSOCA
  GpioCtrlRegs.GPADIR.bit.GPIO31 = 1;    // 1=OUTput,  0=INput
  GpioDataRegs.GPACLEAR.bit.GPIO31 = 1; // uncomment if --> Set Low initially
//  GpioDataRegs.GPASET.bit.GPIO31 = 0;    // uncomment if --> Set High initially
//-------------------------------------------------------------------------------------
//  GPIO-20 - PIN FUNCTION = Enable Signal for High Side High Res HotSwap - Server4
  GpioCtrlRegs.GPAMUX2.bit.GPIO20 = 0;  // 0=GPIO,  1=I2C-SDA,  2=SYNCI,  3=ADCSOCA
  GpioCtrlRegs.GPADIR.bit.GPIO20 = 1;    // 1=OUTput,  0=INput
  GpioDataRegs.GPACLEAR.bit.GPIO20 = 1; // uncomment if --> Set Low initially
//  GpioDataRegs.GPASET.bit.GPIO20 = 0;    // uncomment if --> Set High initially
//-------------------------------------------------------------------------------------
//  GPIO-22 - PIN FUNCTION = Enable Signal for High Side Low Res HotSwap - Server4
  GpioCtrlRegs.GPAMUX2.bit.GPIO22 = 0;  // 0=GPIO,  1=I2C-SDA,  2=SYNCI,  3=ADCSOCA
  GpioCtrlRegs.GPADIR.bit.GPIO22 = 1;    // 1=OUTput,  0=INput
  GpioDataRegs.GPACLEAR.bit.GPIO22 = 1; // uncomment if --> Set Low initially
//  GpioDataRegs.GPASET.bit.GPIO22 = 0;    // uncomment if --> Set High initially
//-------------------------------------------------------------------------------------
//  GPIO-87 - PIN FUNCTION = Enable Signal for Low Side HotSwap - Server4
  GpioCtrlRegs.GPBMUX2.bit.GPIO51 = 0;  // 0=GPIO,  1=I2C-SDA,  2=SYNCI,  3=ADCSOCA
  GpioCtrlRegs.GPBDIR.bit.GPIO51 = 1;    // 1=OUTput,  0=INput
  GpioDataRegs.GPBCLEAR.bit.GPIO51 = 1; // uncomment if --> Set Low initially
//  GpioDataRegs.GPASET.bit.GPIO51 = 0;    // uncomment if --> Set High initially
//-------------------------------------------------------------------------------------
//  GPIO-28 - PIN FUNCTION = Enable Signal for Stack 1
  GpioCtrlRegs.GPAMUX2.bit.GPIO28 = 0;  // 0=GPIO,  1=I2C-SDA,  2=SYNCI,  3=ADCSOCA
  GpioCtrlRegs.GPADIR.bit.GPIO28 = 1;    // 1=OUTput,  0=INput
  GpioDataRegs.GPACLEAR.bit.GPIO28 = 1; // uncomment if --> Set Low initially
//  GpioDataRegs.GPASET.bit.GPIO28 = 0;    // uncomment if --> Set High initially
//-------------------------------------------------------------------------------------
//  GPIO-30 - PIN FUNCTION = Enable Signal for Stack 2
  GpioCtrlRegs.GPAMUX2.bit.GPIO30 = 0;  // 0=GPIO,  1=I2C-SDA,  2=SYNCI,  3=ADCSOCA
  GpioCtrlRegs.GPADIR.bit.GPIO30 = 1;    // 1=OUTput,  0=INput
  GpioDataRegs.GPACLEAR.bit.GPIO30 = 1; // uncomment if --> Set Low initially
//  GpioDataRegs.GPASET.bit.GPIO30 = 0;    // uncomment if --> Set High initially
//-------------------------------------------------------------------------------------
//  GPIO-32 - PIN FUNCTION = Enable Signal for Stack 3
  GpioCtrlRegs.GPBMUX1.bit.GPIO32 = 0;  // 0=GPIO,  1=I2C-SDA,  2=SYNCI,  3=ADCSOCA
  GpioCtrlRegs.GPBDIR.bit.GPIO32 = 1;    // 1=OUTput,  0=INput
  GpioDataRegs.GPBCLEAR.bit.GPIO32 = 1; // uncomment if --> Set Low initially
//  GpioDataRegs.GPBSET.bit.GPIO32 = 0;    // uncomment if --> Set High initially
//-------------------------------------------------------------------------------------
//  GPIO-34 - PIN FUNCTION = Enable Signal for Stack 4
  GpioCtrlRegs.GPBMUX1.bit.GPIO34 = 0;  // 0=GPIO,  1=I2C-SDA,  2=SYNCI,  3=ADCSOCA
  GpioCtrlRegs.GPBDIR.bit.GPIO34 = 1;    // 1=OUTput,  0=INput
  GpioDataRegs.GPBCLEAR.bit.GPIO34 = 1; // uncomment if --> Set Low initially
//  GpioDataRegs.GPBSET.bit.GPIO34 = 0;    // uncomment if --> Set High initially
  EDIS;
}

void Adc_Config()
{
  EALLOW;
  AdcRegs.ADCCTL2.bit.ADCNONOVERLAP = 1;       // Enable non-overlap mode. This will eliminate 1st sample issue and improve INL/DNL performance.
  AdcRegs.ADCCTL1.bit.INTPULSEPOS = 1;       // ADCINT1 trips 1 cycle prior to ADC result latching into its result register
```

```c
  AdcRegs.INTSEL1N2.bit.INT1E     = 1;      // Enabled ADCINT1
  AdcRegs.INTSEL1N2.bit.INT1CONT  = 0;      // Disable ADCINT1 Continuous mode
  AdcRegs.INTSEL1N2.bit.INT1SEL   = 3;        // setup EOC5 to trigger ADCINT1 to fire. SEE below two lines. This is why EOC5 sets ADCINT1- it corresponds to the last conversion

  AdcRegs.ADCSOC0CTL.bit.CHSEL  = 8;//0;//8;        // set SOC0 channel select to ADCINB0; SOCx can be set to any ADCINyz
  AdcRegs.ADCSOC1CTL.bit.CHSEL  = 9;//3;//9;        // set SOC1 channel select to ADCINB1
  AdcRegs.ADCSOC2CTL.bit.CHSEL  = 10;//2;//10;        // set SOC2 channel select to ADCINB2
  AdcRegs.ADCSOC3CTL.bit.CHSEL  = 11;//;//11;       // set SOC3 channel select to ADCINB3
  AdcRegs.ADCSOC4CTL.bit.CHSEL  = 12;//;//11;       // set SOC4 channel select to ADCINB4

  AdcRegs.ADCSOC0CTL.bit.TRIGSEL  = 5;        // set SOC0 start trigger on EPWM1A
  AdcRegs.ADCSOC1CTL.bit.TRIGSEL  = 5;        // set SOC1 start trigger on EPWM1A, due to round-robin SOC0 converts first then SOC1
  AdcRegs.ADCSOC2CTL.bit.TRIGSEL  = 5;        // set SOC2 start trigger on EPWM1A, due to round-robin SOC1 converts first then SOC2
  AdcRegs.ADCSOC3CTL.bit.TRIGSEL  = 5;        // set SOC3 start trigger on EPWM1A, due to round-robin SOC2 converts first then SOC3
  AdcRegs.ADCSOC4CTL.bit.TRIGSEL  = 5;        // set SOC4 start trigger on EPWM1A, due to round-robin SOC4 converts first then SOC4

  AdcRegs.ADCSOC0CTL.bit.ACQPS  = 6;      // set SOC0 S/H Window to 7 ADC Clock Cycles, (6 ACQPS plus 1)
  AdcRegs.ADCSOC1CTL.bit.ACQPS  = 6;      // set SOC1 S/H Window to 7 ADC Clock Cycles, (6 ACQPS plus 1)
  AdcRegs.ADCSOC2CTL.bit.ACQPS  = 6;      // set SOC2 S/H Window to 7 ADC Clock Cycles, (6 ACQPS plus 1)
  AdcRegs.ADCSOC3CTL.bit.ACQPS  = 6;      // set SOC3 S/H Window to 7 ADC Clock Cycles, (6 ACQPS plus 1)
  AdcRegs.ADCSOC4CTL.bit.ACQPS  = 6;      // set SOC4 S/H Window to 7 ADC Clock Cycles, (6 ACQPS plus 1)
  EDIS;

// Below lines assume ePWM1 clock is already enabled in InitSysCtrl();
  EPwm1Regs.ETSEL.bit.SOCAEN  = 1;        // Enable SOC on A group
  EPwm1Regs.ETSEL.bit.SOCASEL = 1;        // Select SOC from TBCTR = 0
  EPwm1Regs.ETPS.bit.SOCAPRD  = 2;        // Generate pulse on 2nd event. These bits determine how many selected ETSEL[SOCASEL] events need to occur before an EPWMxSOCA pulse is generated.
}

void EPwm_Config()
{
  EPwm1_Config();
  EPwm2_Config();
  EPwm3_Config();
  EPwm4_Config();
  EPwm5_Config();
  EPwm6_Config();
  EPwm7_Config();
  EPwm8_Config();
}

void EPwm1_Config()
{
  EPwm1Regs.TBPRD = period;                 // Set timer period, PWM frequency = 1 / period
  EPwm1Regs.TBPHS.half.TBPHS = 0;         // Time-Base Phase Register, master's phase = 0

  EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;  // Count-up mode: used for asymmetric PWM
  EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;       // Disable phase loading
  EPwm1Regs.TBCTL.bit.PRDLD = TB_SHADOW;        // Set Shadowed load
```

```
  EPwm1Regs.TBCTL.bit.SYNCOSEL = TB_CTR_ZERO;    // Sync down-stream module
  EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Prescaler = 0 for max freq
  EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;      // Prescaler = 1 for max freq

  EPwm1Regs.AQCTLA.bit.ZRO = AQ_SET;                // Set PWM2A on Zero
  EPwm1Regs.AQCTLA.bit.PRD = AQ_CLEAR;              // Clear PWM2A on event A, up count
  EPwm1Regs.AQCTLB.bit.ZRO = AQ_CLEAR;              // Set PWM2B on Zero
  EPwm1Regs.AQCTLB.bit.PRD = AQ_SET;                // Clear PWM2B on event B, up count
}

void EPwm2_Config()
{
  EPwm2Regs.TBPRD = period;                  // Set timer period, PWM frequency = 1 / period
  EPwm2Regs.TBPHS.half.TBPHS = phi_1;        // Time-Base Phase Register, slave's phase = phi

  EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;  // Count-up mode: used for asymmetric PWM
  EPwm2Regs.TBCTL.bit.PHSEN = TB_ENABLE;         // Enable phase loading, This is slave
  EPwm2Regs.TBCTL.bit.PRDLD = TB_SHADOW;         // Set Shadowed load
  EPwm2Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;     // Sync down-stream module, slave, sync from epwm1
  EPwm2Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Prescaler = 0 for max freq
  EPwm2Regs.TBCTL.bit.CLKDIV = TB_DIV1;      // Prescaler = 0 for max freq

  EPwm2Regs.AQCTLA.bit.ZRO = AQ_SET;                // Set PWM2A on Zero
  EPwm2Regs.AQCTLA.bit.PRD = AQ_CLEAR;              // Clear PWM2A on event A, up count
  EPwm2Regs.AQCTLB.bit.ZRO = AQ_CLEAR;              // Set PWM2B on Zero
  EPwm2Regs.AQCTLB.bit.PRD = AQ_SET;                // Clear PWM2B on event B, up count
}

void EPwm3_Config()
{
  EPwm3Regs.TBPRD = period;                  // Set timer period, PWM frequency = 1 / period
  EPwm3Regs.TBPHS.half.TBPHS = 0;          // Time-Base Phase Register, slave's phase = phi

  EPwm3Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;  // Count-up mode: used for asymmetric PWM
  EPwm3Regs.TBCTL.bit.PHSEN = TB_DISABLE;        // Enable phase loading, This is slave
  EPwm3Regs.TBCTL.bit.PRDLD = TB_SHADOW;         // Set Shadowed load
  EPwm3Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;     // Sync down-stream module, slave, sync from epwm1
  EPwm3Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Prescaler = 0 for max freq
  EPwm3Regs.TBCTL.bit.CLKDIV = TB_DIV1;      // Prescaler = 0 for max freq

  EPwm3Regs.AQCTLA.bit.ZRO = AQ_SET;                // Set PWM2A on Zero
  EPwm3Regs.AQCTLA.bit.PRD = AQ_CLEAR;              // Clear PWM2A on event A, up count
  EPwm3Regs.AQCTLB.bit.ZRO = AQ_CLEAR;              // Set PWM2B on Zero
  EPwm3Regs.AQCTLB.bit.PRD = AQ_SET;                // Clear PWM2B on event B, up count
}

void EPwm4_Config()
{
  EPwm4Regs.TBPRD = period;                  // Set timer period, PWM frequency = 1 / period
  EPwm4Regs.TBPHS.half.TBPHS = phi_2;          // Time-Base Phase Register, slave's phase = phi

  EPwm4Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;  // Count-up mode: used for asymmetric PWM
  EPwm4Regs.TBCTL.bit.PHSEN = TB_ENABLE;         // Enable phase loading, This is slave
  EPwm4Regs.TBCTL.bit.PRDLD = TB_SHADOW;         // Set Shadowed load
  EPwm4Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;     // Sync down-stream module, slave, sync from epwm1
  EPwm4Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Prescaler = 0 for max freq
  EPwm4Regs.TBCTL.bit.CLKDIV = TB_DIV1;      // Prescaler = 0 for max freq

  EPwm4Regs.AQCTLA.bit.ZRO = AQ_SET;                // Set PWM2A on Zero
  EPwm4Regs.AQCTLA.bit.PRD = AQ_CLEAR;              // Clear PWM2A on event A, up count
  EPwm4Regs.AQCTLB.bit.ZRO = AQ_CLEAR;              // Set PWM2B on Zero
```

```
  EPwm4Regs.AQCTLB.bit.PRD = AQ_SET;               // Clear PWM2B on event B, up count
}

void EPwm5_Config()
{
  EPwm5Regs.TBPRD = period;                  // Set timer period, PWM frequency = 1 / period
  EPwm5Regs.TBPHS.half.TBPHS = 0;        // Time-Base Phase Register, slave's phase = phi

  EPwm5Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;  // Count-up mode: used for asymmetric PWM
  EPwm5Regs.TBCTL.bit.PHSEN = TB_DISABLE;        // Enable phase loading, This is slave
  EPwm5Regs.TBCTL.bit.PRDLD = TB_SHADOW;         // Set Shadowed load
  EPwm5Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;     // Sync down-stream module, slave, sync from epwm1
  EPwm5Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Prescaler = 0 for max freq
  EPwm5Regs.TBCTL.bit.CLKDIV = TB_DIV1;      // Prescaler = 0 for max freq

  EPwm5Regs.AQCTLA.bit.ZRO = AQ_SET;               // Set PWM2A on Zero
  EPwm5Regs.AQCTLA.bit.PRD = AQ_CLEAR;             // Clear PWM2A on event A, up count
  EPwm5Regs.AQCTLB.bit.ZRO = AQ_CLEAR;             // Set PWM2B on Zero
  EPwm5Regs.AQCTLB.bit.PRD = AQ_SET;               // Clear PWM2B on event B, up count
}

void EPwm6_Config()
{
  EPwm6Regs.TBPRD = period;                  // Set timer period, PWM frequency = 1 / period
  EPwm6Regs.TBPHS.half.TBPHS = phi_3;          // Time-Base Phase Register, slave's phase = phi

  EPwm6Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;  // Count-up mode: used for asymmetric PWM
  EPwm6Regs.TBCTL.bit.PHSEN = TB_ENABLE;         // Enable phase loading, This is slave
  EPwm6Regs.TBCTL.bit.PRDLD = TB_SHADOW;         // Set Shadowed load
  EPwm6Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;     // Sync down-stream module, slave, sync from epwm1
  EPwm6Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Prescaler = 0 for max freq
  EPwm6Regs.TBCTL.bit.CLKDIV = TB_DIV1;

  EPwm6Regs.AQCTLA.bit.ZRO = AQ_SET;               // Set PWM2A on Zero
  EPwm6Regs.AQCTLA.bit.PRD = AQ_CLEAR;             // Clear PWM2A on event A, up count
  EPwm6Regs.AQCTLB.bit.ZRO = AQ_CLEAR;             // Set PWM2B on Zero
  EPwm6Regs.AQCTLB.bit.PRD = AQ_SET;               // Clear PWM2B on event B, up count
}

void EPwm7_Config()
{
  EPwm7Regs.TBPRD = period;                  // Set timer period, PWM frequency = 1 / period
  EPwm7Regs.TBPHS.half.TBPHS = 0;        // Time-Base Phase Register, slave's phase = phi

  EPwm7Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;  // Count-up mode: used for asymmetric PWM
  EPwm7Regs.TBCTL.bit.PHSEN = TB_DISABLE;        // Enable phase loading, This is slave
  EPwm7Regs.TBCTL.bit.PRDLD = TB_SHADOW;         // Set Shadowed load
  EPwm7Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;     // Sync down-stream module, slave, sync from epwm1
  EPwm7Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Prescaler = 0 for max freq
  EPwm7Regs.TBCTL.bit.CLKDIV = TB_DIV1;      // Prescaler = 0 for max freq

  EPwm7Regs.AQCTLA.bit.ZRO = AQ_SET;               // Set PWM2A on Zero
  EPwm7Regs.AQCTLA.bit.PRD = AQ_CLEAR;             // Clear PWM2A on event A, up count
  EPwm7Regs.AQCTLB.bit.ZRO = AQ_CLEAR;             // Set PWM2B on Zero
  EPwm7Regs.AQCTLB.bit.PRD = AQ_SET;               // Clear PWM2B on event B, up count
}

void EPwm8_Config()
{
  EPwm8Regs.TBPRD = period;                  // Set timer period, PWM frequency = 1 / period
  EPwm8Regs.TBPHS.half.TBPHS = phi_4;        // Time-Base Phase Register, slave's phase = phi
```

```c

    EPwm8Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;  // Count-up mode: used for asymmetric PWM
    EPwm8Regs.TBCTL.bit.PHSEN = TB_ENABLE;          // Enable phase loading, This is slave
    EPwm8Regs.TBCTL.bit.PRDLD = TB_SHADOW;          // Set Shadowed load
    EPwm8Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;      // Sync down-stream module, slave, sync from epwm1
    EPwm8Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;        // Prescaler = 0 for max freq
    EPwm8Regs.TBCTL.bit.CLKDIV = TB_DIV1;       // Prescaler = 0 for max freq

    EPwm8Regs.AQCTLA.bit.ZRO = AQ_SET;                  // Set PWM2A on Zero
    EPwm8Regs.AQCTLA.bit.PRD = AQ_CLEAR;                // Clear PWM2A on event A, up count
    EPwm8Regs.AQCTLB.bit.ZRO = AQ_CLEAR;                // Set PWM2B on Zero
    EPwm8Regs.AQCTLB.bit.PRD = AQ_SET;                  // Clear PWM2B on event B, up count
}
//===========================================================================
// End of file.
//===========================================================================
```

APPENDIX C

FREQUENCY RESPONSE OF A SIX-LEVEL FCML BUCK CONVERTER

In Section 7.2.5, the dynamic behavior of the FCML buck converter was approximated by the dynamic behavior of a conventional buck converter in order to tune the compensator parameters. This approximation is experimentally validated by comparing the frequency response of FCML and conventional (two-level) buck converters. Here, the experimental comparison study is briefly explained and the results are provided.

Control-to-inductor current and control-to-output voltage frequency response are needed to develop multiloop feedback in PFC applications. A low-cost experimental setup to measure open loop frequency response of the six-level FCML and conventional buck converters is created. Figure C.1 depicts a high-level diagram of the experimental setup which consists of six-level and conventional buck converter prototypes used in Chapter 8, a Texas Instruments F28377D microcontroller, and a Tektronix MSO4034 digital oscilloscope. The six-level FCML converter prototype shown in Figure 8.3 is configured as a conventional buck converter, and both the six-level FCML and conventional buck converter are updated with the parameters listed in Table C.1 for the frequency response comparison study. The microcontroller both runs the converters in dc-dc operation and generates disturbance on control input (i.e., duty ratio). The oscilloscope measures the disturbance, output voltage and current.

In Figure C.1, $\mathrm{D_{fixed}}$ is a fixed duty ratio, equal to 0.12 given the operating conditions in Table C.1. At every transistor switching period (i.e., $\frac{1}{f_{sw}}$), $\mathrm{D_{fixed}}$ is disturbed by $\tilde{d}$ which is generated by the Trigonometric Math Unit (TMU) of the microcontroller and a constant scaler, which are shown in Figure C.1 as $sin(.)$ and K, respectively. The TMU outputs a sine wave consisting of floating point numbers between -1 and 1 at disturbance frequency, $f_{\tilde{d}}$. The TMU output is sent to a digital to analog converter (DAC) module to generate a representative voltage (D_{m}) of the disturbance. Note that since the DAC module cannot generate negative voltage, TMU output is properly scaled and offset considering DAC module range and resolution in order to be captured by

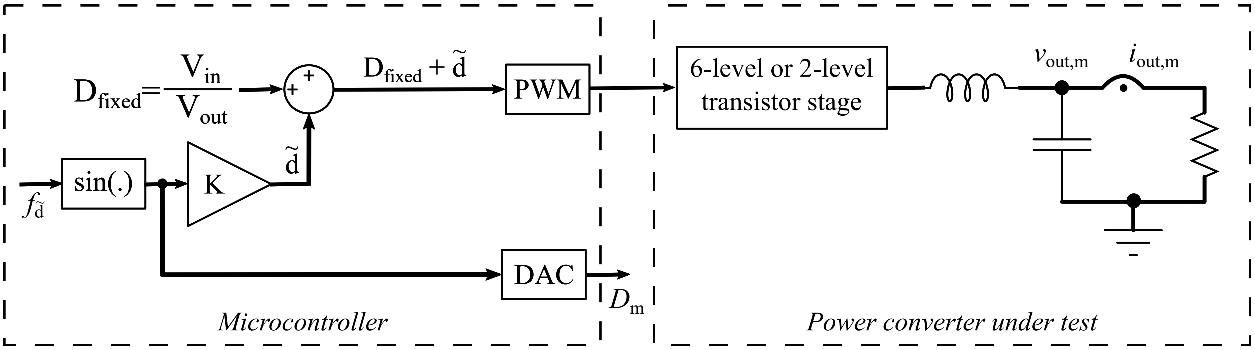


Figure C.1: High-level diagram of the frequency response comparison experimental setup.

Table C.1: Updated components and specifications for the frequency response comparison of two-level and six-level configurations of the hardware prototype

Specification	New Value
Input Voltage	40 V
Output Voltage	4.8 V
Output Power	20 W
Component	**New Value**
Filter Inductor	6.8 μH
Output Capacitor	160 μF
Flying Capacitor (only in 6-level)	13.2 μF per level
f_{sw}	80 kHz

the oscilloscope. Disturbed control signal ($D_{fixed} + \tilde{d}$) is sent to PWM module the microcontroller to drive six-level or two-level transistor stage, both consisting of the same transistors, inductor and output capacitor as shown in Figure C.1.

In order to analyze the frequency response of the converter under test, $f_{\tilde{d}}$ is manually adjusted between 20 Hz and 10 kHz at select frequencies while the converter is operating. The oscilloscope continuously captures D_{m}, the ac coupled output voltage ($v_{out,m}$) and the ac coupled output current ($i_{out,m}$) for each different $f_{\tilde{d}}$, and calculates the peak-to-peak values of D_{m}, $v_{out,m}$, and $i_{out,m}$, and the phase difference between D_{m} and $v_{out,m}$, and between D_{m} and $i_{out,m}$. The calculated quantities are manually recorded in a spreadsheet. In this experiment, K is empirically chosen as 0.01. Note that the peak-to-peak voltage of D_{m} needs to be rescaled by K to properly represent $\tilde{d}$.

Manually recorded quantities of the control-to-output voltage frequency response of the conventional and six-level FCML buck converters are given in Tables C.2 and C.3, respectively. For the control-to-output current response measurement, the conventional buck converter with specifications given in Table C.1 resulted in excessive current ripple, and does not produce meaningful results. Thus, control-to-output current response is only recorded for the six-level FCML converter as given in Table C.4. The results given in Tables C.2, C.3 and C.4 are also plotted in Figures C.2 and C.3 along with the theoretical model of the conventional buck converter as explained in [186] for reference.

Table C.2: Control-to-output voltage frequency response of the six-level FCML buck converter

$f_{\tilde{d}}$ $[Hz]$	Phase [°]	$v_{out,m}$ $[V_{p-p}]$	D_{m} $[V_{p-p}]$	$D_{\mathrm{m}}\times\mathrm{K}$ $[V_{p-p}]$	Gain [dB] $(=20\times log(\frac{v_{out,m}}{D_{\mathrm{m}}\times\mathrm{K}}))$
20	-21.12	0.92	2.26	0.0226	32.19
50	-10.83	0.94	2.24	0.0224	32.46
100	-7.23	0.92	2.28	0.0228	32.12
200	-2.4	0.96	2.24	0.0224	32.64
300	-1.804	0.94	2.24	0.0224	32.46
397.7	0	0.94	2.24	0.0224	32.46
500	8.036	0.96	2.24	0.0224	32.64
603.4	-2.793	0.98	2.24	0.0224	32.82
701	1.816	0.94	2.24	0.0224	32.46
795.8	7.162	0.96	2.28	0.0228	32.49
909.9	2.948	0.94	2.24	0.0224	32.46
1000	5.7	0.98	2.28	0.0228	32.67
2013	18.12	1.08	2.28	0.0228	33.51
3051	30.51	1.36	2.24	0.0224	35.67
3982	58.94	1.74	2.2	0.022	37.96
4167	80.45	1.82	2.24	0.0224	38.20
4563	73.54	1.84	2.2	0.022	38.45
4778	99.3	1.82	2.2	0.022	38.35
4977	115.6	1.66	2.24	0.0224	37.40
6284	165.1	1.04	2.24	0.0224	33.34
7650	165.5	0.66	2.2	0.022	29.54
7751	171	0.54	2.12	0.0212	28.12
9007	153	0.36	2.28	0.0228	23.97

As shown in Figures C.2 and C.2, the measured control-to-output voltage and control-to-output current frequency response of the six-level FCML converter match the theoretical model of a conventional buck converter.

Table C.3: Control-to-output voltage frequency response of the conventional buck converter

$f_{\tilde{d}}$ $[Hz]$	Phase $[°]$	$v_{out,m}$ $[V_{p-p}]$	D_{m} $[V_{p-p}]$	$D_{\mathrm{m}}\times\mathrm{K}$ $[V_{p-p}]$	Gain [dB] $(=20\times log(\frac{v_{out,m}}{D_{\mathrm{m}}\times\mathrm{K}}))$
20.04	19.12	1.02	2.64	0.0264	31.74
49.5	2.926	1.04	2.7	0.027	31.71
100.7	-1.83	1.06	2.72	0.0272	31.81
199.2	1.233	1.12	2.72	0.0272	32.29
303.3	2.69	1.1	2.76	0.0276	32.01
395.3	-7.332	1.1	2.8	0.028	31.88
503.4	-3.603	1.06	2.88	0.0288	31.32
603	-0.491	1.08	2.76	0.0276	31.85
705	-5.806	1.08	2.68	0.0268	32.11
793.7	-4.353	1.12	2.88	0.0288	31.80
909.1	11.57	1.12	2.76	0.0276	32.17
1000	0.1	1.121	2.6	0.026	32.69
1992	-10.29	1.24	2.76	0.0276	33.05
3027	-24.29	1.56	2.67	0.0267	35.33
4000	-60.48	2.42	2.44	0.0244	39.93
4223	-69	2.56	2.68	0.0268	39.60
4449	-101	2.76	2.48	0.0248	40.93
4973	-109.5	2.3	2.52	0.0252	39.21
4996	-130	1.52	2.64	0.0264	35.20
6635	-154.7	1	2.76	0.0276	31.18
7634	-160	0.7	2.8	0.028	27.96
8922	-142.9	0.58	2.6	0.026	26.97

Table C.4: Control-to-output current frequency response of the six-level FCML buck converter

$f_{\tilde{d}}$ $[Hz]$	Phase $[°]$	$i_{out,m}$ $[A_{p-p}]$	D_{m} $[V_{p-p}]$	$D_{\mathrm{m}}\times\mathrm{K}$ $[V_{p-p}]$	Gain [dB] $(=20\times log(\frac{i_{out,m}}{D_{\mathrm{m}}\times\mathrm{K}}))$
1000	128.2	0.8	2.24	0.0224	31.06
2000	123	2.48	2.32	0.0232	40.58
2280	135	3.44	2.28	0.0228	43.57
4001	45.78	11	2.2	0.022	53.98
4218	19.67	11.8	2.2	0.022	54.59
4444	20.27	13.4	2.2	0.022	55.69
4751	11.4	13.8	2.24	0.0224	55.79
5333	-84.72	12.6	2.2	0.022	55.16
5720	-113.6	11.5	2.2	0.022	54.37
6667	-121	8.08	2.24	0.0224	51.14
8005	-158.2	5.12	2.16	0.0216	47.50
10000	-169.9	3.08	2.26	0.0226	42.69

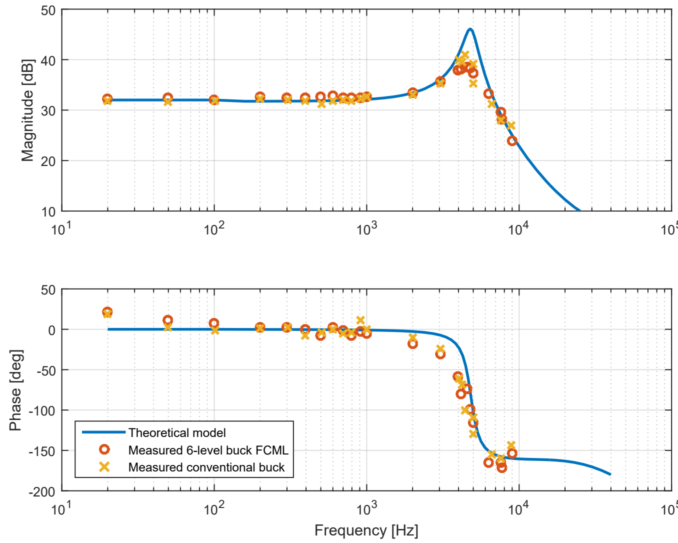


Figure C.2: Control-to-output voltage frequency response of the six-level FCML and conventional buck converter.

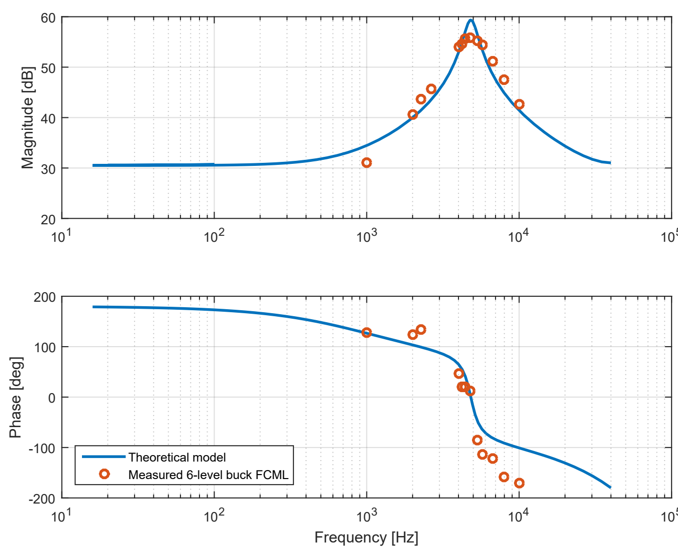


Figure C.3: Control-to-output current frequency response of the six-level FCML buck converter.

APPENDIX D

DESIGN FILES OF PROTOTYPE FCML BUCK CONVERTER

This appendix contains PCB layouts of the prototype FCML buck converter.

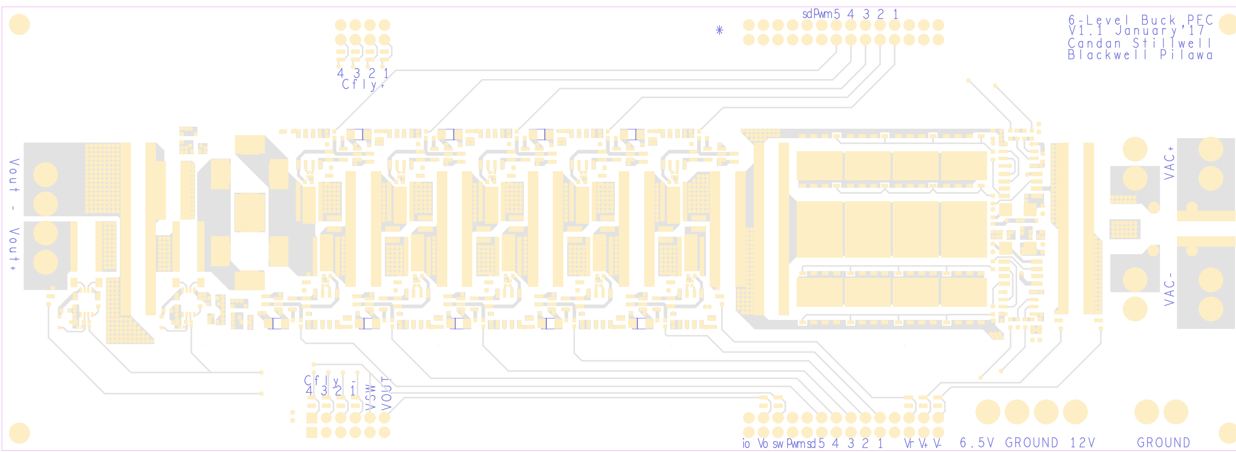


Figure D.1: PCB layout of prototype FCML hardware: Top layer, silkscreen and solder mask. Not to scale due to page width.

Figure D.2: PCB layout of prototype FCML hardware: Bottom layer, silkscreen and solder mask. Not to scale due to page width.

Figure D.3: PCB layout of prototype FCML hardware: Second layer. Not to scale due to page width.

Figure D.4: PCB layout of prototype FCML hardware: Third layer. Not to scale due to page width.

APPENDIX E

MICROCONTROLLER CODE USED IN FCML BUCK PFC CONVERTER EXPERIMENTAL STUDY

This appendix contains microcontroller code used in FCML buck PFC converter experimental study.

Listing E.1: main.c

```c
#include "F28x_Project.h"     // Device Header file and Examples Include File

// Include header files - by EC
#include "global_variables.h"
#include "global_define.h"
#include "initialize.h"
#include "operation.h"

void main(void)
{
// Step 1. Initialize System Control:
    InitSysCtrl();

    // Disable all peripheral clocks to save power. Required clocks must be initialized by commenting out the corresponding lines in DisableAllPeripheralClks() in initialize.c
    // NOTE: Peripheral clocks were initialized in InitSysCtrl().
    DisableAllPeripheralClks();

// Step 2. Initialize GPIO
    InitGpio();
    InitRectifierGPIOs(); //Initialize the active rectifier GPIOs
    InitDebugGPIOs();     //Initialize the Debug GPIO

// Step 3. Initialize Interrupts
    InitInterrupts();   //Initialize the interrupts

// Step 4. Initialize ADC, DAC and ePWM modules
    InitADCs();         //Initialize the ADC modules
    InitDACs();         //Initialize the DAC modules
    InitEPwmModules();  //Initialize the ePWM modules

    // measure MCU amplifier bias value here at zero current.
    bias_measurement();  // this function will take a few seconds

    #ifdef ADC_CALIBRATION
    ADC_calibration();
    #endif

    // This has to be done, otherwise the corresponding XX_sum variable might not have the correct number
    // Clear the moving average array for Vout signal
```

```c
    memset(Vout_array, 0, NUM_POINTS_HC);

// Step 5. Enable Interrupts
    EnableInterrupts();

// Step 6. infinite loop
    while(1){};
}

interrupt void adc_trigger(void) // This function is called at every SAMPLING_PERIOD seconds.
{
    #ifdef TIMING_CHECK
    GpioDataRegs.GPASET.bit.GPIO16 = 1;
    #endif

    period_counter++; // Increase period counter.

    Vac_neg = (signed)AdcaResultRegs.ADCRESULT0;
    Vac_pos = (signed)AdcbResultRegs.ADCRESULT0;
    Vout = (signed)AdcaResultRegs.ADCRESULT1;
    Iind = (signed)AdccResultRegs.ADCRESULT0;

    Iind = Iind-Iind_bias;

    Vac_neg_adcin = Vac_neg*REG2ADCIN;
    Vac_pos_adcin = Vac_pos*REG2ADCIN;
    Vout_adcin = Vout*REG2ADCIN;
    Iind_adcin = Iind*REG2ADCIN;

    Vac_neg_real = Vac_neg_adcin * ADCIN2REAL_VAC_NEG;
    Vac_pos_real = Vac_pos_adcin * ADCIN2REAL_VAC_POS;
    Vout_real = Vout_adcin * ADCIN2REAL_VOUT;
    Iind_real = Iind_adcin * ADCIN2REAL_IIND; //Iind_adcin_zero is for additional offset if needed, initially zero

    Vac_real = fabs(-Vac_pos_real+Vac_neg_real); //Since ADCIN2REAL_VAC_NEG and ADCIN2REAL_VAC_POS are not the same, calculate the real value before the moving average.

    //ADC Averaging
    //Moving average for Vout
    Vout_array_sum = Vout_array_sum + Vout - Vout_array[moving_ave_pointer]; // Sum = Sum + newest value - oldest value
    Vout_array[moving_ave_pointer] = Vout;                       // Replace the sample value
    Vout_moving_ave = Vout_array_sum*MOV_AVE_DIVIDER;                 // Update the moving average, This is the integer representation of the real value
    Vout_real_moving_ave = Vout_moving_ave*REG2ADCIN*ADCIN2REAL_VOUT;

    moving_ave_pointer++;                                       // Move the pointer forward
    if (moving_ave_pointer==NUM_POINTS_HC)                      // Reset HC_moving_ave_pointer at every NUM_POINTS_HC iterations
        moving_ave_pointer = 0;

    //Window average for Vac_pos, Vac_neg and Vout around Vac_peak
    if ((VAC_PEAK_START < period_counter) && (period_counter<VAC_PEAK_END)){ // Averaging window around peak
        Vac_pos_sum = Vac_pos_sum + Vac_pos;
        Vac_neg_sum = Vac_neg_sum + Vac_neg;
        // Division, ADC to real value conversion and sum reset are done after pos_h_cyc or neg_h_cyc is decided.
    }

```

```
// PLL
notch_out2 = notch_out1;
notch_out1 = notch_out;
notch_in2 = notch_in1;
notch_in1 = notch_in;
notch_in = __cospuf32(theta)*(Vac_pos-Vac_neg);
notch_out = notch_b2*notch_in+notch_b1*notch_in1+notch_b0*notch_in2-notch_a1*notch_out1-notch_a0
 *notch_out2;

notch_out_sum = notch_out_sum+Ki_pll*notch_out;
pll_PI_out = Kp_pll*notch_out+notch_out_sum;
theta_pre = theta;
theta = theta + (SAMPLING_PERIOD)*(60+pll_PI_out);

// Determine active rectifier gating signal, aka zero crossings
if (theta_pre<0.5 && theta>=0.5)
    neg_h_cyc = 1;
else if (theta_pre<1 && theta>=1)
    pos_h_cyc = 1;
else
{
    pos_h_cyc = 0;
    neg_h_cyc = 0;
}

if (theta>=1)
    theta = theta - 1;
else if (theta<0)
    theta = theta + 1;

if(pos_h_cyc == 1){// This block is executed once when pos half cycle starts
    Vac_neg_real_ave = (Vac_neg_sum>>VAC_PEAK_DIVIDER)*REG2ADCIN*ADCIN2REAL_VAC_NEG;
    Vac_neg_sum = 0;

    GpioDataRegs.GPASET.bit.GPIO12 = 1; //REC_POS_PWM
    GpioDataRegs.GPACLEAR.bit.GPIO15 = 1; //REC_NEG_PWM
}
if(neg_h_cyc == 1){// This block is executed once when neg half cycle starts
    Vac_pos_real_ave = (Vac_pos_sum>>VAC_PEAK_DIVIDER)*REG2ADCIN*ADCIN2REAL_VAC_POS;
    Vac_pos_sum = 0;

    GpioDataRegs.GPACLEAR.bit.GPIO12 = 1;   //REC_POS_PWM
    GpioDataRegs.GPASET.bit.GPIO15 = 1;     //REC_NEG_PWM
}

if((pos_h_cyc == 1)||(neg_h_cyc == 1)) //This block is executed once in every half cycle when
 each half cycle starts.
{
    Vac_real_peak = 0.5*(Vac_pos_real_ave+Vac_neg_real_ave); // more ripple, faster response

    //Deciding the reference voltage for the output
    if (Vout_ramp_mode == 1){ //Vout is ramped to Vout_ideal after Vac_peak exceeds
 Vout_ramp_start
        if (startup_completed == 0){
            if (Vac_real_peak<Vout_ramp_start)
                Vref_startup = VREF_SCALER*Vac_real_peak;
            else{
                if (Vout_real_moving_ave_fixed<Vout_ideal)
                    Vref_startup = Vref_startup + 0.01;
                else{
                    Vref_startup = Vout_ideal;
```

```
                        startup_completed = 1;
                    }
                }
                    Vref = Vref_startup;
            }
            if((startup_completed == 1)&&(Vac_real_peak<Vout_ramp_start)) // Vout is ramped down from Vout_ideal if Vac_peak becomes less than Vout_ramp_start
                Vref = Vref - 0.005;
        }
        else // if Vout is not ramped up to Vout_ideal, Vref follows Vac_peak by preserving conversion ration given by VREF_SCALER
            Vref = VREF_SCALER*Vac_real_peak;

        if (Vref_manual_mode == 1) // Overwrites previous output voltage reference decision. Manually adjust the reference by updating Vref_fixed.
            Vref = Vref_fixed;

        // Voltage Loop
        Vout_err = Vref - Vout_real_moving_ave;

        Vout_err_sum = Vout_err_sum + Ki_v*Vout_err;

        if(Vout_err_sum > Vout_err_sum_sat)
            Vout_err_sum = Vout_err_sum_sat;
        if(Vout_err_sum < -Vout_err_sum_sat)
            Vout_err_sum = -Vout_err_sum_sat;

        iL_peak = Kp_v*Vout_err + Vout_err_sum;

        if(iL_peak > iL_peak_sat)
            iL_peak = iL_peak_sat;
        if(iL_peak < iL_peak_sat_)
            iL_peak = iL_peak_sat_;

        // Change phase shift direction if needed
        Update_PS_dir(ps_dir);

        // Fix the Vout_moving_ave to use a constant value in FF mode
        Vout_real_moving_ave_fixed = Vout_real_moving_ave;

        // Reset current PI loop variables
        iL_err = 0;
        iL_ref = 0;
        Iind_real = 0;
        if(iL_err_sum_reset==0)
            iL_err_sum = 0;

        // Reset counters and flags
        period_counter = 0;
        pfc_counter = 0;
    }

    // Once PLL is locked, generate an internal Vac signal.
    Vac_internal = Vac_real_peak * fabs(__sinpuf32(theta));

#ifndef MANUAL_CUTOFF
    if (Vout_real < Vac_internal){// TURN ON THE CONVERTER
        enable_pfc = 1; // indicates the converter is on
        pfc_counter++;

        GpioDataRegs.GPASET.bit.GPIO16 = 1;
```

```
    }
    else if (Vout_real > Vac_internal){ // TURN OFF THE CONVERTER
        enable_pfc = 0; // indicates the converter is off

        GpioDataRegs.GPACLEAR.bit.GPIO16 = 1;
    }
#endif
#ifdef MANUAL_CUTOFF
    if ((manual_cutoff_count < period_counter) && (period_counter<NUM_POINTS_HC-manual_cutoff_count)
     ){ // CONVERTER ON
        enable_pfc = 1; // indicates the converter is on
        pfc_counter++;

        //GpioDataRegs.GPASET.bit.GPIO16 = 1;
    }
    else{ // CONVERTER OFF
        enable_pfc = 0; // indicates the converter is off

        //GpioDataRegs.GPACLEAR.bit.GPIO16 = 1;
    }
#endif

    if (period_counter > NUM_POINTS_HC+1) { //period_counter = SAMPLING_FREQ / TWICE_LINE_FREQ at
     half cycle, not sure if this if-block is ever called. Keeping it here just in case.
        GpioDataRegs.GPACLEAR.bit.GPIO12 = 1; //REC_POS_PWM
        GpioDataRegs.GPACLEAR.bit.GPIO13 = 1; //REC_NEG_SD
        GpioDataRegs.GPACLEAR.bit.GPIO14 = 1; //REC_POS_SD
        GpioDataRegs.GPACLEAR.bit.GPIO15 = 1; //REC_NEG_PWM

        period_counter = 0;
    }

    // Current Loop
    if (enable_pfc == 1){
        // Calculate current reference offset
        if ((iL_ref_offset_mode == 1) && (pfc_counter == 1)){
            if (iL_ref_mode == 1)
                iL_ref_offset = fabs(__sinpuf32(theta))*iL_peak;
            else
                iL_ref_offset = __sinpuf32(theta)*__sinpuf32(theta)*iL_peak;
        }
        else if ((iL_ref_offset_mode == 0) && (pfc_counter == 1))
            iL_ref_offset = iL_ref_offset_debug;

        Iind_real = Iind_real + Iind_adcin_zero;

        if(iL_ref_mode == 1)
            iL_ref = fabs(__sinpuf32(theta))*iL_peak - iL_ref_offset;
        else
            iL_ref = __sinpuf32(theta)*__sinpuf32(theta)*iL_peak - iL_ref_offset;

        iL_err = iL_ref - Iind_real;
        iL_err_sum = iL_err_sum + Ki_iL*iL_err;

        if(iL_err_sum > iL_err_sum_sat)
            iL_err_sum = iL_err_sum_sat;
        if(iL_err_sum < iL_err_sum_sat_)
            iL_err_sum = iL_err_sum_sat_;

        D_iL = Kp_iL*iL_err + iL_err_sum;

```

```
        if(ff_mode == 0)
            D_ff = 0;
        else if(ff_mode == 1)
            D_ff = Vout_real / Vac_internal;
        else if(ff_mode == 2)
            D_ff = Vref / Vac_internal;
        else if(ff_mode == 3)
            D_ff = Vout_real_moving_ave / Vac_internal;
        else if(ff_mode == 4)
            D_ff = Vout_real_moving_ave_fixed / Vac_internal;
        else if(ff_mode == 5){
            delta_iL = iL_ref - iL_ref_1;
            D_ff = (FF_CONSTANT*delta_iL+Vref) / Vac_internal; //FF_CONSTANT is defined in gloabal_define.h
            iL_ref_1 = iL_ref;
        }
        if(D_ff>D_ff_sat)
            D_ff = D_ff_sat;
        else if (D_ff<D_ff_sat_)
            D_ff = D_ff_sat_;

        D = D_iL + D_ff;

        if(D>D_sat)
            D = D_sat;
        else if (D<D_sat_)
            D = D_sat_;

        enable_FCML = 1;
    }
    else if (enable_pfc == 0){
        enable_FCML = 0;
    }

#ifdef FIXED_DUTY
    D = fixed_D;
    enable_FCML = 1;
#endif
    Update_duty(D);

    if(enable_FCML != enable_FCML_pre){ // Turn ON/OFF FCML if needed
        if(enable_FCML == 1)
            Update_deadtime(deadtime_hs, deadtime_ls);
        else if(enable_FCML == 0)
            Update_deadtime((2*PERIOD)+deadtime_hs, (2*PERIOD)+deadtime_ls);
    }

    enable_FCML_pre = enable_FCML;

    switch(dacc_select){
        case 1:
            dacc_out = D;
            break;
        case 2:
            dacc_out = D_iL;
            break;
        case 3:
            dacc_out = D_ff;
            break;
        case 4:
            dacc_out = iL_ref;
```

```
            break;
        case 5:
            dacc_out = Iind_real;
            break;
        case 6:
            dacc_out = iL_err;
            break;
        case 7:
            dacc_out = iL_err_sum;
            break;
        case 8:
            dacc_out = Vac_internal;
            break;
        case 9:
            dacc_out = Vout_real_moving_ave_fixed;
            break;
        case 10:
            dacc_out = Vout_real_moving_ave;
            break;
        case 11:
            dacc_out = Vac_neg;
            break;
        case 12:
            dacc_out = Vac_pos;
            break;
        case 13:
            dacc_out = Vout_real;
            break;
        case 14:
            dacc_out = Vac_real;
            break;
        case 15:
            dacc_out = Vac_real_peak;
            break;
        case 16:
            dacc_out = Vac_real_peak;
    }

    DaccRegs.DACVALS.bit.DACVALS = dacc_out*dacc_multiplier+dacc_offset;

    // Clear the flag and wait for next interrupt
    AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //clear INT1 flag
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;

    #ifdef TIMING_CHECK
    GpioDataRegs.GPACLEAR.bit.GPIO16 = 1;
    #endif
}
```

Listing E.2: initialize.c

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

void DisableAllPeripheralClks(){

    EALLOW;
    CpuSysRegs.PCLKCR0.bit.CLA1 = 0;
    CpuSysRegs.PCLKCR0.bit.DMA = 0;
```

```
    CpuSysRegs.PCLKCR0.bit.CPUTIMER0 = 0;
    CpuSysRegs.PCLKCR0.bit.CPUTIMER1 = 0;
    CpuSysRegs.PCLKCR0.bit.CPUTIMER2 = 0;
    CpuSysRegs.PCLKCR0.bit.HRPWM = 0;
    CpuSysRegs.PCLKCR0.bit.TBCLKSYNC = 0;

    CpuSysRegs.PCLKCR1.bit.EMIF1 = 0;
    CpuSysRegs.PCLKCR1.bit.EMIF2 = 0;

    CpuSysRegs.PCLKCR2.bit.EPWM1 = 0;
    CpuSysRegs.PCLKCR2.bit.EPWM2 = 0;
    CpuSysRegs.PCLKCR2.bit.EPWM3 = 0;
    CpuSysRegs.PCLKCR2.bit.EPWM4 = 0;
    CpuSysRegs.PCLKCR2.bit.EPWM5 = 0;
    CpuSysRegs.PCLKCR2.bit.EPWM6 = 0;
    CpuSysRegs.PCLKCR2.bit.EPWM7 = 0;
    CpuSysRegs.PCLKCR2.bit.EPWM8 = 0;
    CpuSysRegs.PCLKCR2.bit.EPWM9 = 0;
    CpuSysRegs.PCLKCR2.bit.EPWM10 = 0;
    CpuSysRegs.PCLKCR2.bit.EPWM11 = 0;
    CpuSysRegs.PCLKCR2.bit.EPWM12 = 0;

    CpuSysRegs.PCLKCR3.bit.ECAP1 = 0;
    CpuSysRegs.PCLKCR3.bit.ECAP2 = 0;
    CpuSysRegs.PCLKCR3.bit.ECAP3 = 0;
    CpuSysRegs.PCLKCR3.bit.ECAP4 = 0;
    CpuSysRegs.PCLKCR3.bit.ECAP5 = 0;
    CpuSysRegs.PCLKCR3.bit.ECAP6 = 0;

    CpuSysRegs.PCLKCR4.bit.EQEP1 = 0;
    CpuSysRegs.PCLKCR4.bit.EQEP2 = 0;
    CpuSysRegs.PCLKCR4.bit.EQEP3 = 0;

    CpuSysRegs.PCLKCR6.bit.SD1 = 0;
    CpuSysRegs.PCLKCR6.bit.SD2 = 0;

    CpuSysRegs.PCLKCR7.bit.SCI_A = 0;
    CpuSysRegs.PCLKCR7.bit.SCI_B = 0;
    CpuSysRegs.PCLKCR7.bit.SCI_C = 0;
    CpuSysRegs.PCLKCR7.bit.SCI_D = 0;

    CpuSysRegs.PCLKCR8.bit.SPI_A = 0;
    CpuSysRegs.PCLKCR8.bit.SPI_B = 0;
    CpuSysRegs.PCLKCR8.bit.SPI_C = 0;

    CpuSysRegs.PCLKCR9.bit.I2C_A = 0;
    CpuSysRegs.PCLKCR9.bit.I2C_B = 0;

    CpuSysRegs.PCLKCR10.bit.CAN_A = 0;
    CpuSysRegs.PCLKCR10.bit.CAN_B = 0;

    CpuSysRegs.PCLKCR11.bit.McBSP_A = 0;
    CpuSysRegs.PCLKCR11.bit.McBSP_B = 0;
    CpuSysRegs.PCLKCR11.bit.USB_A = 0;

    CpuSysRegs.PCLKCR12.bit.uPP_A = 0;

    CpuSysRegs.PCLKCR13.bit.ADC_A = 0;
    CpuSysRegs.PCLKCR13.bit.ADC_B = 0;
    CpuSysRegs.PCLKCR13.bit.ADC_C = 0;
    CpuSysRegs.PCLKCR13.bit.ADC_D = 0;
```

```

    CpuSysRegs.PCLKCR14.bit.CMPSS1 = 0;
    CpuSysRegs.PCLKCR14.bit.CMPSS2 = 0;
    CpuSysRegs.PCLKCR14.bit.CMPSS3 = 0;
    CpuSysRegs.PCLKCR14.bit.CMPSS4 = 0;
    CpuSysRegs.PCLKCR14.bit.CMPSS5 = 0;
    CpuSysRegs.PCLKCR14.bit.CMPSS6 = 0;
    CpuSysRegs.PCLKCR14.bit.CMPSS7 = 0;
    CpuSysRegs.PCLKCR14.bit.CMPSS8 = 0;

    CpuSysRegs.PCLKCR16.bit.DAC_A = 0;
    CpuSysRegs.PCLKCR16.bit.DAC_B = 0;
    CpuSysRegs.PCLKCR16.bit.DAC_C = 0;

    EDIS;
}

void InitRectifierGPIOs(){

    // Configure GPIO12, GPIO13, GPIO14, GPIO15 as GPIO output pins for active rectifier.
    // GPIO12 is connected to REC_POS_PWM net
    // GPIO13 is connected to REC_NEG_SD net
    // GPIO14 is connected to REC_POS_SD net
    // GPIO15 is connected to REC_NEG_PWM net
    // make sure they are off before changing the pin to output
    GpioDataRegs.GPACLEAR.bit.GPIO12 = 1; //REC_POS_PWM
    GpioDataRegs.GPACLEAR.bit.GPIO13 = 1; //REC_NEG_SD
    GpioDataRegs.GPACLEAR.bit.GPIO14 = 1; //REC_POS_SD
    GpioDataRegs.GPACLEAR.bit.GPIO15 = 1; //REC_NEG_PWM
    GPIO_SetupPinMux(12, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(12, GPIO_OUTPUT, GPIO_PUSHPULL);
    GPIO_SetupPinMux(13, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(13, GPIO_OUTPUT, GPIO_PUSHPULL);
    GPIO_SetupPinMux(14, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(14, GPIO_OUTPUT, GPIO_PUSHPULL);
    GPIO_SetupPinMux(15, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(15, GPIO_OUTPUT, GPIO_PUSHPULL);
}

void InitDebugGPIOs(){

    GpioDataRegs.GPACLEAR.bit.GPIO16 = 1; // GPIO16 is configured as a debug pin
    GPIO_SetupPinMux(16, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(16, GPIO_OUTPUT, GPIO_PUSHPULL);
}

void InitInterrupts(){

    // See Section 2.4.4.1 Enabling Interrupts of Technical Manual spruhm8g.pdf, p.92.

    // Sub-Step 1: Disable interrupts globally.

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

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

```c
    EDIS;

    // Sub-Step 2: Enable the PIE by setting the ENPIE bit of the PIECTRL register.

    // Initialize the PIE vector table with pointers to the shell Interrupt
    // Service Routines (ISR).
    // This will populate the entire table, even if the interrupt
    // is not used in this example.  This is useful for debug purposes.
    // The shell ISR routines are found in F2837xD_DefaultIsr.c.
    // ENPIE bit of the PIECTRL register is set in at the end of the InitPieVectTable() function.
    InitPieVectTable();

    // Sub-Step 3: Write the ISR vector for each interrupt to the appropriate location in the PIE vector table, which can be found in Table 2-2.

    //Map ISR functions
    EALLOW;

    PieVectTable.ADCA1_INT = &adc_trigger; //function for ADC interrupt. ADCA1 interrupt has the highest priority. See Table 2-2.

    // Sub-Step 4: Set the appropriate PIEIERx bit for each interrupt. The PIE group and channel assignments can be found in Table 2-2.
    PieCtrlRegs.PIEIER1.bit.INTx1 = 1;  //Enable PIE interrupt for ADCa_INT1, see Table 2.2

    // Sub-Step 5: Set the CPU IER bit for any PIE group containing enabled interrupts. Enable higher priority real-time debug events
    IER |= M_INT1; //Enable group 1 interrupts
    //ERTM; // Enable Global realtime interrupt DBGM. "EC:Not sure what this does, commenting out for now."

    EDIS;

    // Sub-Step 6: Enable the interrupt in the peripheral.
    // This is done in peripheral initialization.

    // Sub-Step 7: Enable interrupts globally.
    // This step is done in main.c by EnableInterrupts()
}

void InitADCs(){

    // Initialize ADC sampling
    InitADCa(); // initialize ADCa
    InitADCb(); // initialize ADCb
    InitADCc(); // initialize ADCc
    InitADCd(); // initialize ADCd
}

void InitADCa(){

    EALLOW;
    CpuSysRegs.PCLKCR13.bit.ADC_A = 1; // Enable ADC_A Clock. Initially it was disabled in DisableAllPeripheralClks().

    //write configurations
    AdcaRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4
    AdcSetMode(ADC_ADCA, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE);

    //Set pulse positions to late (at the end of conversion)
    AdcaRegs.ADCCTL1.bit.INTPULSEPOS = 1;
```

```

    //power up the ADC
    AdcaRegs.ADCCTL1.bit.ADCPWDNZ = 1;

    //SOC0 measures VAC_POS_SENSE on pin ADCINA2
    AdcaRegs.ADCSOC0CTL.bit.CHSEL = 2;  //SOC0 will convert pin A2
    AdcaRegs.ADCSOC0CTL.bit.ACQPS = 20; //sample window (# of SYSCLK, needs to corresponds to at least 75ns)
    AdcaRegs.ADCSOC0CTL.bit.TRIGSEL = 5; //trigger on ePWM1 SOCA

    AdcaRegs.ADCSOC1CTL.bit.CHSEL = 4;  //SOC1 will convert pin A4
    AdcaRegs.ADCSOC1CTL.bit.ACQPS = 20; //sample window (# of SYSCLK, needs to corresponds to at least 75ns)
    AdcaRegs.ADCSOC1CTL.bit.TRIGSEL = 5; //trigger on ePWM1 SOCA

    AdcaRegs.ADCINTSEL1N2.bit.INT1SEL = 1; //end of SOC1 will set INT1 flag
    AdcaRegs.ADCINTSEL1N2.bit.INT1E = 1;   //enable INT1 interrupts
    AdcaRegs.ADCINTSEL1N2.bit.INT1CONT = 0;   //No further ADCINT1 pulses are generated until ADCINT1 flag is cleared by user
    AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //make sure INT1 flag is cleared



    EDIS;
}

void InitADCb(void){

    EALLOW;
    CpuSysRegs.PCLKCR13.bit.ADC_B = 1; // Enable ADC_B Clock. Initially it was disabled in DisableAllPeripheralClks().

    //write configurations
    AdcbRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4
    AdcSetMode(ADC_ADCB, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE);

    //Set pulse positions to late (at the end of conversion)
    AdcbRegs.ADCCTL1.bit.INTPULSEPOS = 1;

    //power up the ADC
    AdcbRegs.ADCCTL1.bit.ADCPWDNZ = 1;

    //SOC0 measures VAC_POS_SENSE on pin ADCINB2
    AdcbRegs.ADCSOC0CTL.bit.CHSEL = 2;  //SOC0 will convert pin B2
    AdcbRegs.ADCSOC0CTL.bit.ACQPS = 20; //sample window (# of SYSCLK, needs to corresponds to at least 75ns)
    AdcbRegs.ADCSOC0CTL.bit.TRIGSEL = 5; //trigger on ePWM1 SOCA

    EDIS;
}

void InitADCc(void){

    EALLOW;
    CpuSysRegs.PCLKCR13.bit.ADC_C = 1; // Enable ADC_C Clock. Initially it was disabled in DisableAllPeripheralClks().

    //write configurations
    AdccRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4
    AdcSetMode(ADC_ADCC, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE);

```

```c
    //Set pulse positions to late (at the end of conversion)
    AdccRegs.ADCCTL1.bit.INTPULSEPOS = 1;

    //power up the ADC
    AdccRegs.ADCCTL1.bit.ADCPWDNZ = 1;

    //SOC0 measures VAC_POS_SENSE on pin ADCINB2
    AdccRegs.ADCSOC0CTL.bit.CHSEL = 4;  //SOC0 will convert pin C4
    AdccRegs.ADCSOC0CTL.bit.ACQPS = 20; //sample window (# of SYSCLK, needs to corresponds to at
     least 75ns)
    AdccRegs.ADCSOC0CTL.bit.TRIGSEL = 5; //trigger on ePWM1 SOCA

    EDIS;
}


void InitADCd(void){

    EALLOW;
    CpuSysRegs.PCLKCR13.bit.ADC_D = 1; // Enable ADC_D Clock. Initially it was disabled in
     DisableAllPeripheralClks().

    //write configurations
    AdcdRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4
    AdcSetMode(ADC_ADCD, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE);

    //Set pulse positions to late (at the end of conversion)
    AdcdRegs.ADCCTL1.bit.INTPULSEPOS = 1;

    //power up the ADC
    AdcdRegs.ADCCTL1.bit.ADCPWDNZ = 1;

    //SOC0 measures VRECT_SENSE on pin ADCIND2
    AdcdRegs.ADCSOC0CTL.bit.CHSEL = 2;  //SOC0 will convert pin D2
    AdcdRegs.ADCSOC0CTL.bit.ACQPS = 20; //sample window (# of SYSCLK, needs to corresponds to at
     least 75ns)
    AdcdRegs.ADCSOC0CTL.bit.TRIGSEL = 5; //trigger on ePWM1 SOCA

    EDIS;
}

void InitDACs(){

    EALLOW;

    // Enable DACOUTA

    CpuSysRegs.PCLKCR16.bit.DAC_A = 1; // Enable DAC_A Clock. Initially it was disabled in
     DisableAllPeripheralClks().
    //Use VDAC as the reference for DAC
    DacaRegs.DACCTL.bit.DACREFSEL = 1;
    //Enable DAC output
    DacaRegs.DACOUTEN.bit.DACOUTEN = 1;
    //Set DAC to some initial value
    DacaRegs.DACVALS.bit.DACVALS = 2048;

    // Enable DACOUTB
    CpuSysRegs.PCLKCR16.bit.DAC_B = 1; // Enable DAC_B Clock. Initially it was disabled in
     DisableAllPeripheralClks().
    //Use VDAC as the reference for DAC
    DacbRegs.DACCTL.bit.DACREFSEL = 1;
```

```c
    //Enable DAC output
    DacbRegs.DACOUTEN.bit.DACOUTEN = 1;
    //Set DAC to some initial value
    DacbRegs.DACVALS.bit.DACVALS = 2048;

    // Enable DACOUTC
    CpuSysRegs.PCLKCR16.bit.DAC_C = 1; // Enable DAC_C Clock. Initially it was disabled in DisableAllPeripheralClks().
    //Use VDAC as the reference for DAC
    DaccRegs.DACCTL.bit.DACREFSEL = 1;
    //Enable DAC output
    DaccRegs.DACOUTEN.bit.DACOUTEN = 1;
    //Set DAC to some initial value
    DaccRegs.DACVALS.bit.DACVALS = 2048;

    EDIS;
}

void InitEPwmModules(){

    // Set PWM clock the same as SYSCLK
    EALLOW;
    ClkCfgRegs.PERCLKDIVSEL.bit.EPWMCLKDIV = 0x0;
    EDIS;

    // Enable ePWM clocks. Initially they were disabled in DisableAllPeripheralClks()
    EALLOW;
    CpuSysRegs.PCLKCR2.bit.EPWM1=1;
    CpuSysRegs.PCLKCR2.bit.EPWM2=1;
    CpuSysRegs.PCLKCR2.bit.EPWM3=1;
    CpuSysRegs.PCLKCR2.bit.EPWM4=1;
    CpuSysRegs.PCLKCR2.bit.EPWM5=1;
    CpuSysRegs.PCLKCR2.bit.EPWM6=1;
    EDIS;

    // Init GPIO pins
    InitEPwm1Gpio(); //EPwm1 is used as master clock
    InitEPwm2Gpio();    //EPwm2a/b is for SW1H/L
    InitEPwm3Gpio();    //EPwm3a/b is for SW2H/L
    InitEPwm4Gpio();    //EPwm4a/b is for SW3H/L
    InitEPwm5Gpio();    //EPwm5a/b is for SW4H/L
    InitEPwm6Gpio();    //EPwm6a/b is for SW5H/L


    int ps2=(2*PERIOD)*2/(NUM_LEVELS-1);
    int ps3=(2*PERIOD)*1/(NUM_LEVELS-1);
    int ps4=0;                               //Phase shift registers must be distributed like this
    int ps5=(2*PERIOD)*1/(NUM_LEVELS-1); //in order to achieve the PS-PWM order from top to bottom switch pairs
    int ps6=(2*PERIOD)*2/(NUM_LEVELS-1); //that we are used to from publications.

    InitEPwmMaster(PERIOD);
    InitEPwmFCML(&EPwm2Regs, PERIOD, ps2, 1);
    InitEPwmFCML(&EPwm3Regs, PERIOD, ps3, 1);
    InitEPwmFCML(&EPwm4Regs, PERIOD, ps4, 0);
    InitEPwmFCML(&EPwm5Regs, PERIOD, ps5, 0);
    InitEPwmFCML(&EPwm6Regs, PERIOD, ps6, 0); //Note that the direction bit has no effect unless PWM is COUNT_UP_DOWN

    //Synchronize all ePWMs to the TBCLK. Note that this will also start ADC conversion since ADC are triggered by ePWM1
```

```
    EALLOW;
        CpuSysRegs.PCLKCR0.bit.TBCLKSYNC = 1;
    EDIS;

}

void InitEPwmMaster(int32 period){

    EALLOW;

    EPwm1Regs.TBPHS.all = 0x00000000;
    EPwm1Regs.TBPRD = period;                          // Set timer period. Note PWM counting starts from 0
    EPwm1Regs.TBPRDHR = 0;
    EPwm1Regs.TBCTR = 0x0000;                          // Clear counter

    // Set up TBCLK
    EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;    // Count up and down
    EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;            // Disable Phase loading
    EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;          // Clock ratio to SYSCLKOUT
    EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;             // Slow to observe on the scope
    EPwm1Regs.TBCTL.bit.SYNCOSEL = TB_CTR_ZERO;        // output EPWMxSYNCO when CTR=0

    // Setup compare register loading
    EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;     // load from shadow register from at both CTR=ZERO and CTR=PRD
    EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;

    // Setup compare
    EPwm1Regs.CMPA.bit.CMPA = period>>1;      // Master duty cycle = 0.5. The value doesn't matter in this configuration. Only used for monitoring epwm1a pin. (PIN#160)

    // Set actions
    EPwm1Regs.AQCTLA.bit.PRD = AQ_SET;
    EPwm1Regs.AQCTLA.bit.ZRO = AQ_CLEAR;

    // Enable master-PWM interrupt pulse generations
    EPwm1Regs.ETSEL.bit.SOCAEN  = 1;             // enable SOC on A group
    EPwm1Regs.ETSEL.bit.SOCASEL = ET_CTR_PRD;    // Select SOC at PRD, so that the measurements are aligned with epwm4b-mid point, and duty cycle update occurs before epwm4a goes low. Measurements occur only 5 times during switch transitions, tested for 0.1<D<0.95. Note that epwm4 controls SW3, a.k.a the middle switch pair in FCML.
    EPwm1Regs.ETPS.bit.SOCAPRD = 1;              // Generate pulse on 1st event

    EDIS;
}

void InitEPwmFCML(volatile struct EPWM_REGS * pwmregs, int32 period, int32 phase, int16 dir){

    EALLOW;

    pwmregs->TBPHS.all = 0x00000000;                 // Reset time-base counter phase relative to the time-base that is supplying the synchronization input
    pwmregs->TBPRD = period;                         // Set timer period. Note PWM counting starts from 0
    pwmregs->TBPRDHR = 0;
    pwmregs->TBPHS.bit.TBPHS = phase;                // Set phase shift
    pwmregs->TBCTL.bit.PHSDIR = dir;                 // phase shift direction
    pwmregs->TBCTR = 0x0000;                         // Clear counter

    // Set up TBCLK
```

```
    pwmregs->TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;    // Count up and down
    pwmregs->TBCTL.bit.PHSEN = TB_ENABLE;            // Phase loading
    pwmregs->TBCTL.bit.HSPCLKDIV = TB_DIV1;          // Clock ratio to SYSCLKOUT
    pwmregs->TBCTL.bit.CLKDIV = TB_DIV1;             // Slow to observe on the scope
    pwmregs->TBCTL.bit.SYNCOSEL = TB_SYNC_IN;        // output EPWMxSYNCO from EPWMxSYNCI

    // Setup compare register loading
    pwmregs->CMPCTL.bit.LOADAMODE = CC_CTR_PRD;     // load from shadow register from at both CTR=
    ZERO and CTR=PRD
    pwmregs->CMPCTL.bit.SHDWAMODE = CC_SHADOW;

    // Setup compare
    pwmregs->CMPA.bit.CMPA = period>>1;
   // pwmregs->CMPA.bit.CMPAHR = (1<<8);              // From Shibin's code, not sure why set to 256
    since HR mode is not used. EC

    // Below registers assume:
    // 1. PWMA is high side, PWMB is low side.
    // 2. main_duty controls high side.
    // 3. deadtime controls the time gap between the falling and rising edge of PWMA and PWMB.
    // 4. deadtime > 2*PERIOD clears both PWMA and PWMB outputs at the same time. Used to FCML
    switch pairs OFF when needed. Tested for 0.005 < main_duty < 0.995, and 1 < deadtime_hs &
    deadtime_ls < 10.

    // Set actions
    pwmregs->AQCTLA.bit.CAU = AQ_SET;
    pwmregs->AQCTLA.bit.CAD = AQ_CLEAR;

    // Setup the deadband
    pwmregs->DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
    pwmregs->DBCTL.bit.POLSEL = DB_ACTV_HIC;
    pwmregs->DBCTL.bit.IN_MODE = DBA_ALL;
    pwmregs->DBRED = deadtime_hs;
    pwmregs->DBFED = deadtime_ls;

    EDIS;
}

void bias_measurement()
{
    // seems that the first ADC reading might not be accurate, do a dummy read
    while (AdcaRegs.ADCINTFLG.bit.ADCINT1 != 1); //wait for first set of measurement to finish
    dummy_read = AdcaResultRegs.ADCRESULT0;
    dummy_read = AdcaResultRegs.ADCRESULT1;
    dummy_read = AdcbResultRegs.ADCRESULT0;
    dummy_read = AdcbResultRegs.ADCRESULT1;
    dummy_read = AdccResultRegs.ADCRESULT0;
    dummy_read = AdccResultRegs.ADCRESULT1;
    dummy_read = AdcdResultRegs.ADCRESULT0;
    dummy_read = AdcdResultRegs.ADCRESULT1;
    AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //clear INT1 flag

    // wait........
    // make sure wait for 1s at least for all the external circuit to power on !!!!!
    // 1s is the measured delay from power on to current sensing amp has valid signal
    // otherwise the bias measurement might have unexpected error
    DELAY_US(1000000);

    int32 Iind_bias_sum = 0;
    int32 adc_count = 0;
    for (adc_count=0;adc_count<512;adc_count++)
```

```
    {
        while (AdcaRegs.ADCINTFLG.bit.ADCINT1 != 1); //wait first set of measurements to finish
        Iind_bias_sum += AdccResultRegs.ADCRESULT0;  //read result from ADCc0 for IL_bias

        AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //clear INT1 flag
    }
    Iind_bias = Iind_bias_sum>>9; // Divide by 2^9 (=512)
}

void ADC_calibration()
{
    int16 moving_ave_pointer = 0;
    int16 dummy_read_array[NUM_POINTS_HC]={0};
    int32 dummy_read_array_sum = 0;

    memset(dummy_read_array, 0, NUM_POINTS_HC);

    while(1)
    {
        while (AdcaRegs.ADCINTFLG.bit.ADCINT1 != 1); //Wait until ADC triggers.
        //Following commands are executed at SWITCHING_FREQUENCY

        //dummy_read = AdcaResultRegs.ADCRESULT0;  //Vac_neg
        dummy_read = AdcaResultRegs.ADCRESULT1;  //Vout
        //dummy_read = AdcbResultRegs.ADCRESULT0;  //Vac_pos
        //dummy_read = AdcbResultRegs.ADCRESULT1;  //
        //dummy_read = AdccResultRegs.ADCRESULT0;  //Iind
        //dummy_read = AdccResultRegs.ADCRESULT1;  //
        //dummy_read = AdcdResultRegs.ADCRESULT0;  //Vrec
        //dummy_read = AdcdResultRegs.ADCRESULT1;

        //Although not needed, perform moving average for dummy_read to read stable numbers in Expressions panel
        dummy_read_array_sum = dummy_read_array_sum + dummy_read - dummy_read_array[moving_ave_pointer]; // Sum = Sum + newest value - oldest value
        dummy_read_array[moving_ave_pointer] = dummy_read;                        // Replace the sample value
        dummy_read_moving_ave = dummy_read_array_sum*MOV_AVE_DIVIDER*REG2ADCIN;                  // Update the moving average, This should correspond to the voltage at the ADC pin.

        moving_ave_pointer++;                                      // Move the pointer forward
        if (moving_ave_pointer==NUM_POINTS_HC)                     // Reset moving_ave_pointer at every NUM_POINTS_HC iterations
            moving_ave_pointer = 0;

        AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //Clear ADC flag
    }
}
```

Listing E.3: initialize.h

```
#ifndef INITIALIZE_H_
#define INITIALIZE_H_

#include "global_define.h"

// Function definitions
void DisableAllPeripheralClks();
void InitRectifierGPIOs(); //Initialize active rectifier control GPIO pins
void InitDebugGPIOs(); //Initialize debug GPIO pins
```

```

void InitInterrupts(void); // initialize necessary interrupts
interrupt void adc_trigger(void);

void InitADCs(void);
void InitADCa(void); // Initialize ADCa, measure Vac_neg on A2 (SOC0)
void InitADCb(void); // Initialize ADCb, measure Vac_pos on B2 (SOC0)
void InitADCc(void); // Initialize ADCc, measure current on C4 (SOC0)
void InitADCd(void); // Initialize ADCd, measure Vrect  on D2 (SOC0)

void bias_measurement(void); // measure bias value from current amplifier

void ADC_calibration(void); // ADC calibration

void InitDACs(); // initialize DAC A, B and C

void InitEPwmModules(void); //initialize ePWM modules
void InitEPwmMaster(int32 period); //initialize ePWM1 as clock master
void InitEPwmFCML(volatile struct EPWM_REGS * pwmregs, int32 period, int32 phase, int16 dir); // initialize ePWM2 - ePWM6 as FCML control signals

// FCML control variables
extern volatile float D;
extern int16 deadtime_hs;
extern int16 deadtime_ls;
extern int16 Iind_bias;
extern int16 dummy_read;
extern float32 dummy_read_moving_ave;
//extern int16 dir;

// Function to clear a block of memory
void memset(void *mem, int ch, size_t length);

#endif /* INITIALIZE_H_ */
```

Listing E.4: operation.c

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


void Update_duty(float32 duty)
{
    Uint16 d = 0;

    if (duty > 1) //This case is only executed if main_duty is entered by mistake in dc-dc mode
        d = 0.5;  //AC-DC mode must have it's own saturation block. d is never bigger than 0.995.
    else if (duty < 0) //This case is only executed if main_duty is entered by mistake in dc-dc mode
        d = 0.5;  //AC-DC mode must have it's own saturation block. d is never bigger than 0.995.
    else
        d = (1-duty)*PERIOD; //To control high side switch on-time with duty around COUNT_UP_DOWN peak, not zero

    EPwm2Regs.CMPA.bit.CMPA = d;
    EPwm3Regs.CMPA.bit.CMPA = d;
    EPwm4Regs.CMPA.bit.CMPA = d;
    EPwm5Regs.CMPA.bit.CMPA = d;
    EPwm6Regs.CMPA.bit.CMPA = d;

```

```
}

void Update_deadtime(float32 deadtime_hs, float32 deadtime_ls)
{

    EPwm2Regs.DBRED = deadtime_hs;
    EPwm2Regs.DBFED = deadtime_ls;
    EPwm3Regs.DBRED = deadtime_hs;
    EPwm3Regs.DBFED = deadtime_ls;
    EPwm4Regs.DBRED = deadtime_hs;
    EPwm4Regs.DBFED = deadtime_ls;
    EPwm5Regs.DBRED = deadtime_hs;
    EPwm5Regs.DBFED = deadtime_ls;
    EPwm6Regs.DBRED = deadtime_hs;
    EPwm6Regs.DBFED = deadtime_ls;
}

void Update_PS_dir(int16 dir)
{
    EPwm2Regs.TBCTL.bit.PHSDIR = dir;
    EPwm3Regs.TBCTL.bit.PHSDIR = dir;
    EPwm5Regs.TBCTL.bit.PHSDIR = 1-dir;
    EPwm6Regs.TBCTL.bit.PHSDIR = 1-dir;
}
```

Listing E.5: operation.h

```
#ifndef OPERATION_H_
#define OPERATION_H_

#include "global_define.h"



#endif /* OPERATION_H_ */

void Update_duty(float duty);  // change duty ratio to a given value
void Update_deadtime(float deadtime_hs, float deadtime_ls);  // change deadtime to a given value
void Update_PS_dir(int dir);
```

APPENDIX F

ADDITIONAL EXPERIMENTAL RESULTS WITH SIX-LEVEL BUCK CONVERTER

F.1 Step-down dc-dc application

Although the six-level buck converter prototype is designed for single-phase PFC application, the FCML buck stage shown in Figure 8.2 can be operated as a step-down dc-dc converter. As mentioned in Section 2.1, nominal 400 V is a common voltage level in data center applications; therefore, the six-level FCML converter is configured as a 400 V to 48 V dc-dc converter to showcase its performance as a high voltage step-down dc-dc converter. Using the same 100 V rated GaN transistors, the six-level buck converter stage of the hardware prototype in Section 8.1 can withstand 400 V input voltage with 20% margin. The switching frequency, flying capacitor and inductor of the FCML buck stage are updated as in Table F.1 to increase the efficiency and output power for 400 V to 48 V dc-dc step-down application. The updated FCML buck stage for step-down dc-dc conversion still fits in a box volume of 1.63 in^3. Experimental results provided here are achieved with the same heat sink mentioned in Section 8.1, resulting in 3.38 in^3 total converter box volume.

Efficiency of the six-level buck stage for 380/400 V to 48 V dc-dc conversion is given in Figure F.1. For 380/400 V dc input voltage, the six-level buck converter can provide up to 775 W output power, yielding 229 W/ in^3 power density. At 380 V input peak and full load efficiency are 97% and 94.1%, respectively. At 400 V dc input voltage, peak and full load efficiency are 96.7% and 94.5%, respectively. If cooled down with forced air, at 400 V dc input voltage, the six-level buck converter output power can be increased to 1100 W, yielding 325 W/ in^3 power density. It achieves 96.7% peak and 91.9% full load efficiency.

F.2 Power factor correction application

The power factor at 240 V_{RMS} input voltage can be increased by adding a 10 Ω resistor in series between the power supply and the converter to damp distortion of the input current waveform. The

Table F.1: Updated components of the FCML buck stage for step-down dc-dc application

Component	Manufacturer & Part Number	Details
Flying capacitor	TDK C5750X6S	450 V, 2.2 μF, 8 in parallel per level
Inductor	Vishay ILHP5050EZER	5.6 μH, 2 in series
Input capacitor	TDK C5750X6S	450 V, 2.2 μF, 8 in parallel
Switching Frequency	60 kHz	

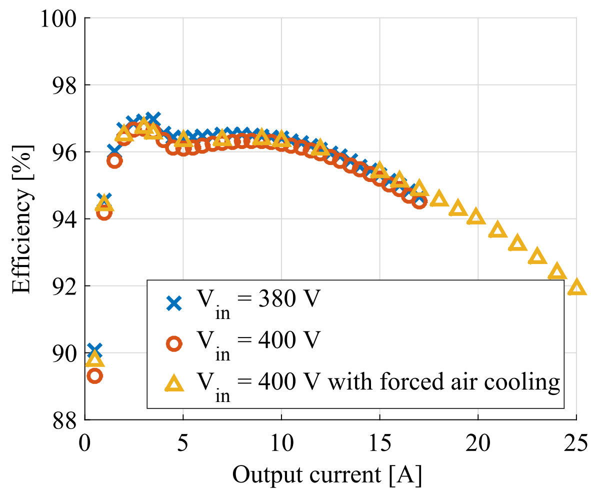


Figure F.1: Six-level FCML buck converter dc-dc conversion efficiency at 400 V input voltage.

six-level FCML buck converter then can achieve 0.8386 power factor and 92.083% power conversion efficiency (excluding 13.67 W power loss on the series input resistor) at rated current, as can be seen in Figure F.2.

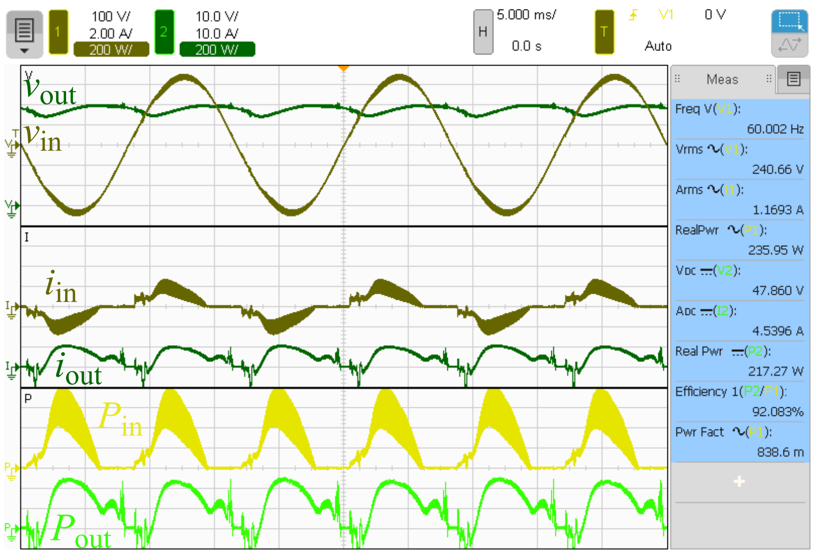


Figure F.2: The input and output voltage, current, and power of the six-level buck converter in PFC operation at 240 V_{RMS} input voltage with 10 Ω series resistor between the ac power supply and the converter.

REFERENCES

[1] J. G. Koomey, "Estimating total power consumption by servers in the us and the world," Lawrence Berkeley National Laboratory, Tech. Rep., 2007.

[2] R. E. Brown, E. R. Masanet, B. Nordman, W. F. Tschudi, A. Shehabi, J. Stanley, J. G. Koomey, D. A. Sartor, and P. T. Chan, "Report to congress on server and data center energy efficiency: public law 109-431," Lawrence Berkeley National Laboratory, Tech. Rep., 2007.

[3] A. Shehabi, S. J. Smith, D. A. Sartor, R. E. Brown, M. Herrlin, J. G. Koomey, E. R. Masanet, N. Horner, I. L. Azevedo, and W. Lintner, "United states data center energy usage report," Ernest Orlando Lawrence Berkeley National Laboratory, Tech. Rep. LBNL-1005775, June 2016 2016.

[4] R. Ascierto, "Uptime institute global data center survey," Uptime Institute, Tech. Rep., 2018.

[5] P. T. Krein, "Data center challenges and their power electronics," *CPSS Trans. Power Electron. Appl.*, vol. 2, no. 1, 2017.

[6] P. S. Shenoy and P. T. Krein, "Differential power processing for dc systems," *IEEE Trans. Power Electron.*, vol. 28, no. 4, pp. 1795–1806, April 2013.

[7] E. Candan, "A series-stacked power delivery architecture with isolated converters for energy efficient data centers," M.S. thesis, University of Illinois at Urbana-Champaign, 2014.

[8] Y. Lei, W. C. Liu, and R. C. N. Pilawa-Podgurski, "An analytical method to evaluate and design hybrid switched-capacitor and multilevel converters," *IEEE Trans. Power Electron.*, vol. 33, no. 3, pp. 2227–2240, March 2018.

[9] E. Candan, D. Heeger, P. Shenoy, and R. Pilawa-Podgurski, "Hot-swapping analysis and implementation of series-stacked server power delivery architectures," *IEEE Tran. Power Electron.*, vol. 32, no. 10, pp. 18 071–8088, Oct 2017.

[10] E. Candan, P. S. Shenoy, and R. C. N. Pilawa-Podgurski, "A distributed bi-directional hysteresis control algorithm for server-to-virtual bus differential power processing," in *Proc. IEEE 16th Workshop Control and Modeling Power Electron.*, July 2015, pp. 1–8.

[11] E. Candan, P. S. Shenoy, and R. C. N. Pilawa-Podgurski, "Unregulated bus operation of server-to-virtual bus differential power processing for data centers," in *Proc. 32nd Annu. IEEE Appl. Power Electron. Conf. and Expo.*, March 2017, pp. 1632–1639.

[12] E. Candan, P. S. Shenoy, and R. C. N. Pilawa-Podgurski, "A 6-level flying capacitor multilevel converter for single phase buck-type power factor correction," in *Proc. 34th Annu. IEEE Appl. Power Electron. Conf. and Expo.*, March 2017.

[13] E. Candan and R. C. Pilawa-Podgurski, "DC data centers," in *DC Distribution Systems and Microgrids*, T. Dragičević, P. Wheeler, and F. Blaabjerg, Eds. UK: Institution of Engineering and Technology, 2018, pp. 343–366. [Online]. Available: https://digital-library.theiet.org/content/books/10.1049/pbpo115e_ch14

[14] N. Rasmussen, "Ac vs. dc power distribution for data centers," Schneider Electric, White Paper, 2011. [Online]. Available: https://www.apc.com/salestools/SADE-5TNRLG/SADE-5TNRLG_R6_EN.pdf

[15] D. F. D. Tan, "A review of immediate bus architecture: A system perspective," *IEEE J. Emerging Sel. Topics Power Electron.*, vol. 2, no. 3, pp. 363–373, Sept 2014.

[16] M. Ton, B. Fortenbery, and W. Tschudi, "Dc power for improved data center efficiency," Lawrence Berkeley National Laboratory, Report, 2008. [Online]. Available: https://datacenters.lbl.gov/sites/all/files/DC%20Power%20Demo_2008.pdf

[17] T. Yamashita, S. Muroyama, S. Furubo, and S. Ohtsu, "270 v dc system - a highly efficient and reliable power supply system for both telecom and datacom systems," in *Proc. 21st IEEE Int. Telecommun. Energy Conf.*, Jun 1999, pp. 1–6.

[18] U. Carlsson, M. Flodin, J. Akerlund, and A. Ericsson, "Powering the internet - broadband equipment in all facilities - the need for a 300 v dc powering and universal current option," in *Proc. 25th IEEE Int. Telecommun. Energy Conf.*, Oct 2003, pp. 164–169.

[19] A. Sannino, G. Postiglione, and M. H. J. Bollen, "Feasibility of a dc network for commercial facilities," *IEEE Trans. Ind. Appl.*, vol. 39, no. 5, pp. 1499–1507, Sept 2003.

[20] F. Bodi and E. H. Lim, "380/400 v dc powering option," in *Proc. 33rd IEEE Int. Telecommun. Energy Conf.*, Oct 2011, pp. 1–8.

[21] M. Salato, A. Zolj, D. J. Becker, and B. J. Sonnenberg, "Power system architectures for 380v dc distribution in telecom datacenters," in *Proc. IEEE Int. Telecommun. Energy Conf.*, Sept 2012, pp. 1–7.

[22] A. Pratt, P. Kumar, and T. V. Aldridge, "Evaluation of 400v dc distribution in telco and data centers to improve energy efficiency," in *Proc. 29th IEEE Int. Telecommun. Energy Conf.*, Sept 2007, pp. 32–39.

[23] *Final draft ETSI EN 300 132-3-1 environmental engineering (EE); power supply interface at the input to telecommunications and datacom (ICT) equipment; part 3: operated by rectified current source, alternating current source or direct current source up to 400 V; sub-part 1: direct current source up to 400 V*, European Telecommunications Standards Institute Std. EN 300 123-3-1, Rev. 2.1.1, 10 2011.

[24] *Draft ETSI EN 301 605 environmental engineering (EE); earthing and bonding of 400 VDC data and telecom (ICT) equipment*, European Telecommunications Standards Institute Std. ETSI EN 301 605, Rev. V1.1.1, 10 2013.

[25] *IEC 61643-21 low voltage surge protective devices part 21: surge protective devices connected to telecommunications and signalling networks performance requirements and testing methods*, International Electrotechnical Commission Std. IEC 61 643-21, Rev. 1.2, 07 2012.

[26] T. Aldridge, A. Pratt, P. Kumar, D. Dupy, and G. AlLee, "Evaluating 400v direct-current for data centers a case study comparing 400 vdc with 480-208 vac power distribution for energy efficiency and other benefits," Intel Labs, White Paper, 2010. [Online]. Available: https://blogs.intel.com/wp-content/mt-content/com/research/Direct%20400Vdc%20White%20Paper.pdf

[27] V. Sithimolada and P. W. Sauer, "Facility-level dc vs. typical ac distribution for data centers: a comparative reliability study," in *Proc. TENCON IEEE Region 10 Conf.*, Nov 2010, pp. 2102–2107.

[28] D. Kintner, "Duke energy - EPRI dc powered data center demonstration executive summary," Electric Power Research Institute (EPRI), Tech. Rep., 2011.

[29] N. Rasmussen and J. Spitaels, "A quantitative comparison of high efficiency ac vs. dc power distribution for data centers," Schneider Electric, White Paper, 2012. [Online]. Available: https://www.apc.com/salestools/NRAN-76TTJY/NRAN-76TTJY_R4_EN.pdf

[30] M. Szpek, B. J. Sonnenberg, and S. M. Lisy, "400vdc distribution architectures for central offices and data centers," in *Proc. 36th IEEE Int. Telecommun. Energy Conf.*, Sept 2014, pp. 1–6.

[31] N. Rasmussen, "Review of four studies comparing efficiency of ac and dc distribution for data centers," Schneider Electric, White Paper, 2012. [Online]. Available: https://www.schneider-electric.com.ar/es/download/document/APC_VAVR-8Q7K7N_EN/

[32] L. Schrittwieser, J. W. Kolar, and T. B. Soeiro, "99% efficient three-phase buck-type SiC MOSFET PFC rectifier minimizing life cycle cost in dc data centers," *CPSS Trans. Power Electron. Appl.*, vol. 2, no. 1, pp. 47–58, 2017.

[33] F. Xu, B. Guo, L. M. Tolbert, F. Wang, and B. J. Blalock, "An all-SiC three-phase buck rectifier for high-efficiency data center power supplies," *IEEE Trans. Ind. Appl.*, vol. 49, no. 6, pp. 2662–2673, Nov 2013.

[34] Z. Liu, F. C. Lee, Q. Li, and Y. Yang, "Design of GaN-based MHz totem-pole PFC rectifier," *IEEE J. Emerging Sel. Topics Power Electron.*, vol. 4, no. 3, pp. 799–807, Sept 2016.

[35] M. Kasper, D. Bortis, G. Deboy, and J. W. Kolar, "Design of a highly efficient (97.7%) and very compact (2.2 kw/dm^3) isolated acdc telecom power supply module based on the multicell isop converter approach," *IEEE Trans. Power Electron.*, vol. 32, no. 10, pp. 7750–7769, Oct 2017.

[36] 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 J. Emerging Sel. Topics Power Electron.*, pp. 1–1, 2018.

[37] D. Huang, S. Ji, and F. C. Lee, "LLC resonant converter with matrix transformer," *IEEE Trans. Power Electron.*, vol. 29, no. 8, pp. 4339–4347, Aug 2014.

[38] W. Zhang, B. Guo, F. Xu, Y. Cui, Y. Long, F. Wang, L. M. Tolbert, B. J. Blalock, and D. J. Costinett, "Wide bandgap power devices based high efficiency power converters for data center application," in *Proc. 2nd IEEE Workshop Wide Bandgap Power Devices and Appl.*, Oct 2014, pp. 121–126.

[39] M. D. Seeman, S. R. Bahl, D. I. Anderson, and G. A. Shah, "Advantages of GaN in a high-voltage resonant LLC converter," in *Proc. 29th Annu. IEEE Appl. Power Electron. Conf. and Expo.*, March 2014, pp. 476–483.

[40] J. Biela, U. Badstuebner, and J. W. Kolar, "Design of a 5-kW, 1-U, 10-kW/dm3 resonant dc-dc converter for telecom applications," *IEEE Trans. Power Electron.*, vol. 24, no. 7, pp. 1701–1710, July 2009.

[41] Z. Ye, Y. Lei, and R. C. N. Pilawa-Podgurski, "A resonant switched capacitor based 4-to-1 bus converter achieving 2180W/in3 power density and 98.9% peak efficiency," in *Proc. 33rd Annu. IEEE Appl. Power Electron. Conf. and Expo.*, March 2018, pp. 121–126.

[42] D. Reusch and J. Strydom, "Evaluation of gallium nitride transistors in high frequency resonant and soft-switching dc-dc converters," *IEEE Trans. Power Electron.*, vol. 30, no. 9, pp. 5151–5158, Sept 2015.

[43] Y. Li, X. Lyu, D. Cao, S. Jiang, and C. Nan, "A 98.55% efficiency switched-tank converter for data center application," *IEEE Trans. Ind. Appl.*, pp. 1–1, 2018.

[44] D. Cao, X. Lyu, Y. Li, Z. Ni, J. Johnson, S. Jiang, and C. Nan, "An ultra efficient composite modular power delivery architecture for solar farm and data center," in *Proc. 33rd Annu. IEEE Appl. Power Electron. Conf. and Expo.*, March 2018, pp. 73–80.

[45] J. S. Rentmeister and J. T. Stauth, "A 48v:2v flying capacitor multilevel converter using current-limit control for flying capacitor balance," in *Proc. 32nd Annu. IEEE Appl. Power Electron. Conf. and Expo.*, March 2017, pp. 367–372.

[46] A. Kumar and K. K. Afridi, "Single-stage isolated 48v-to-1.8v point-of-load converter utilizing an impedance control network for wide input range operation," in *Proc. IEEE Energy Conversion Congr. and Expo.*, Oct 2017, pp. 2003–2009.

[47] M. Chen, K. K. Afridi, S. Chakraborty, and D. J. Perreault, "Multitrack power conversion architecture," *IEEE Trans. Power Electron.*, vol. 32, no. 1, pp. 325–340, Jan 2017.

[48] L. Jia, S. Lakshmikanthan, X. Li, and Y. F. Liu, "New modeling method and design optimization for a soft-switched dc-dc converter," *IEEE Trans. Power Electron.*, vol. 33, no. 7, pp. 5754–5772, July 2018.

[49] D. Reusch, F. C. Lee, D. Gilham, and Y. Su, "Optimization of a high density gallium nitride based non-isolated point of load module," in *Proc. IEEE Energy Conversion Congr. and Expo.*, Sept 2012, pp. 2914–2920.

[50] B. K. Rhea, L. L. Jenkins, W. E. Abell, F. T. Werner, C. G. Wilson, R. N. Dean, and D. K. Harris, "A 12 to 1 v five phase interleaving GaN POL converter for high current low voltage applications," in *Proc. 2nd IEEE Workshop Wide Bandgap Power Devices and Appl.*, Oct 2014, pp. 155–158.

[51] L. L. Jenkins, C. G. Wilson, J. D. Moses, J. M. Aggas, B. K. Rhea, and R. N. Dean, "Optimization of a 96% efficient 121 v gallium nitride based point of load converter," in *Proc. 29th Annu. IEEE Appl. Power Electron. Conf. and Expo.*, March 2014, pp. 2098–2104.

[52] P. S. Shenoy, M. Amaro, J. Morroni, and D. Freeman, "Comparison of a buck converter and a series capacitor buck converter for high-frequency, high-conversion-ratio voltage regulators," *IEEE Trans. Power Electron.*, vol. 31, no. 10, pp. 7006–7015, Oct 2016.

[53] W. Lee and B. Sarlioglu, "Design and analysis of integrated planar inductor for GaN HEMT-based zero-voltage switching synchronous buck converter," in *Proc. IEEE Power Energy Conf. at Illinois*, Feb 2018, pp. 1–6.

[54] P. T. Krein, R. S. Balog, and M. Mirjafari, "Minimum energy and capacitance requirements for single-phase inverters and rectifiers using a ripple port," *IEEE Trans. Power Electron.*, vol. 27, no. 11, pp. 4690–4698, Nov 2012.

[55] "Detailed inverter specifications, testing procedure, and technical approach and testing application requirements for the little box challenge." [Online]. Available: https://littleboxchallenge.com/pdf/LBC-InverterRequirements-20150717.pdf

[56] S. Qin, Y. Lei, C. Barth, W. C. Liu, and R. C. N. Pilawa-Podgurski, "A high power density series-stacked energy buffer for power pulsation decoupling in single-phase converters," *IEEE Trans. Power Electron.*, vol. 32, no. 6, pp. 4905–4924, June 2017.

[57] U. Anwar, D. Maksimovic, and K. K. Afridi, "A simple control architecture for four-switch buck-boost converter based power factor correction rectifier," in *Proc. IEEE 18th Workshop Control and Modeling Power Electron.*, July 2017, pp. 1–6.

[58] D. Reusch and J. Glaser, *Dc-dc converter handbook - a supplement to GaN transistors for efficient power conversion.* Efficient Power Conversion Corporation, 2015.

[59] L. A. Barroso, J. Clidaras, and U. Hölzle, *The datacenter as a computer: an introduction to the design of warehouse-scale machines*, 2nd ed. Morgan & Claypool, 2013.

[60] D. Maiquet and G. Kervarrec, "New flexible powering architecture for integrated service operators," in *Proc. Twenty-Seventh IEEE Int. Telecommun. Energy Conf.*, Sept 2005, pp. 575–580.

[61] G. AlLee and W. Tschudi, "Edison redux: 380 vdc brings reliability and efficiency to sustainable data centers," *IEEE Power Energy Mag.*, vol. 10, no. 6, pp. 50–59, Nov 2012.

[62] W. P. Turner, J. H. Seader, and V. E. Renaud, "Tier standard: topology," Uptime Institute, Tech. Rep., 2012. [Online]. Available: https://uptimeinstitute.com/publications/asset/tier-standard-topology

[63] *1100 IEEE recommended practice for powering and grounding electronic equipment*, Institute of Electrical and Electronics Engineers Std. IEEE STD 1100-2005, December 2015.

[64] M. Chen, M. Araghchini, K. K. Afridi, J. H. Lang, C. R. Sullivan, and D. J. Perreault, "A systematic approach to modeling impedances and current distribution in planar magnetics," *IEEE Trans. Power Electron.*, vol. 31, no. 1, pp. 560–580, Jan 2016.

[65] U. Badstuebner, J. Biela, and J. W. Kolar, "Design of an 99%-efficient, 5kW, phase-shift pwm dc-dc converter for telecom applications," in *Proc. 25th Annu. IEEE Appl. Power Electron. Conf. and Expo.*, Feb 2010, pp. 773–780.

[66] R. Simanjorang, H. Yamaguchi, H. Ohashi, K. Nakao, T. Ninomiya, S. Abe, M. Kaga, and A. Fukui, "High-efficiency high-power dc-dc converter for energy and space saving of power-supply system in a data center," in *Proc. 26th Annu. IEEE Appl. Power Electron. Conf. and Expo.*, March 2011, pp. 600–605.

[67] *Vicor BCM bus converter BCM48Bx480y300A00 isolated fixed ratio dc-dc converter*, 1st ed., Vicorpower, 08 2016. [Online]. Available: http://cdn.vicorpower.com/documents/datasheets/BCM48B_480_300A00.pdf

[68] S. V. Araujo, P. Zacharias, and B. Sahan, "Novel grid-connected non-isolated converters for photovoltaic systems with grounded generator," in *Proc. 29th IEEE Annu. Power Electron. Specialists Conf.*, June 2008, pp. 58–65.

[69] Y. Sverdlik, "Apple reaches 100% renewable energy across all data centers," 2013. [Online]. Available: https://www.datacenterdynamics.com/news/apple-reaches-100-renewable-energy-across-all-data-centers/

[70] Y. Sverdlik, "eBay's Utah data center offers a glimpse into the future," 2013. [Online]. Available: https://www.datacenterdynamics.com/news/ebays-utah-data-center-offers-a-glimpse-into-the-future/

[71] R. C. N. Pilawa-Podgurski and D. J. Perreault, "Submodule integrated distributed maximum power point tracking for solar photovoltaic applications," *IEEE Trans. Power Electron.*, vol. 28, no. 6, pp. 2957–2967, June 2013.

[72] S. Qin, S. T. Cady, A. D. Domínguez-García, and R. C. N. Pilawa-Podgurski, "A distributed approach to maximum power point tracking for photovoltaic submodule differential power processing," *IEEE Trans. Power Electron.*, vol. 30, no. 4, pp. 2024–2040, April 2015.

[73] T. P. Foulkes, "Developing an active, high-heat-flux thermal management strategy for power electronics via jumping-droplet phase-change cooling," M.S. thesis, University of Illinois at Urbana-Champaign, 2017.

[74] T. Foulkes, J. Oh, P. Birbarah, J. Neely, N. Miljkovic, and R. C. N. Pilawa-Podgurski, "Active hot spot cooling of GaN transistors with electric field enhanced jumping droplet condensation," in *Proc. 32nd Annu. IEEE Appl. Power Electron. Conf. and Expo.*, March 2017, pp. 912–918.

[75] J. Koomey, "A simple model for determining true total cost of ownership for data centers," Uptime Institute, White Paper, 2007.

[76] N. Rasmussen, "Determining total cost of ownership for data center and network room infrastructure," Schneider Electric, White paper, 2011. [Online]. Available: https://www.apc.com/salestools/CMRP-5T9PQG/CMRP-5T9PQG_R4_EN.pdf

[77] E. M. Landsman, "Scalable data center architecture for on-demand power infrastructure," in *Proc. Eighteenth Annu. IEEE Appl. Power Electron. Conf. and Expo.*, vol. 1, Feb 2003, pp. 10–13 vol.1.

[78] T. Shimizu, M. Hirakata, T. Kamezawa, and H. Watanabe, "Generation control circuit for photovoltaic modules," *IEEE Trans. Power Electron.*, vol. 16, no. 3, pp. 293–300, May 2001.

[79] T. Shimizu, O. Hashimoto, and G. Kimura, "A novel high-performance utility-interactive photovoltaic inverter system," *IEEE Trans. Power Electron.*, vol. 18, no. 2, pp. 704–711, March 2003.

[80] G. R. Walker and J. C. Pierce, "Photovoltaic dc-dc module integrated converter for novel cascaded and bypass grid connection topologies design and optimization," in *Proc. 37th IEEE Power Electron. Specialists Conf.*, June 2006, pp. 1–7.

[81] P. Shenoy, K. Kim, B. Johnson, and P. Krein, "Differential power processing for increased energy production and reliability of photovoltaic systems," *IEEE Trans. Power Electron.*, vol. 28, no. 6, pp. 2968–2979, June 2013.

[82] C. Olalla, D. Clement, M. Rodriguez, and D. Maksimovic, "Architectures and control of submodule integrated dc-dc converters for photovoltaic applications," *IEEE Trans. Power Electron.*, vol. 28, no. 6, pp. 2980–2997, June 2013.

[83] C. Olalla, C. Deline, and D. Maksimovic, "Performance of mismatched PV systems with submodule integrated converters," *IEEE J. Photovoltaics*, vol. 4, no. 1, pp. 396–404, Jan 2014.

[84] C. Schaef and J. T. Stauth, "Multilevel power point tracking for partial power processing photovoltaic converters," *IEEE J. Emerging Sel. Topics Power Electron.*, vol. 2, no. 4, pp. 859–869, Dec 2014.

[85] Y. Levron, D. R. Clement, B. Choi, C. Olalla, and D. Maksimovic, "Control of submodule integrated converters in the isolated-port differential power-processing photovoltaic architecture," *IEEE J. Emerging Sel. Topics Power Electron.*, vol. 2, no. 4, pp. 821–832, Dec 2014.

[86] C. Olalla, C. Deline, D. Clement, Y. Levron, M. Rodriguez, and D. Maksimovic, "Performance of power-limited differential power processing architectures in mismatched pv systems," *IEEE Trans. Power Electron.*, vol. 30, no. 2, pp. 618–631, Feb 2015.

[87] J. Galtieri and P. T. Krein, "Impact of differential power processing on inter-row shading in solar arrays," in *Proc. IEEE 16th Workshop Control and Modeling Power Electron.*, July 2015, pp. 1–8.

[88] R. Bell and R. C. N. Pilawa-Podgurski, "Decoupled and distributed maximum power point tracking of series-connected photovoltaic submodules using differential power processing," *IEEE J. Emerging Sel. Topics Power Electron.*, vol. 3, no. 4, pp. 881–891, Dec 2015.

[89] S. Qin, C. B. Barth, and R. C. N. Pilawa-Podgurski, "Enhancing microinverter energy capture with submodule differential power processing," *IEEE Trans. Power Electron.*, vol. 31, no. 5, pp. 3575–3585, May 2016.

[90] K. Doubleday, B. Choi, D. Maksimovic, C. Deline, and C. Olalla, "Recovery of inter-row shading losses using differential power-processing submodule dc-dc converters," *Solar Energy*, vol. 135, pp. 512 – 517, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0038092X1630192X

[91] Y. T. Jeon, H. Lee, K. A. Kim, and J. H. Park, "Least power point tracking method for photovoltaic differential power processing systems," *IEEE Trans. Power Electron.*, vol. 32, no. 3, pp. 1941–1951, March 2017.

[92] C. Liu, D. Li, Y. Zheng, and B. Lehman, "Modular differential power processing (mDPP)," in *Proc. IEEE 18th Workshop Control and Modeling Power Electron.*, July 2017, pp. 1–7.

[93] C. Liu, Y. Zheng, D. Li, and B. Lehman, "Distributed MPPT for modular differential power processing in scalable photovoltaic system," in *Proc. 33rd Annu. IEEE Appl. Power Electron. Conf. and Expo.*, March 2018, pp. 1098–1103.

[94] S. Rajapandian, K. L. Shepard, P. Hazucha, and T. Karnik, "High-voltage power delivery through charge recycling," *IEEE J. Solid-State Circuits*, vol. 41, no. 6, pp. 1400–1410, June 2006.

[95] E. K. Ardestani, R. T. Possignolo, J. L. Briz, and J. Renau, "Managing mismatches in voltage stacking with coreunfolding," *ACM Trans. Archit. Code Optim.*, vol. 12, no. 4, pp. 43:1–43:26, Nov. 2015. [Online]. Available: http://doi.acm.org/10.1145/2835178

[96] C. Schaef and J. T. Stauth, "Efficient voltage regulation for microprocessor cores stacked in vertical voltage domains," *IEEE Trans. Power Electron.*, vol. 31, no. 2, pp. 1795–1808, Feb 2016.

[97] T. Tong, S. K. Lee, X. Zhang, D. Brooks, and G. Y. Wei, "A fully integrated reconfigurable switched-capacitor dc-dc converter with four stacked output channels for voltage stacking applications," *IEEE J. Solid-State Circuits*, vol. 51, no. 9, pp. 2142–2152, Sept 2016.

[98] E. Ebrahimi, R. T. Possignolo, and J. Renau, "SRAM voltage stacking," in *Proc. IEEE Int. Symp. Circuits and Syst.*, May 2016, pp. 1634–1637.

[99] S. K. Lee, T. Tong, X. Zhang, D. Brooks, and G. Y. Wei, "A 16-core voltage-stacked system with adaptive clocking and an integrated switched-capacitor dc-dc converter," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 4, pp. 1271–1284, April 2017.

[100] K. Blutman, A. Kapoor, A. Majumdar, J. G. Martinez, J. Echeverri, L. Sevat, A. P. van der Wel, H. Fatemi, K. A. A. Makinwa, and J. P. de Gyvez, "A low-power microcontroller in a 40-nm CMOS using charge recycling," *IEEE J. Solid-State Circuits*, vol. 52, no. 4, pp. 950–960, April 2017.

[101] S. M. Ahsanuzzaman, A. Parayandeh, A. Prodić, and D. A. Johns, "A building block ic for designing emerging hybrid and multilevel converters," *IEEE J. Emerging Sel. Topics Power Electron.*, vol. 6, no. 2, pp. 500–514, June 2018.

[102] P. Wang and M. Chen, "Towards power FPGA: Architecture, modeling and control of multiport power converters," in *Proc. IEEE 19th Workshop Control and Modeling Power Electron.*, June 2018, pp. 1–8.

[103] Y. Yang and T. Lehmann, "Current recycling in linear regulators for biomedical implants," in *Proc. 53rd IEEE Int. Midwest Symp. Circuits and Syst.*, Aug 2010, pp. 545–548.

[104] Y. Yang, H. Chun, and T. Lehmann, "Dual-stacked current recycling linear regulators with 48% power saving for biomedical implants," *IEEE Trans. Circuits and Syst. I Reg. Papers,*, vol. 60, no. 7, pp. 1946–1958, July 2013.

[105] C. Pascual and P. Krein, "Switched capacitor system for automatic series battery equalization," in *Proc. Twelfth Annl. IEEE Appl. Power Electron. Conf. and Expo.*, vol. 2, Feb 1997, pp. 848–854 vol.2.

[106] N. H. Kutkut, H. L. N. Wiegman, D. M. Divan, and D. W. Novotny, "Design considerations for charge equalization of an electric vehicle battery system," *IEEE Trans. Ind. Appl.*, vol. 35, no. 1, pp. 28–35, Jan 1999.

[107] C. Karnjanapiboon, K. Jirasereeamornkul, and V. Monyakul, "High efficiency battery management system for serially connected battery string," in *IEEE Int. Symp. Ind. Electron.*, July 2009, pp. 1504–1509.

[108] M. Einhorn, W. Guertlschmid, T. Blochberger, R. Kumpusch, R. Permann, F. V. Conte, C. Kral, and J. Fleig, "A current equalization method for serially connected battery cells using a single power converter for each cell," *IEEE Trans. Veh. Technol.*, vol. 60, no. 9, pp. 4227–4237, Nov 2011.

[109] F. Baronti, G. Fantechi, R. Roncella, and R. Saletti, "High-efficiency digitally controlled charge equalizer for series-connected cells based on switching converter and super-capacitor," *IEEE Trans. Ind. Informatics*, vol. 9, no. 2, pp. 1139–1147, May 2013.

[110] M. Y. Kim, J. H. Kim, and G. W. Moon, "Center-cell concentration structure of a cell-to-cell balancing circuit with a reduced number of switches," *IEEE Trans. Power Electron.*, vol. 29, no. 10, pp. 5285–5297, Oct 2014.

[111] F. Mestrallet, L. Kerachev, J. C. Crebier, and A. Collet, "Multiphase interleaved converter for lithium battery active balancing," *IEEE Trans. Power Electron.*, vol. 29, no. 6, pp. 2874–2881, June 2014.

[112] M. Evzelman, M. M. U. Rehman, K. Hathaway, R. Zane, D. Costinett, and D. Maksimovic, "Active balancing system for electric vehicles with incorporated low-voltage bus," *IEEE Trans. Power Electron.*, vol. 31, no. 11, pp. 7887–7895, Nov 2016.

[113] M. Shousha, A. Prodić, V. Marten, and J. Milios, "Design and implementation of assisting converter-based integrated battery management system for electromobility applications," *IEEE J. Emerging Sel. Topics Power Electron.*, vol. 6, no. 2, pp. 825–842, June 2018.

[114] J. McClurg, Y. Zhang, J. Wheeler, and R. Pilawa-Podgurski, "Re-thinking data center power delivery: Regulating series-connected voltage domains in software," in *Proc. IEEE Power Energy Conf. at Illinois*, Feb 2013, pp. 147–154.

[115] E. Candan, P. S. Shenoy, and R. C. N. Pilawa-Podgurski, "A series-stacked power delivery architecture with isolated differential power conversion for data centers," *IEEE Trans. Power Electron.*, vol. 31, no. 5, pp. 3690–3703, May 2016.

[116] J. McClurg, R. Pilawa-Podgurski, and P. Shenoy, "A series-stacked architecture for high-efficiency data center power delivery," in *Proc. IEEE Energy Conversion Congr. and Expo.*, Sept 2014, pp. 170–177.

[117] Y. Zhang, E. Candan, and R. C. N. Pilawa-Podgurski, "A series-stacked architecture with 4-to-1 GaN-based isolated converters for high-efficiency data center power delivery," in *Proc. IEEE Energy Conversion Congr. and Expo.*, Oct 2017, pp. 4467–4474.

[118] D. Das and P. T. Krein, "Voltage regulation of a series-stacked system of processors by differential power processing," in *Proc. IEEE 16th Workshop Control and Modeling Power Electron.*, July 2015, pp. 1–7.

[119] A. Stillwell and R. C. N. Pilawa-Podgurski, "A resonant switched-capacitor converter with GaN transistors for series-stacked processors with 99.8% power delivery efficiency," in *Proc. IEEE Energy Conversion Congr. and Expo.*, Sept 2015, pp. 563–570.

[120] D. Das and P. T. Krein, "A bidirectional wide load range multiphase buck/boost converter for differential power processing," in *Proc. IEEE 18th Workshop Control and Modeling Power Electron.*, July 2017, pp. 1–7.

[121] P. S. Shenoy, "Improving performance, efficiency, and reliability of dc-dc conversion systems by differential power processing," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2012.

[122] B. Davies, "Analysis of inrush currents for dc powered IT equipment," in *Proc. IEEE 33rd Int. Telecommun. Energy Conf.*, Oct 2011, pp. 1–4.

[123] J. Alimeling and W. Hammer, "PLECS-piece-wise linear electrical circuit simulation for simulink," in *Proc. IEEE Int. Conf. Power Electron. and Drive Syst.*, vol. 1, 1999, pp. 355–360 vol.1.

[124] R. W. DeDoncker, M. H. Kheraluwala, and D. M. Divan, "Power conversion apparatus for dc/dc conversion using dual active bridges," U.S. patent US5 027 264A, 1991.

[125] M. Kheraluwala, R. Gascoigne, D. Divan, and E. Baumann, "Performance characterization of a high-power dual active bridge dc-to-dc converter," *IEEE Trans. Ind. Appl.*, vol. 28, no. 6, pp. 1294–1301, Nov 1992.

[126] F. Krismer and J. Kolar, "Accurate power loss model derivation of a high-current dual active bridge converter for an automotive application," *IEEE Trans. Ind. Electron.*, vol. 57, no. 3, pp. 881–891, March 2010.

[127] H. Qin and J. W. Kimball, "Generalized average modeling of dual active bridge dc-dc converter," *IEEE Trans. Power Electron.*, vol. 27, no. 4, pp. 2078–2084, April 2012.

[128] D. Costinett, D. Maksimovic, and R. Zane, "Design and control for high efficiency in high step-down dual active bridge converters operating at high switching frequency," *IEEE Trans. Power Electron.*, vol. 28, no. 8, pp. 3931–3940, Aug 2013.

[129] A. Waterland, *Stress POSIX workload generator*. [Online]. Available: http://people.seas.harvard.edu/~apw/stress

[130] *VICOR PFM4414xB6M48D0yAz Datasheet*, 1st ed., VICOR, Aug. 2018. [Online]. Available: http://www.vicorpower.com/documents/datasheets/ds_PFM4414xB6M48D0yAz.pdf

[131] *VICOR DCM3714xD2H53E0yzz Datasheet*, 1st ed., VICOR, July 2018. [Online]. Available: http://www.vicorpower.com/documents/datasheets/DCM3714xD2H53E0yzz_ds.pdf

[132] "80 plus certification," 2018. [Online]. Available: https://www.plugloadsolutions.com/80PlusPowerSupplies.aspx

[133] O. Garcia, J. A. Cobos, R. Prieto, P. Alou, and J. Uceda, "Single phase power factor correction: a survey," *IEEE Trans. Power Electron.*, vol. 18, no. 3, pp. 749–755, May 2003.

[134] H. Endo, T. Yamashita, and T. Sugiura, "A high-power-factor buck converter," in *Proc. 23rd Annu. IEEE Power Electron. Specialists Conf.*, June 1992, pp. 1071–1076 vol.2.

[135] Y.-W. Lo and R. J. King, "High performance ripple feedback for the buck unity-power-factor rectifier," *IEEE Trans. Power Electron.*, vol. 10, no. 2, pp. 158–163, Mar 1995.

[136] G. Spiazzi, "Analysis of buck converters used as power factor preregulators," in *Proc. Conf. Rec. 28th Annu. IEEE Power Electron. Specialists Conf.*, vol. 1, Jun 1997, pp. 564–570 vol.1.

[137] L. Huber, L. Gang, and M. M. Jovanovic, "Design-oriented analysis and performance evaluation of buck PFC front end," *IEEE Trans. Power Electron.*, vol. 25, no. 1, pp. 85–94, Jan 2010.

[138] V. Vlatkovic, D. Borojevic, and F. C. Lee, "Input filter design for power factor correction circuits," *IEEE Trans. Power Electron.*, vol. 11, no. 1, pp. 199–205, Jan 1996.

[139] M. C. Ghanem, K. Al-Haddad, and G. Roy, "A new single phase buck-boost converter with unity power factor," in *Proc. Conf. Rec. IEEE Ind. Appl. Conf. Twenty-Eighth IAS Annu. Meeting*, Oct 1993, pp. 785–792 vol.2.

[140] G. K. Andersen and F. Blaabjerg, "Current programmed control of a single-phase two-switch buck-boost power factor correction circuit," *IEEE Trans. Ind. Electron.*, vol. 53, no. 1, pp. 263–271, Feb 2005.

[141] M. Chen and J. Sun, "Feedforward current control of boost single-phase PFC converters," *IEEE Trans. Power Electron.*, vol. 21, no. 2, pp. 338–345, March 2006.

[142] F. Zhang, J. Xu, P. Yang, and Z. Chen, "Single-phase two-switch PCCM buck-boost PFC converter with fast dynamic response for universal input voltage," in *Proc. 8th Int. Conf. Power Electron.*, May 2011, pp. 205–209.

[143] M. O. Badawy, Y. Sozer, and J. A. D. Abreu-Garcia, "A novel control for a cascaded buck-boost PFC converter operating in discontinuous capacitor voltage mode," *IEEE Trans. Ind. Electron.*, vol. 63, no. 7, pp. 4198–4210, July 2016.

[144] T. A. Meynard and H. Foch, "Multi-level conversion: high voltage choppers and voltage-source inverters," in *Proc. 23rd Annu. IEEE Power Electron. Specialists Conf.*, Jun 1992, pp. 397–403 vol.1.

[145] D. Reusch, F. C. Lee, and M. Xu, "Three level buck converter with control and soft startup," in *Proc. IEEE Energy Conversion Congr. and Expo.*, Sept 2009, pp. 31–35.

[146] K. Kesarwani and J. T. Stauth, "Resonant and multi-mode operation of flying capacitor multi-level dc-dc converters," in *Proc. IEEE 16th Workshop Control and Modeling Power Electron.*, July 2015, pp. 1–8.

[147] V. Yousefzadeh, E. Alarcon, and D. Maksimovic, "Three-level buck converter for envelope tracking applications," *IEEE Trans. Power Electron.*, vol. 21, no. 2, pp. 549–552, March 2006.

[148] C. A. Teixeira, D. G. Holmes, and B. P. McGrath, "Single-phase semi-bridge five-level flying-capacitor rectifier," *IEEE Trans. Ind. Appl.*, vol. 49, no. 5, pp. 2158–2166, Sept 2013.

[149] A. Parastar, A. Gandomkar, and J. Seok, "High-efficiency multilevel flying-capacitor dc/dc converter for distributed renewable energy systems," *IEEE Trans. Ind. Electron.*, vol. 62, no. 12, pp. 7620–7630, Dec 2015.

[150] 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 *Proc. 32nd Annu. IEEE Appl. Power Electron. Conf. and Expo.*, March 2017, pp. 1090–1096.

[151] D. Chou, Y. Lei, and R. C. N. Pilawa-Podgurski, "A zero-voltage switching, physically flexible multilevel GaN dc-dc converter," in *Proc. IEEE Energy Conversion Congr. and Expo.*, Oct 2017, pp. 3433–3439.

[152] 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 Trans. Power Electron.*, pp. 1–1, 2018.

[153] A. Stillwell, M. E. Blackwell, and R. C. N. Pilawa-Podgurski, "Design of a 1 kV bidirectional dc-dc converter with 650 V GaN transistors," in *Proc. 33rd Annu. IEEE Appl. Power Electron. Conf. and Expo.*, March 2018, pp. 1155–1162.

[154] 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 *Proc. 32nd Annu. IEEE Appl. Power Electron. Conf. and Expo.*, March 2017, pp. 2582–2589.

[155] 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 seven-level flying capacitor multilevel inverter with an active energy buffer," *IEEE Trans. Power Electron.*, vol. 32, no. 11, pp. 8570–8581, Nov 2017.

[156] C. B. Barth, T. Foulkes, W. H. Chung, T. Modeer, P. Assem, P. Assem, Y. Lei, and R. C. N. Pilawa-Podgurski, "Design and control of a GaN-based, 13-level, flying capacitor multilevel inverter," in *Proc. IEEE 17th Workshop Control and Modeling Power Electron.*, June 2016, pp. 1–6.

[157] 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 *Proc. 33rd Annu. IEEE Appl. Power Electron. Conf. and Expo.*, March 2018, pp. 1656–1661.

[158] I. Moon, C. F. Haken, E. K. Saathoff, E. Bian, Y. Lei, S. Qin, D. Chou, S. Sedig, W. H. Chung, and R. C. N. Pilawa-Podgurski, "Design and implementation of a 1.3 kW, 7-level flying capacitor multilevel ac-dc converter with power factor correction," in *Proc. 32nd Annu. IEEE Appl. Power Electron. Conf. and Expo.*, March 2017, pp. 67–73.

[159] S. Qin, Z. Liao, Z. Ye, D. Chou, N. Brooks, and R. C. N. Pilawa-Podgurski, "A 99.1% efficient, 490 W/in3 power density power factor correction front end based on a 7-level flying capacitor multilevel converter," in *Proc. 33rd Annu. IEEE Appl. Power Electron. Conf. and Expo.*, March 2018, pp. 729–736.

[160] B. P. McGrath and D. G. Holmes, "Multicarrier PWM strategies for multilevel inverters," *IEEE Trans. Ind. Electron.*, vol. 49, no. 4, pp. 858–867, Aug 2002.

[161] R. H. Wilkinson, T. A. Meynard, and H. du Toit Mouton, “Natural balance of multicell converters: The general case,” *IEEE Trans. Power Electron.*, vol. 21, no. 6, pp. 1658–1666, Nov 2006.

[162] Y. Lei, “High-performance power converters leveraging capacitor-based energy transfer,” Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2017.

[163] A. Ruderman, B. Reznikov, and M. Margaliot, “Simple analysis of a flying capacitor converter voltage balance dynamics for dc modulation,” in *Proc. 13th Int. Power Electron. and Motion Control Conf.*, Sept 2008, pp. 260–267.

[164] B. Reznikov and A. Ruderman, “Four-level single-leg flying capacitor converter voltage balance dynamics analysis,” in *Proc. 13th European Conf. Power Electron. and Appl.*, Sept 2009, pp. 1–10.

[165] A. Ruderman and B. Reznikov, “Five-level single-leg flying capacitor converter voltage balance dynamics analysis,” in *Proc. 35th Annu. Conf. IEEE Ind. Electron. Soc.*, Nov 2009, pp. 486–491.

[166] B. Reznikov and A. Ruderman, “Six-level single-leg flying capacitor converter voltage balancing dynamics analysis,” in *Proc. 14th Int. Power Electron. and Motion Control Conf.*, Sept 2010, pp. 7–14.

[167] A. Shukla, A. Ghosh, and A. Joshi, “Capacitor voltage balancing schemes in flying capacitor multilevel inverters,” in *Proc. IEEE 38th Annu. Power Electron. Specialists Conf.*, June 2007, pp. 2367–2372.

[168] M. Khazraei, H. Sepahvand, K. Corzine, and M. Ferdowsi, “A generalized capacitor voltage balancing scheme for flying capacitor multilevel converters,” in *Proc. 25th Annu. IEEE Appl. Power Electron. Conf. and Expo.*, Feb 2010, pp. 58–62.

[169] A. Stillwell, E. Candan, and R. C. N. Pilawa-Podgurski, “Constant effective duty cycle control for flying capacitor balancing in flying capacitor multi-level converters,” in *Proc. IEEE 19th Workshop Control and Modeling Power Electron.*, July 2018, pp. 1–6.

[170] M. E. Blackwell, A. Stillwell, and R. C. Pilawa-Podgurski, “Dynamic level selection for full range ZVS in flying capacitor multi-level converters,” in *Proc. IEEE 19th Workshop Control and Modeling Power Electron.*, July 2018, pp. 1–6.

[171] 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 *Proc. IEEE 18th Workshop Control and Modeling Power Electron.*, July 2017, pp. 1–7.

[172] 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 *Proc. 32nd Annu. IEEE Appl. Power Electron. Conf. and Expo.*, March 2017, pp. 2925–2931.

[173] N. Pallo, T. Modeer, and R. C. N. Pilawa-Podgurski, “Electrically thin approach to switching cell design for flying capacitor multilevel converters,” in *Proc. IEEE 5th Workshop Wide Bandgap Power Devices and Appl.*, Oct 2017, pp. 411–416.

[174] S. Pirog and R. Stala, "Selection of parameters for balancing circuit of dc-dc and ac-ac multicell converters," in *Proc. European Conf. Power Electron. and Appl.*, Sept 2005, pp. 1–10.

[175] R. Stala, S. Pirog, M. Baszynski, A. Mondzik, A. Penczek, J. Czekonski, and S. Gasiorek, "Results of investigation of multicell converters with balancing circuit – Part I," *IEEE Trans. Ind. Electron.*, vol. 56, no. 7, pp. 2610–2619, July 2009.

[176] R. Stala, S. Pirog, A. Mondzik, M. Baszynski, A. Penczek, J. Czekonski, and S. Gasiorek, "Results of investigation of multicell converters with balancing circuit – Part II," *IEEE Trans. Ind. Electron.*, vol. 56, no. 7, pp. 2620–2628, July 2009.

[177] Y. S. Lee, S. J. Wang, and S. Y. R. Hui, "Modeling, analysis, and application of buck converters in discontinuous-input-voltage mode operation," *IEEE Trans. Power Electron.*, vol. 12, no. 2, pp. 350–360, Mar 1997.

[178] V. Grigore and J. Kyyra, "High power factor rectifier based on buck converter operating in discontinuous capacitor voltage mode," *IEEE Trans. Power Electron.*, vol. 15, no. 6, pp. 1241–1249, Nov 2000.

[179] L. Solero, V. Serrao, M. Montuoro, and A. Romanelli, "Low THD variable load buck PFC converter," in *Proc. 29th IEEE Annu. Power Electron. Specialists Conf.*, June 2008, pp. 906–912.

[180] J. Cardesin, J. Sebastian, P. Villegas, A. Hernando, and A. Fernandez, "Average small-signal model of buck converter used as power factor preregulator," in *Proc. Conf. Rec. IEEE Ind. Appl. Conf. Thirty-Seventh IAS Annu. Meeting*, vol. 3, Oct 2002, pp. 2147–2151 vol.3.

[181] X. Wu, J. Yang, J. Zhang, and M. Xu, "Design considerations of soft-switched buck PFC converter with constant on-time (COT) control," *IEEE Trans. Power Electron.*, vol. 26, no. 11, pp. 3144–3152, Nov 2011.

[182] X. Wu, J. Yang, J. Zhang, and Z. Qian, "Variable on-time (VOT)-controlled critical conduction mode buck PFC converter for high-input ac/dc HB-LED lighting applications," *IEEE Trans. Power Electron.*, vol. 27, no. 11, pp. 4530–4539, Nov 2012.

[183] M. C. Ghanem, K. Al-Haddad, and G. Roy, "A new control strategy to achieve sinusoidal line current in a cascade buck-boost converter," *IEEE Trans. Ind. Electron.*, vol. 43, no. 3, pp. 441–449, Jun 1996.

[184] R. Ramabhadran, Y. Levy, B. Roberts, and V. Pradeep, "Low THD multipliers for BCM buck and cascaded buck-boost PFC converters," in *Proc. IEEE Energy Conversion Congr. and Expo.*, Oct 2017, pp. 5293–5297.

[185] S. Qin, "Toward high-efficiency high power density single-phase dc-ac and ac-dc power conversion - architecture, topology and control," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2017.

[186] L. Corradini, D. Maksimovic, P. Mattavelli, and R. Zane, *Digital Control of High-Frequency Switched-Mode Power Converters.* Wiley-IEEE Press, 2015.

[187] 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 *Proc. 33rd Annu. IEEE Appl. Power Electron. Conf. and Expo.*, March 2018, pp. 2717–2724.

[188] B. P. McGrath and D. G. Holmes, "Analytical modelling of voltage balance dynamics for a flying capacitor multilevel converter," *IEEE Trans. Power Electron.*, vol. 23, no. 2, pp. 543–550, March 2008.

[189] J. Sun, "On the zero-crossing distortion in single-phase PFC converters," *IEEE Trans. Power Electron.*, vol. 19, no. 3, pp. 685–692, May 2004.