Closed Loop Control for a Piezoelectric-Resonator-Based DC-DC Power Converter

by

Joshua J. Piel

S.B., Electrical Engineering and Computer Science
Massachusetts Institute of Technology, 2022

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2022

Author .....
Department of Electrical Engineering and Computer Science
January 21, 2022

Certified by .....
David J. Perreault
Joseph F. and Nancy P. Keithley Professor of Electrical Engineering
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Closed Loop Control for a Piezoelectric-Resonator-Based DC-DC Power Converter

by

Joshua J. Piel

Submitted to the Department of Electrical Engineering and Computer Science
on January 21, 2022, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Miniaturization of power electronics reduces their cost and increases their scope of potential applications. Power electronics traditionally rely on magnetics for energy storage, but magnetics are fundamentally less efficient and power dense when scaled to small sizes. Piezoelectric resonators (PRs), which store energy in mechanical inertia and compliance, are promising alternatives to magnetic energy storage for miniaturized power electronics because of their high quality factors and favorable scaling properties. Dc-dc converters relying on only a PR for energy storage have been demonstrated to achieve high efficiency through specific behaviors including PR soft charging, ZVS of all active switches, and all-positive instantaneous power transfer. However, closed-loop control of PR-based dc-dc converters is necessary for them to be practically viable. Implementation of this closed loop control is challenging because achieving all desired high-efficiency behaviors requires simultaneous control of duty cycle, dead time, and frequency.

This thesis presents a closed-loop control scheme for PR-based dc-dc power converters that are implemented with six-stage switching sequences and two-half-bridge topologies. The voltage regulation range of a PR-based converter can be derived from its operating modes, referred to as switching sequences. The regulation range is then used to conceptualize each half-bridge in the converter topology as regulating or nonregulating. Control methods for the regulating and nonregulating half-bridges capable of achieving all desired high-efficiency behaviors are proposed.

This thesis also presents several methods for modeling the operation of PR-based dc-dc converters, both in periodic steady state (PSS) and in dynamic operation. PSS solutions are obtained using conservation equations associated with the switching sequence, including strategies for both ideal solutions and solutions considering the mechanical loss of the PR. Several methods for modeling converter dynamics are proposed, including a linearizable state space model.

Finally, this thesis designs and implements an example PR-based dc-dc converter and a microcontroller-based closed-loop controller. The converter is operated at 30 V to 10 V with a 0.5 W output power. The controller was verified to meet all of

the desired high efficiency behaviors, and its transient response characteristics are evaluated.

Thesis Supervisor: David J. Perreault
Title: Joseph F. and Nancy P. Keithley Professor of Electrical Engineering

Acknowledgments

I would like to thank Jessica Boles for being an incredible mentor to me throughout my undergraduate and graduate studies at MIT. She has pushed me to strive for excellence with my research and I would not be where I am today without her mentorship and guidance. I also want to thank Dave Perreault for being a great teacher, advisor, and mentor.

I also want to thank my friends from MacGregor, Roboteam, the Power Electronics lab, and my fellow AFROTC cadets for their support throughout my time at MIT. I also want to thank my parents and my friends back in Virginia for helping me get though virtual MIT.

I finally would like to thank my COVID-19 infection that delayed the submission of my thesis by a week.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Reducing the size of power converters can make them more cost-effective and useful to a wider range of applications. However, the use of magnetic energy storage is a major obstacle to miniaturization. Traditional dc-dc power converters utilize magnetics for energy storage, but magnetics have fundamentally lower efficiency and power density capabilities when scaled to small sizes [20]. Switched capacitor converters are capable of high power densities but still require magnetics to achieve voltage regulation [13, 18, 14, 9]. Piezoelectric resonators (PRs), which store energy in mechanical compliance and inertia, offer a promising alternative to magnetics for miniaturized power conversion. Unlike magnetics, piezoelectrics have favorable efficiency and power density characteristics at small scales [12, 2, 6]. Piezoelectrics also offer planar form factors, ease of batch fabrication, and potential for integration.

The capabilities of piezoelectric materials and piezoelectric-based power conversion have been heavily in [4, 3, 2, 5] and the references therein. [4] enumerates all possible PR-based dc-dc power converters operating modes, called switching sequences, that meet certain high efficiency behaviors and practical constraints, including output regulation, PR soft charging, and ZVS. An experimental prototype achieving greater than 99% efficiency is demonstrated. [4] is integral to the developments in this thesis. [3] explores the use of piezoelectric transformers (PTs) in dc-dc converters. PTs are two-port counterparts of PRs, and they are capable of providing both voltage transformation and galvanic isolation. Finally, [2, 5, 6] explore in depth how the material

properties and geometries of PRs can be optimized to give high performance in power electronics.

Other recent work has also successfully demonstrated PR-based dc-dc converters [16, 6, 21] that achieve high efficiencies over a wide range of output voltages. However, there has been relatively little investigation into closed-loop control strategies for PR-based converters. [19] implements pulse frequency modulation, but without ZVS. The control schemes of [16, 22] achieve ZVS, but their reliance on sensing may be challenging to scale to high frequencies. Control of dc-dc power converters based on PTs has been studied more thoroughly. ZVS operation in magnetics-less PT-based converters has been analyzed in [3, 17, 8, 11], and multiple control strategies have been proposed [1, 24, 7]. These implementations are effective in achieving ZVS and quickly responding to transients, but they also involve complex sensing and waveform reconstruction techniques.

Implementing closed-loop control that achieves the desired high efficiency behaviors is challenging because it requires elements of frequency modulation, pulse width modulation, dead time control, and phase shift control between half-bridges. This thesis presents a closed-loop control strategy for PR-based dc-dc converters based on six-stage switching sequences and topologies with two half-bridges [4]. The highest-efficiency switching sequence proposed in [4] is used as an example to demonstrate the proposed control. This strategy conceptualizes each half-bridge as either "regulating" or "non-regulating", each of which serve different roles in maintaining the switching sequence. The proposed scheme maintains the precise switch timing needed to achieve the desired high-efficiency behaviors, which we validate in an experimental prototype. It also has potential for scaling to high frequencies.

Chapter 2 summarizes the relevant information from [4] to develop switching sequences and operating ranges for PR-based dc-dc converters. Chapter 3 develops the methods used to solve for the exact steady state behavior of a switching sequence, including when PR loss is considered.

Chapter 4 proposes the main control scheme developed in this thesis. It derives the regulating and nonregulating half-bridges, then presents two control strategies for

regulating half-bridges and one for non-regulating half-bridges. Finally, the specific requirements for the example switching sequence are given. Chapter 5 then proposes multiple methods for modeling the dynamics of the proposed control strategies.

Chapter 6 presents the design for a prototype PR-based dc-dc converter that realizes the example switching sequence. Chapter 7 then details the full implementation of the feedback controller on a microcontroller including the gate signal generation, sensing circuitry, and feedback loop computations. Finally, Chapter 8 validates the controller experimentally.

Chapter 2

Piezoelectric Resonators and Converter Switching Sequences

This chapter covers the basics of piezoelectric resonators and introduces switching sequences, which are used to describe the operation of PR-based dc-dc converters.

2.1 What is a Piezoelectric Resonator?

Piezoelectric resonators (PRs) are two-terminal devices that couple electrical and mechanical states and store energy in mechanical inertia and compliance. The piezoelectric and inverse piezoelectric effects relate the electric displacement and voltage of the device to mechanical stress and strain. A common material used in PRs is Lead Zirconium Titanate (PZT), though other materials such as Lithium Niobiate are also being investigated for use in power electronics [2, 6]. PRs can be manufactured in different shapes and can resonate in different vibration modes. Figure 2-1 shows an image of several different PRs, and further details about the material properties of PRs can be found in [2].

An equivalent circuit representation of PRs is given by the Butterworth Van-Dyke model. As illustrated in Figure 2-2, the equivalent circuit is a capacitor in parallel with a "motional" series RLC branch. The capacitor C_p, also called the static capacitance, represents the portion of the device's physical capacitance that does

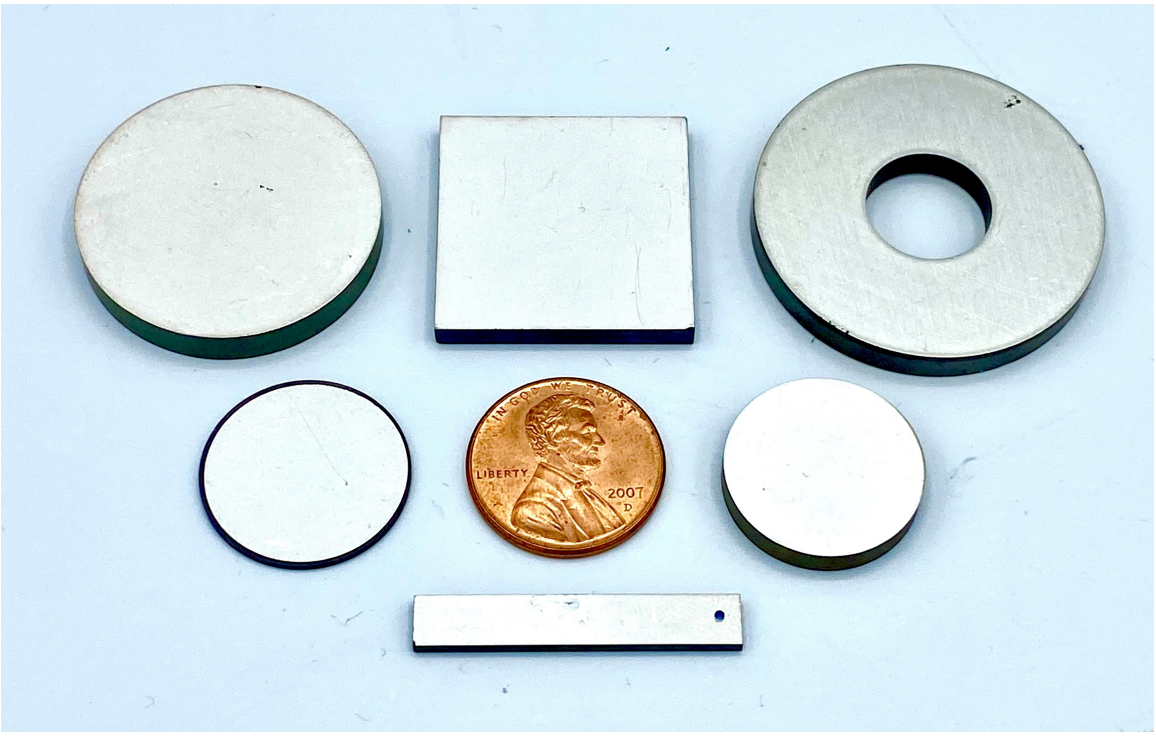


Figure 2-1: Picture of several commercially available piezoelectric resonators.

not couple with the mechanical domain. The RLC branch models the mechanical resonance properties of the device. Energy stored in the inductor L is analogous to energy stored in mechanical inertia, energy stored in the capacitor C_r is analogous to energy stored in mechanical compliance, and the resistor models mechanical loss to the first order. Two important quantities that will be referred to throughout this thesis are v_p, the voltage across C_p and the PR terminals, and i_L, the analogous current flowing through the motional inductor. The PR has two relevant resonant frequencies: the series resonant frequency and the parallel resonant frequency. The series resonant frequency f_{series}, as given in Equation 2.1, is the resonant frequency of just the motional RLC branch:

$$f_{series} = \frac{1}{2\pi\sqrt{LC_r}} \tag{2.1}$$

The parallel resonant frequency $f_{parallel}$, as given in Equation 2.2, is the frequency at which the net reactance of the motional RLC branch resonates together with the

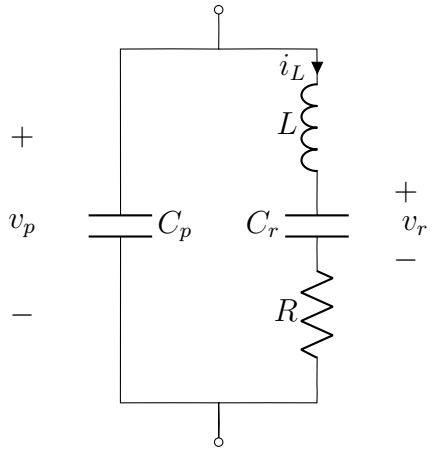


Figure 2-2: Butterworth-Van Dyke circuit model for PRs [23].

parallel capacitor:

$$f_{parallel} = \frac{1}{2\pi\sqrt{L\frac{C_p C_r}{C_p + C_r}}} \tag{2.2}$$

2.2 Switching Sequences

Switching sequences describe the steady state operating modes of PR based dc-dc converters. A switching sequence is a temporal sequence of different "stages", where each stage consists of a different way the PR's terminals are connected (or not) within the converter. There are two main types of stages, connected stages and open stages. Connected stages are stages where both terminals of the PR are is connected to the source load system in one of several ways. As illustrated in Figure 2-3, the possibilities are $\pm V_{out}$, $\pm V_{in}$, $\pm\left(V_{in} - V_{out}\right)$, and $Zero$. Connected stages allow energy transfer between the PR and the source-load system. A $Zero$ stage is a special case where the PR terminals are shorted together. Open stages are stages where the PR has one or both nodes open circuited, and the motional branch resonates with C_p. Open stages are useful because they allow the PR to internally change v_p through resonance.

[4] enumerates all switching sequences that meet the following high-efficiency behaviors and design constraints:

- PR Soft Charging - The PR only begins a connected stage when v_p is equal to

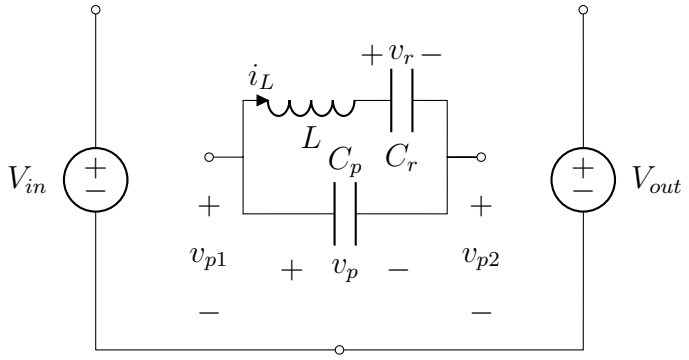


Figure 2-3: Common-negative system considered for switching sequence enumeration, illustrated with an ideal PR.

the voltage of the connected stage. This requires that there must be an open stage between connected stages.

- Zero Voltage Switching (ZVS) - Switches only turn on when their drain-source voltage is zero.

- All-Positive Instantaneous Power Transfer - The converter never sends energy back into the source or retrieves energy from the load.

- Output Voltage Regulation - There is a continuous range of possible output voltages that the converter can efficiently provide.

- Minimal number of stages - This produces the simplest switching sequences, which is important for minimizing control requirements.

- Minimal active switches - This produces the simplest topological implementation, which is important for practical considerations such as cost.

As given in [4], the simplest switching sequences that achieve all of the desired behaviors are six-stage switching sequences. There are eight suitable six-stage switching sequences, and nine possible converter topologies, each using four unidirectional-voltage-blocking switches. There are two classes of topologies, 2+2 topologies and 3+1 topologies. 2+2 topologies have two half-bridges, with both PR nodes driven by a half-bridge. 3+1 topologies have one PR node permanently fixed to the source load

system, while the other node can be connected in three ways. 2+2 topologies support multiple possible switching sequences, while 3+1 topologies only support one.

Additionally, constraints must be placed on the zero-crossings of i_L to maintain the desired high-efficiency behaviors within a given switching sequence. All-positive instantaneous power transfer constrains the i_L polarity during connected stages. However, since zero stages do not transfer power to the source or load, i_L could potentially have either polarity. Open stages need to either increase or decrease V_p, so they also require a specific average i_L polarity to charge or discharge C_p. To minimize circulating currents in the PR, we constrain all stages to have only unidirectional current, which forces i_L zero crossings to occur on stage transitions. However, an exception is made to achieve ZVS on 2+2 circuit topologies, where ZVS requires one of the open stages to be split into two parts, with the zero crossing happening between the two parts. Some switching sequences support multiple i_L zero crossing constraints, which give different output voltage regulation ranges. This is further described in Section 2.3, and more information about i_L zero crossing constraints can be found in [4].

To develop closed-loop control for PR-based converters in this thesis, we focus on the 6-stage sequence V_{in}-V_{out}, $Zero$, V_{out}, however the control concepts developed will apply to all switching sequences that can be implemented on topologies using two half bridges. The switching sequence is named based on its connected stage voltages in the order they occur. All stages are numbered, starting with the first connected stage in the switching sequence. Thus, connected stages are labelled with odd numbers, and open stages with even numbers. V_{in}-V_{out}, $Zero$, V_{out} can be realized with the topology shown in Fig. 2-4. Fig. 2-5 shows a time-domain plot of the switching sequence, and the following list describes its operation during each of the six stages:

1. The PR is connected in series between the input and load, and $v_p = V_{in} - V_{out}$. S1 and S3 are on. i_L is positive.

2. The PR is open circuited and v_p resonates from $V_{in} - V_{out}$ to 0. S3 is on. i_L is positive.

3. The PR is short circuited, allowing for energy redistribution. S2 and S3 are on.

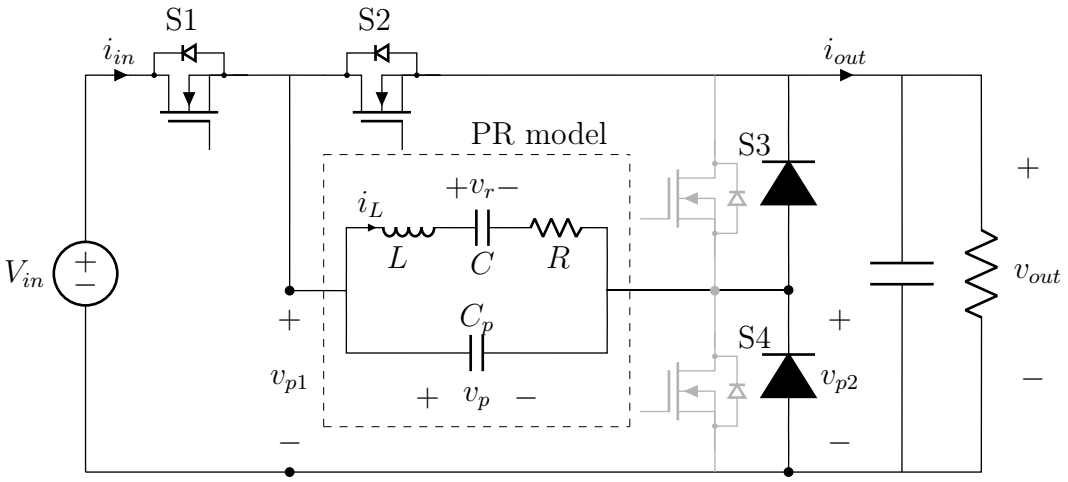


Figure 2-4: PR converter topology for switching sequence $V_{in} - V_{out}$, $Zero$, V_{out}. The PR equivalent circuit as given in [23] is within the dotted lines. Switches are labeled $S1 - S4$ and the PR terminals are labeled v_{p1} and v_{p2}. v_p is the PR voltage and i_L is the current through the PR series inductor.

i_L is entirely positive or entirely negative.

4. The PR is open circuited, and v_p resonates from 0 to V_{out}. S2 is on. i_L is negative.

5. The PR is connected to the load, and $v_p = V_{out}$. S2 and S4 are on. i_L is negative.

6A) The PR is open circuited and resonates from V_{out} to V_{in}, allowing for ZVS of S1. S4 is on. i_L is negative.

6B) The PR remains open circuited, and resonates from V_{in} to $V_{in} - V_{out}$. S1 is on. i_L is positive.

Stage 6, an open stage, is split into two halves (designated 6A and 6B) divided by an i_L zero crossing. v_p resonates to V_{in} at this point to allow ZVS of S1. The second i_L zero crossing occurs either between stages 2 and 3 or stages 3 and 4 to ensure unidirectional current within stages, minimizing loss due to circulating current. When the i_L zero crossing occurs between stages 3 and 4, then S3 and S4 both act as diodes, allowing them to be implemented passively as diodes if desired.

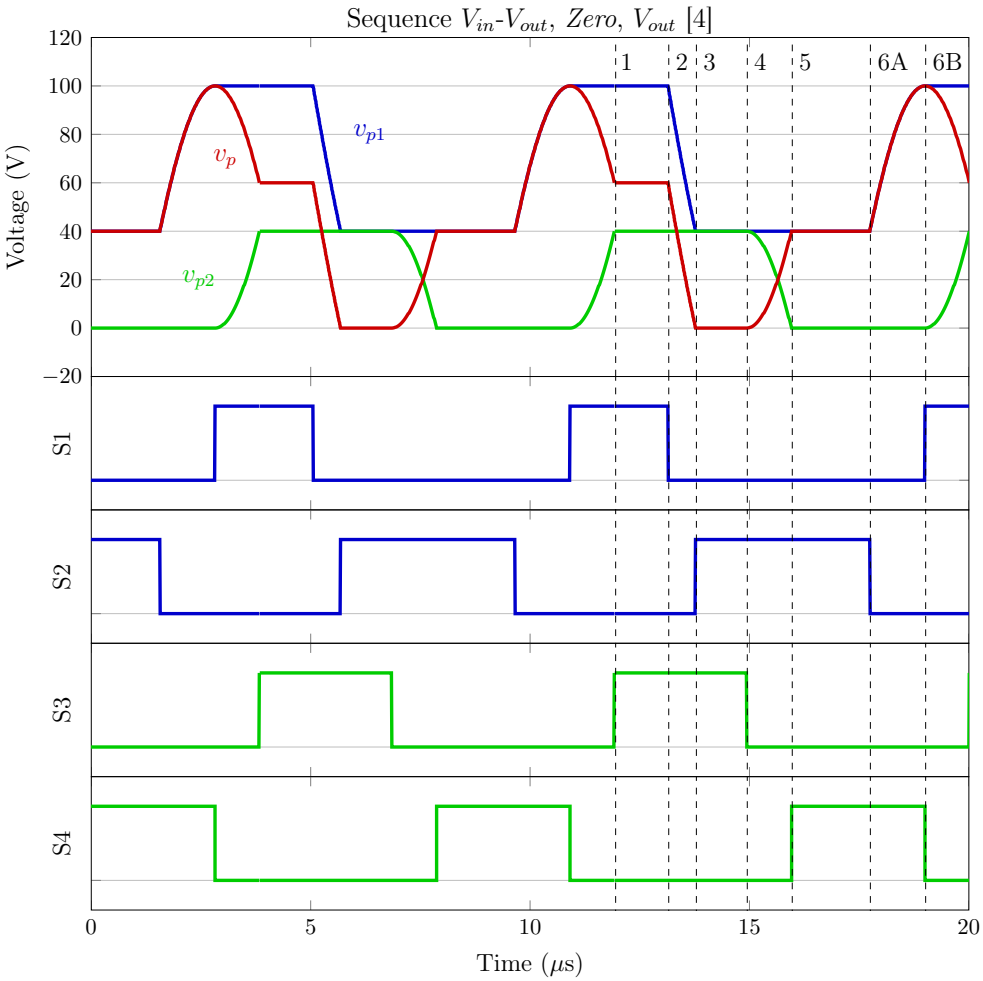


Figure 2-5: Simulation of $V_{in} - V_{out}$, $Zero$, V_{out} from [4] for $V_{in} = 100V$, $V_{out} = 40V$, and $P_{out} = 6W$. Numbers 1-6B designate connected/zero stages (odd) and open stages (even). The PR parameters used are $C_p = 4.3$nF, $C_r = 1.4$nF, $L = 1.4$mH, and $R = 2.4\Omega$.

2.3 Operating Ranges

Conservation of energy (CoE) and conservation of charge (CoC) are fundamental principles that must be met for periodic steady-state (PSS) operation of a PR in a converter, and are valuable when analyzing any power converter operating in PSS. By analyzing the energy and charge balance constraints on the PR in steady state, we can derive the operating range for a switching sequence that satisfies all desired behaviors (enumerated in Section 2.2), including those necessary for high efficiency. We refer to the charge transferred by i_L during stage n as q_n, so the charge transferred in stage 1 is q_1 (preserving the polarity of i_L). The sum of charges over all stages must equal 0 for charge balance on C_r:

$$q_1 + q_2 + q_3 + q_4 + q_5 + q_6 = 0 \tag{2.3}$$

Since v_p only changes during open stages, the sum of all open stage charges must also equal 0 for charge balance on C_p:

$$q_2 + q_4 + q_6 = 0 \tag{2.4}$$

Thus, combining Equations 2.3 and 2.4 requires that the connected stage charges must also balance:

$$q_1 + q_3 + q_5 = 0 \tag{2.5}$$

The sum of connected stage energy changes must also be 0 for energy balance over a cycle within the PR. For a general switching sequence with connected stage voltages V_1, V_3, and V_5, energy balance is given by:

$$V_1 q_1 + V_3 q_3 + V_5 q_5 = 0 \tag{2.6}$$

Combining Equations 2.5 and 2.6, and using the i_L polarity constraints to fix the signs of the charges results in a range of possible V_{out}. We will derive the regulation

range for the V_{in}-V_{out}, $Zero$, V_{out} sequence under both possible i_L zero crossing constraints. To intuitively represent the signs of the charge quantities, we will use their absolute values. Constraining the i_L zero crossings between stages 3 and 4 requires $q_1 > 0$, $q_3 > 0$ and $q_5 < 0$, giving:

$$|q_1| + |q_3| = |q_5| \tag{2.7}$$

Similarly, constraining the i_L zero crossings between stages 2 and 3 requires $q_1 > 0$, $q_3 < 0$ and $q_5 < 0$, giving:

$$|q_1| = |q_3| + |q_5| \tag{2.8}$$

The connected stage voltages are $V_{in} - V_{out}$, $Zero$, and V_{out}, so to balance PR energy:

$$(V_{in} - V_{out})|q_1| + (0)|q_3| = (V_{out})|q_5| \tag{2.9}$$

Rearranging gives:

$$\frac{V_{out}}{V_{in}} = \frac{1}{1 + \frac{|q_5|}{|q_1|}} \tag{2.10}$$

Now, we can manipulate the charge balance equations to determine the possible regulation range. When the i_L zero crossing occurs between stages 3 and 4, Equation 2.7 requires $|q_1| < |q_5|$ and thus:

$$1 < \frac{|q_5|}{|q_1|} < \infty \tag{2.11}$$

Plugging Equation 2.11 into Equation 2.10 then provides the regulation range of $V_{in} - V_{out}$, $Zero$, and V_{out} with positive q_3:

$$0 < \frac{V_{out}}{V_{in}} < \frac{1}{2} \tag{2.12}$$

In the other case, when the i_L zero crossing occurs between stages 2 and 3, Equation 2.8 requires $|q_1| > |q_5|$ and thus:

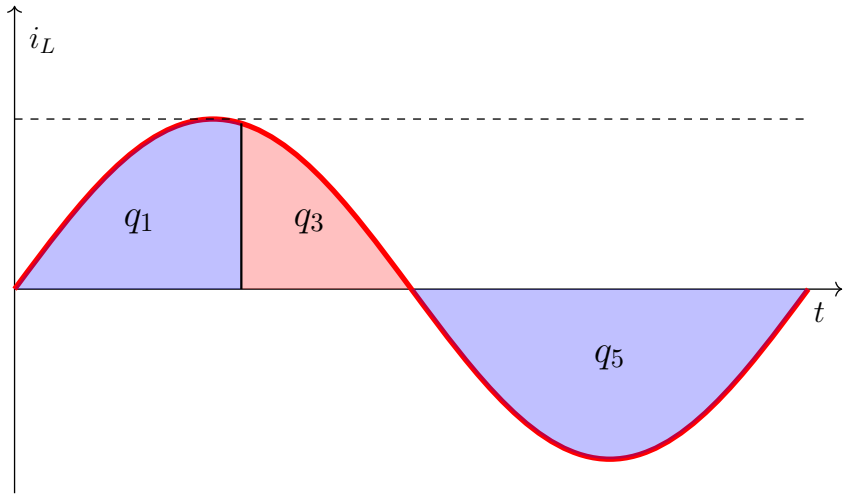


Figure 2-6: Plot of connected stage charge transfers for $V_{in} - V_{out}$, $Zero$, V_{out} with $V_{out} < \frac{1}{2}V_{in}$. The output voltage can be regulated by trading off between q_1 and q_3.

$$0 < \frac{|q_5|}{|q_1|} < 1 \tag{2.13}$$

Plugging Equation 2.13 into Equation 2.10 then provides the regulation range of $V_{in} - V_{out}$, $Zero$, and V_{out} with negative q_3:

$$\frac{1}{2} < \frac{V_{out}}{V_{in}} < 1 \tag{2.14}$$

Equations 2.12 and 2.14 show that $V_{in} - V_{out}$, $Zero$, and V_{out} is a step down switching sequence, with different i_L zero crossing constraints required depending on whether the gain is less than or greater than $\frac{1}{2}$. Figures 2-6 and 2-7 show how regulation can be achieved by modulating charge proportions in each case. In these figures, we assume the open stage charges are negligible compared to the connected stage charges.

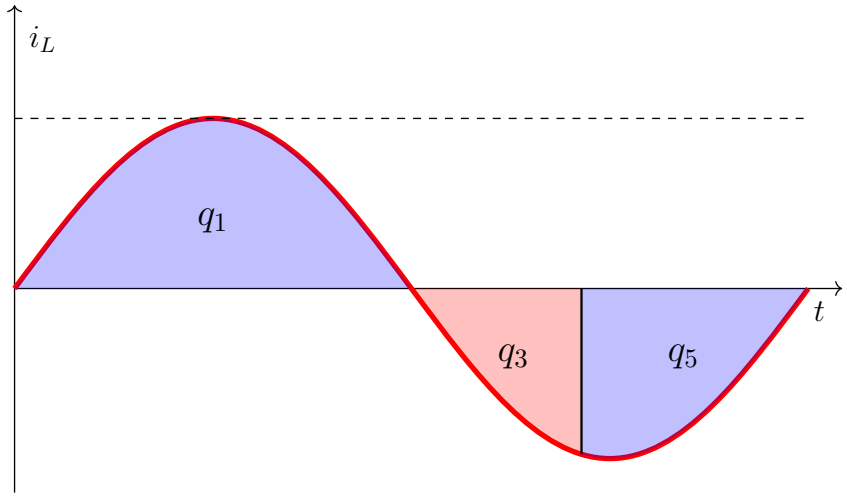


Figure 2-7: Plot of connected stage charge transfers for $V_{in} - V_{out}$, $Zero$, V_{out} with $\frac{1}{2}V_{in} < V_{out} < V_{in}$. The output voltage can be regulated by trading off between q_3 and q_5.

Chapter 3

PR Converter Periodic Steady State Solution

This chapter introduces the periodic steady state (PSS) analysis techniques that quantify exactly how a PR-based dc-dc converter operates. First, we use a state plane visualization to understand the forms that steady state solutions can take. Then, we present multiple methods to solve for PSS solutions, depending on whether PR mechanical losses are ignored or considered.

3.1 State Plane Visualization

The state plane is a useful tool for visualizing how the PR's internal states evolve over a switching sequence. The state plane plots one state variable against another, and state plane curves are formed by parameterizing the time domain waveforms. The state planes we will use to capture full PSS behavior are the i_L vs v_p and i_L vs v_r state planes. The i_L vs v_p state plane is most important for understanding switching sequence behavior. An example of a state plane can be found in Figure 3-1, and the corresponding time domain waveforms can be found in Figure 3-2. Here, each stage is represented by a specific straight or curved segment. Every stage's segment has an initial point and a final point, where a point refers to the set of state variables (v_p, v_r, i_L) at that point in time.

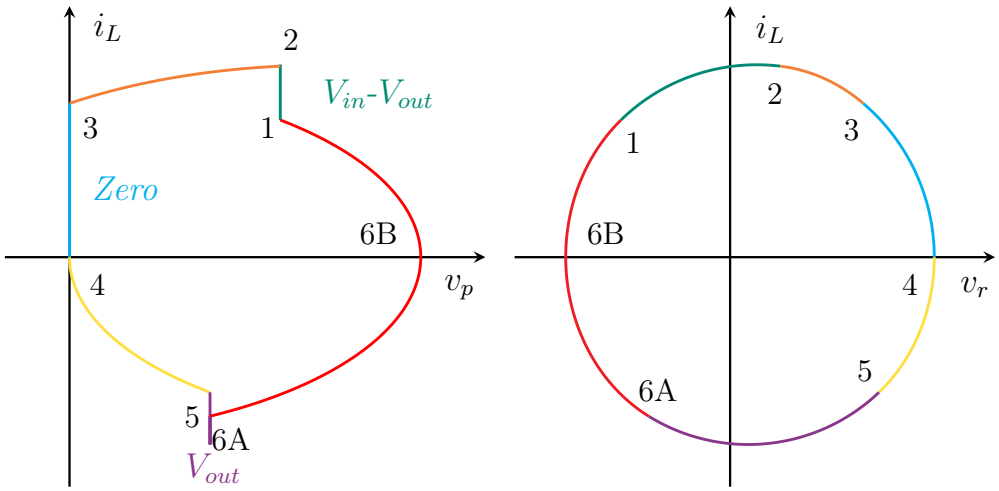


Figure 3-1: State plane example for soft-switched sequence V_{in}-V_{out}, *Zero*, V_{out} with V_{in} = 100 V, V_{out} = 40 V, and P_{out} = 6 W. Numbers 1-6B correspond to the time-domain points indicated in Fig. 3-2.

During connected stages, v_p is held constant at some combination of $\pm V_{in}$, $\pm V_{out}$, and 0, which we will denote capital V_p. L and C_r will resonate according to Figure 3-3a/b. On the i_L vs v_p state plane, connected stages are represented as vertical line segments since v_p is constant, and on the i_L vs v_r state plane, connected stages are represented with elliptical arcs (or circular arcs with appropriate normalizations), with a center of (V_p, 0).

During open stages, C_p, C_r, and L all resonate according to Figure 3-3c. We define the series combination of C_p and C_r as C_{eff}:

$$C_{eff} = \frac{C_p C_r}{C_p + C_r} \tag{3.1}$$

On both the v_p vs i_L and v_r vs i_L state planes, open stages are represented as elliptical (circular when normalized) arcs. The center point for these arcs on both state planes is found to be (V_o, 0), where V_o is the following:

$$V_o = \frac{C_p v_{p,i} + C_r v_{r,i}}{C_p + C_r} \tag{3.2}$$

$v_{p,i}$ and $v_{r,r}$ are the initial values of v_p and v_r for that open stage. More on this derivation can be found in [4].

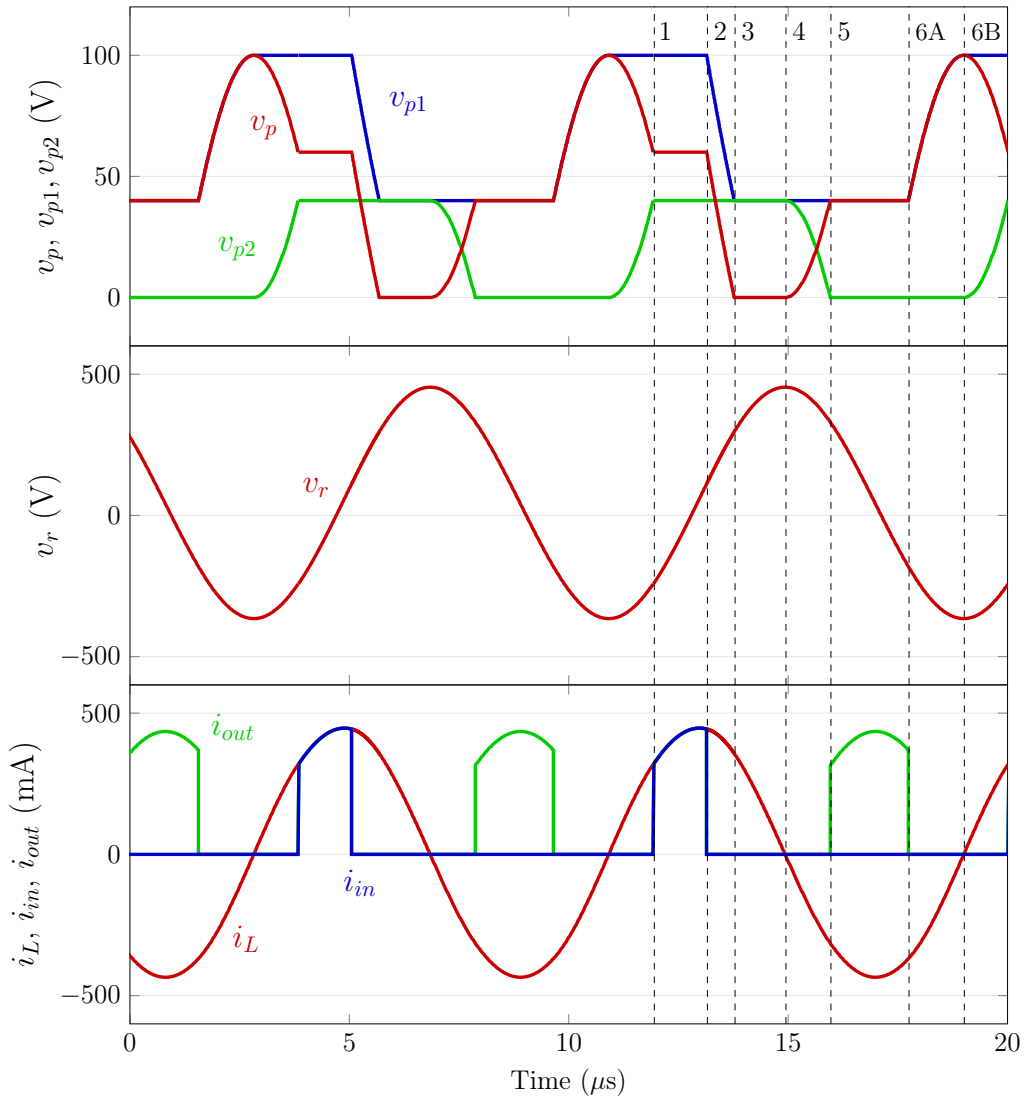


Figure 3-2: Time-domain waveforms for soft-switched sequence V_{in}-V_{out}, *Zero*, V_{out} with $V_{in} = 100$ V, $V_{out} = 40$ V, and $P_{out} = 6$ W. v_{p1} and v_{p2} refer to the switch nodes between S1, S2 and S3, S4, respectively, in Figure 2-4. Designations 1-6B correspond to the state transition points in Figure 3-1. $C_r = 1.4$nF, $L = 1.4$mH, and $R = 2.4\Omega$

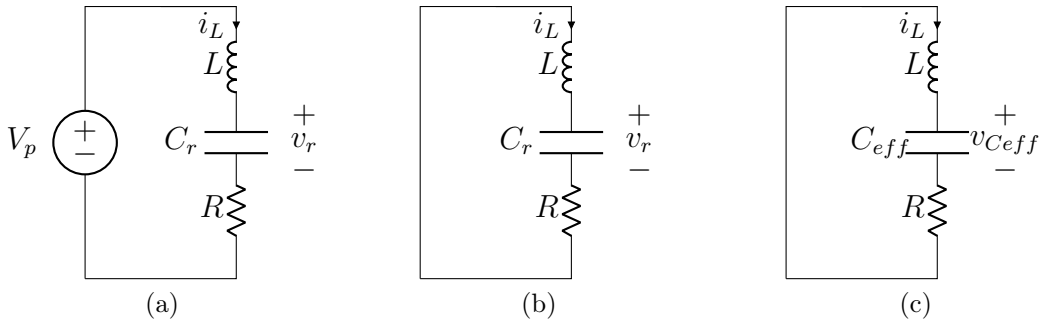


Figure 3-3: Resonant circuits for (a) connected stages, (b) zero stages, and (c) open stages.

Certain switching sequence behaviors can also be easily visualized with the state plane. If the final point of an open stage aligns with the initial point of the following connected stage, then PR soft charging is achieved. Additionally, if the final point of the last stage in the sequence is the initial point of the first stage, then PSS is achieved. Finally, i_L zero crossing constraints can be visualized through a stage's initial or final point lying on the $i_L = 0$ axis. We will also use the term "corner variable" to represent a stage's initial point. For example, the corner variables ($v_{p,2}$, $v_{r,2}$, $i_{L,2}$) refer to the initial variables of stage 2 (and equivalently the final variables of stage 1). Corner variables will be important for setting up and solving for these PSS solutions in later sections.

3.2 Ideal Steady State Solution

Quantifying corner variables and their exact locations on a state plane requires solving the six-stage system for periodic steady state. By ignoring mechanical losses in the PR (equivalently, letting $R = 0$), we can use the CoC and CoE equations that govern each stage of the switching sequence to solve for the corner variables. Each stage has a set of equations that relate its initial variables to its final variables. By equating the final variables of every stage with the initial variables of the following stage, we can create a system of equations that represent the switching sequence. We constrain for PSS by requiring that the final variables of the last stage equal the initial variables of the first stage.

Connected stages have the following CoE constraint, where v_p is fixed based on the PR's terminal connections, the initial variables are (V_p, $v_{r,i}$, $i_{L,i}$), and the final variables are (V_p, $v_{r,f}$, $i_{L,f}$):

$$C_r(v_{r,i} - V_p)^2 + Li_{L,i}^2 = C_r(v_{r,f} - V_p)^2 + Li_{L,f}^2 \tag{3.3}$$

We use a capital V in V_p to indicate this quantity is not a variable but fixed by the terminal connections, as defined by the switching sequence.

Open stages for the PR have both a CoE constraint and a CoC constraint since C_p

now participates in the resonance with the PR's other elements. Thus, they exhibit the following CoE and CoC constraints, where the initial variables are ($V_{p,i}$, $v_{r,i}$, $i_{L,i}$), and the final variables are ($V_{p,f}$, $v_{r,f}$, $i_{L,f}$):

$$C_p V_{p,i}^2 + C_r V_{r,i}^2 + L i_{L,i}^2 = C_p V_{p,f}^2 + C_r V_{r,f}^2 + L i_{L,f}^2 \tag{3.4}$$

$$C_p \left(V_{p,f} - V_{p,i}\right) = -C_r \left(V_{r,f} - V_{r,i}\right) \tag{3.5}$$

Each stage adds two free parameters from its corner variables, $v_{r,x}$ and $i_{L,x}$. Note that we assume the connected stage voltages are all known constants, meaning that V_{in} and V_{out} are fixed before the equations are solved. Additionally, at stage boundaries, v_p is always a known value based on the switching sequence when soft charging is achieved. Connected stages add one constraining equation, and open stages add two constraining equations. Therefore, a six stage sequence with three connected stages and three open stages uses 12 independent variables with 9 equations. However, each switching sequence also has two i_L zero crossing constraints at stage boundaries, increasing the number of equations to 11. Switching sequences with a two-part open stage can be equivalently represented by two consecutive open stages, and thus have 14 independent variables and 13 equations. Notably, there is one unconstrained variable that defines a family of viable PSS solutions; this variable adjusts the output current at a fixed V_{out}.

These equations can be solved using a numerical or analytic solver. Because the equations are purely quadratic, the numerical solutions can be quickly calculated and the MATLAB analytic solver is capable of producing general closed-form solutions. An example set of equations and their solutions for the $V_{in} - V_{out}$, $Zero$, V_{out} switching sequence with $V_{out} < \frac{1}{2} V_{in}$ can be found in Appendix A.1. Additionally, MATLAB code that solves these equations can be found in Appendix D.

3.2.1 Switching Time Calculations

The time spent during each stage can be computed from its initial and final variables. All stages form elliptical arcs on the state plane, and these arcs can be transformed into circular arcs by normalizing the coordinates. Trigonometry can then be used to compute the arc length, and finally the arc length can be scaled by the stage's angular frequency (open or connected) to get the time. The normalization is given by the stage's characteristic impedance. The characteristic impedance for connected stages is:

$$Z_{0,conn} = \sqrt{\frac{L}{C_r}} \tag{3.6}$$

The characteristic impedance for open stages is:

$$Z_{0,open} = \sqrt{\frac{L}{C_{eff}}} \tag{3.7}$$

The time spent in a connected stage with initial variables (V_p, $v_{r,i}$, $i_{L,i}$) and final variables (V_p, $v_{r,f}$, $i_{L,f}$) is:

$$t_{conn} = \sqrt{LC_r}\left(tan^{-1}\left(\frac{i_{L,f}\sqrt{L/C_r}}{V_{r,f}-V_p}\right) - tan^{-1}\left(\frac{i_{L,i}\sqrt{L/C_r}}{V_{r,i}-V_p}\right)\right) \tag{3.8}$$

Similarly, the time spent in an open stage with initial variables ($V_{p,i}$, $v_{r,i}$, $i_{L,i}$) and final variables ($V_{p,f}$, $v_{r,f}$, $i_{L,f}$) is:

$$t_{open} = \sqrt{LC_{eff}}\left(tan^{-1}\left(\frac{i_{L,f}\sqrt{L/C_{eff}}}{V_{p,f}-V_{r,f}}\right) - tan^{-1}\left(\frac{i_{L,i}\sqrt{L/C_{eff}}}{V_{p,i}-V_{r,i}}\right)\right) \tag{3.9}$$

Refer to [4] for more information on computing ideal switching times. Code for computing switching times can also be found in Appendix D.

3.3 Nonideal Steady State Solution

A more complex computation is necessary to obtain the exact PSS solution when PR mechanical loss, or R, is considered. A major advantage of the ideal PSS solution is that its equations are purely offset sinusoids represented by ellipses (or circles) in the state space and do not require time domain analysis. However, accurately computing the energy dissipated through R requires knowledge of the current flowing through R at every point in time, so moving the analysis to the time domain is necessary.

The time domain waveforms are computed from the circuit's differential equations. As a reminder, the equivalent circuit diagrams for connected and open stages can be found in Figure 3-3. Connected stages are described by the following differential equations:

$$\frac{dv_r}{dt} = \frac{i_L(t)}{C_r} \tag{3.10}$$

$$\frac{di_L}{dt} = \frac{v_p(t) - v_r(t) - Ri_L(t)}{L} \tag{3.11}$$

During connected stages, v_p is held constant by the switching sequence at some voltage V_p. Open stages instead have the same equations for v_r and i_L and the following additional equation to describe the change in v_p:

$$\frac{dv_p}{dt} = -\frac{i_L(t)}{C_p} \tag{3.12}$$

Thus, both connected and open stages form a second order system of linear constant-coefficient differential equations. The solutions of these differential equations are damped complex exponentials with the following decay rate and oscillation frequencies:

$$\alpha = -\frac{R}{2L} \tag{3.13}$$

$$\omega_{conn} = \sqrt{\frac{1}{LC_r} - \alpha^2} \tag{3.14}$$

$$\omega_{open} = \sqrt{\frac{1}{LC_{eff}} - \alpha^2} \tag{3.15}$$

v_p and v_r have the same centers of resonance as described in Section 3.1. The time-domain waveforms are second-order, so they must have two initial conditions to be uniquely defined. These differential equations can be easily solved analytically using MATLAB or similar software, and code that accomplishes this can be found in Appendix D.

Now that we have the general time-domain solutions for connected and open stages, we can construct a system of equations analogous to the CoE and CoC equations used in Section 3.2. As before, each stage adds two variables representing the values of v_r and i_L at the beginning of the stage. Now, each stage also adds another variable t_x which represents the exact time duration of that particular stage. The equations for a stage are formed by setting the initial conditions of the general time-domain waveforms to be the initial variables of the stage and requiring that the state variables equal the initial variables of the following stage after t_x time has passed.

During connected stages, only v_r and i_L resonate, so connected stages only contribute two constraining equations. We denote these equations with the vector function, which depends on the current time and the stage's initial variables. The x should be replaced with the appropriate stage number:

$$\overrightarrow{C_x}(t_x; V_{p,i}, v_{r,i}, i_{L,i}) = \begin{bmatrix} V_{r,conn}(t_x; V_{p,i}, v_{r,i}, i_{L,i}) \\ i_{L,conn}(t_x; V_{p,i}, v_{r,i}, i_{L,i}) \end{bmatrix} = \begin{bmatrix} v_{r,f} \\ i_{L,f} \end{bmatrix} \tag{3.16}$$

Open stages can be represented similarly to connected stages. However, open stages contribute three constraining equations since all three states v_p, v_r, and i_L participate in resonance:

$$\overrightarrow{O_x}(t_x; V_{p,i}, v_{r,i}, i_{L,i}) = \begin{bmatrix} V_{p,open}(t_x; V_{p,i}, v_{r,i}, i_{L,i}) \\ V_{r,open}(t_x; V_{p,i}, v_{r,i}, i_{L,i}) \\ i_{L,open}(t_x; V_{p,i}, v_{r,i}, i_{L,i}) \end{bmatrix} = \begin{bmatrix} V_{p,f} \\ v_{r,f} \\ i_{L,f} \end{bmatrix} \tag{3.17}$$

We see that, just as in the ideal case, connected stages contribute one more variable than equation, while open stages contribute an equal number of stages and equations. A six stage sequence will have 15 equations and 18 variables. Factoring the i_L zero crossing constraints brings the number of equations to 17, again leaving a single free variable. Another set of open stage equations can also be added to represent a two-part open stage if necessary.

These equations can only be solved numerically, and there is no general closed form solution. Because these equations are not polynomial, and there are more total equations, the nonideal PSS solution is much more computationally intensive to solve than the ideal PSS solution. Numerical solvers based on Newton's method will not converge reliably with a random initial guess. To aid convergence and computation time, an initial guess for the solution can be computed using the ideal PSS solution and switching times (see Section 3.2). Code that implements and computes the nonideal PSS solution for a general switching sequence can be found in Appendix D.

Chapter 4

PR Converter Control

In this chapter, we will propose multiple feedback control schemes for PR-based dc-dc converters. First, we derive the main control handle for a switching sequence and operating range by defining the regulating and nonregulating half-bridges. Then, we present two control methods for the regulating half-bridge and one control method for the nonregulating half-bridge. We also implement the control for the $V_{in} - V_{out}$, $Zero$, V_{out} switching sequence.

4.1 Regulating and Nonregulating Half Bridges

We can use the operating range concepts from Chapter 2.3 to get a better understanding of how to control PR-based dc-dc converters. We discussed that, using CoC and CoE equations, the range of possible output voltages can be derived. We also determined that by "trading off" between two specific charge quantities, we can regulate the output voltage. By analyzing the roles of each switch in maintaining the switching sequence, we can concretely map switch duty ratios to output voltage regulation.

We will analyze the $V_{in} - V_{out}$, $Zero$, V_{out} switching sequence as an example. Figure 4-1 shows the circuit topology, and Figure 4-2 shows the PR and switching waveforms for the $V_{in} - V_{out}$, $Zero$, V_{out} switching sequence with $V_{out} < \frac{1}{2}V_{in}$. As discussed in Chapter 2.2, S1 and S3 are on during Stage 1, S2 and S3 are on during Stage 3, and S2 and S4 are on during Stage 5. Figure 4-3 illustrates the connected stage charge

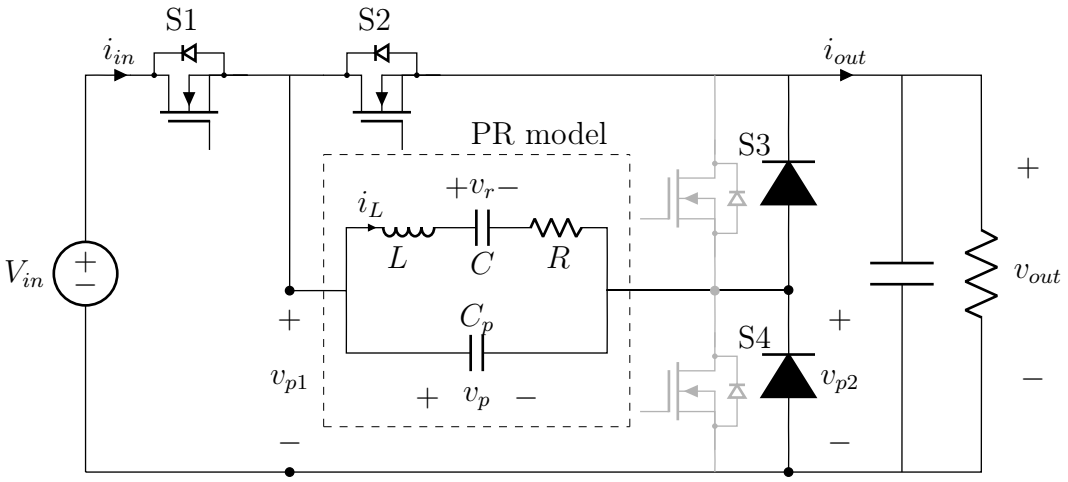


Figure 4-1: PR converter topology for switching sequence $V_{in} - V_{out}$, $Zero$, V_{out}. The PR equivalent circuit as given in [23] is within the dotted lines. Switches are labeled $S1 - S4$ and the PR terminals are labeled v_{p1} and v_{p2}. v_p is the PR voltage and i_L is the current through the PR series inductor.

transfer for the $V_{out} < \frac{1}{2}V_{in}$ case (neglecting open stages for clarity), and Figure 4-4 illustrates the connected stage charge transfer for the $\frac{1}{2}V_{in} < V_{out} < V_{in}$ case. For the $V_{out} < \frac{1}{2}V_{in}$ case, we see that adjusting the duty ratio of the S1-S2 half-bridge modulates the transition point between q_1 and q_3, while the S3-S4 half-bridge's duty cycle is constrained by the i_L zero crossings. Conversely, in the $\frac{1}{2}V_{in} < V_{out} < V_{in}$ case, the half-bridges swap roles, and the S3-S4 half-bridge duty ratio modulates the transition between q_3 and q_5.

Based on this analysis, we define the terms "regulating half-bridge" and "nonregulating half-bridge." The regulating half-bridge is capable of modulating the proportion of charge between two connected stages using its duty cycle, which in turn requires a change in the output voltage to maintain energy balance. The non-regulating half-bridge is constrained by the switching sequence's i_L zero crossings and operates at a fixed duty ratio (approximately 50%). This half-bridge designation can be applied to any six-stage switching sequence (along with corresponding i_L zero crossing constraints) that can implemented on a topology with two half-bridges, and we will focus on control for these types of topologies. However, the operating range analysis can be applied to all six-stage switching sequences proposed in [4].

We name the individual switches within regulating and nonregulating half-bridges

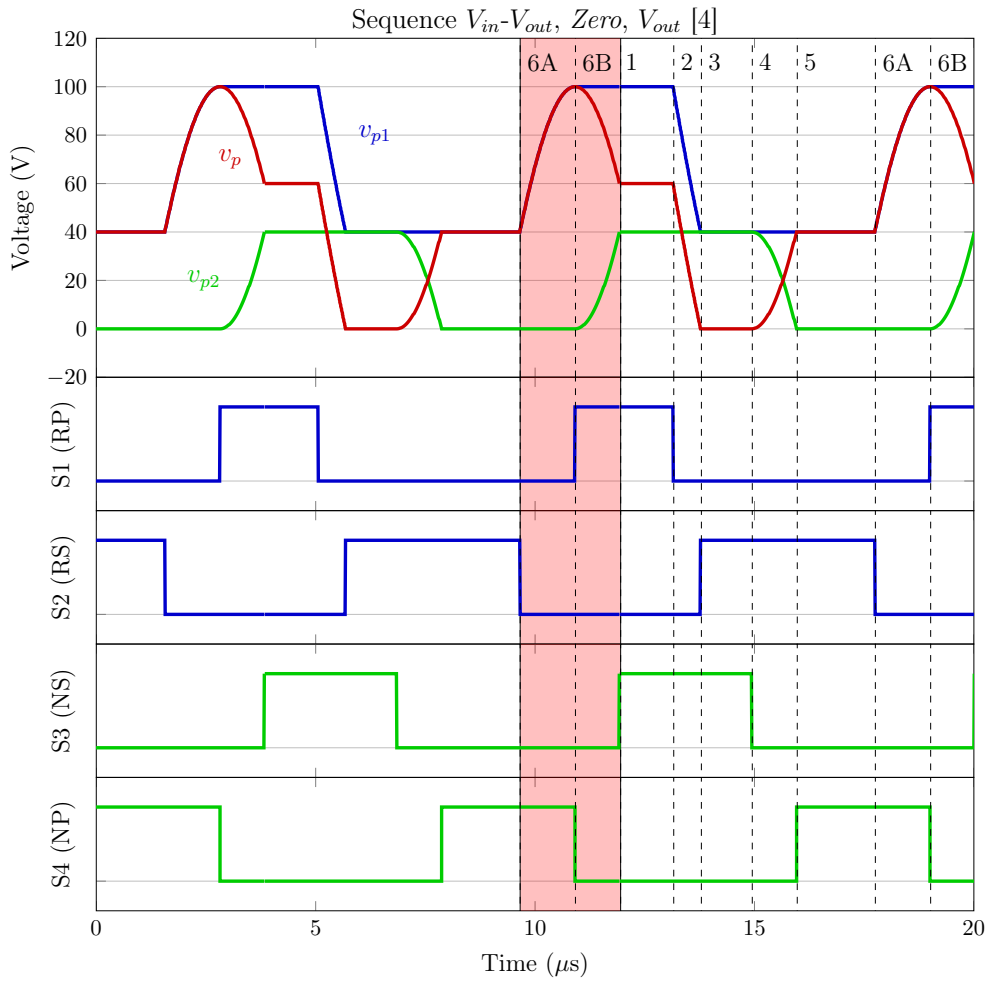


Figure 4-2: Simulation of $V_{in} - V_{out}$, *Zero*, V_{out} with $V_{out} < \frac{1}{2}V_{in}$ with $V_{in} = 100V$, $V_{out} = 40V$, and $P_{out} = 6W$. The corresponding charge transfer distribution among connected stages can be seen in Figure 4-3. S1 and S2 form the regulating half-bridge. The two-part open stage is constituted by stages 6A and 6B, and highlighted in red. S1 and S4 both change state at the start of stage 6B, so they are designated as RP and NP, respectively. S2 and S3 are then designated as RS and NS, respectively. The PR parameters used are $C_p = 4.3$nF, $C_r = 1.4$nF, $L = 1.4$mH, and $R = 2.4\Omega$.

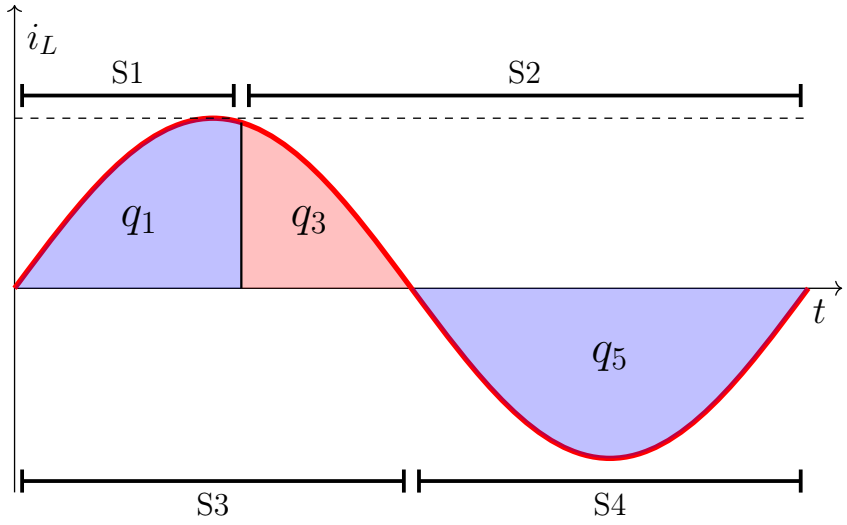


Figure 4-3: Plot of connected stage charge transfers for $V_{in} - V_{out}$, $Zero$, V_{out} with $V_{out} < \frac{1}{2}V_{in}$. The output voltage can be regulated by trading off between q_1 and q_3. S1 and S2 form the regulating half-bridge while S3 and S4 form the nonregulating half-bridge.

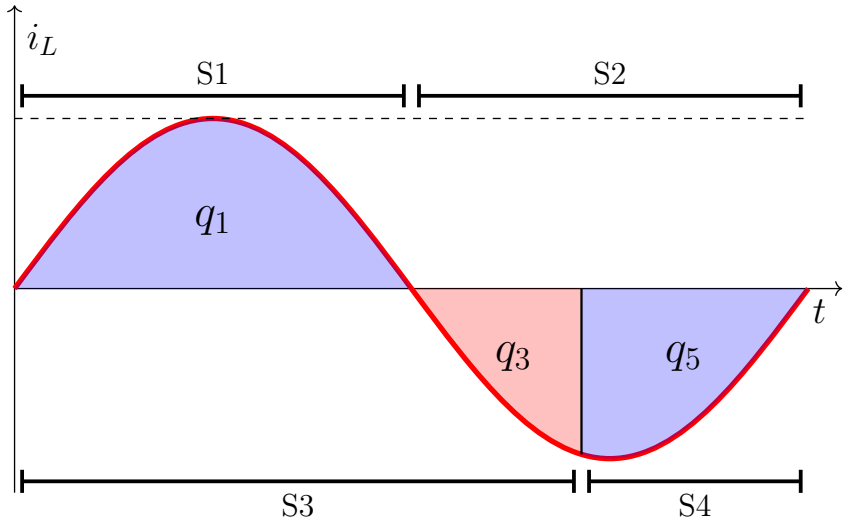


Figure 4-4: Plot of connected stage charge transfers for $V_{in} - V_{out}$, $Zero$, V_{out} with $\frac{1}{2}V_{in} < V_{out} < V_{in}$. The output voltage can be regulated by trading off between q_3 and q_5. S1 and S2 form the nonregulating half-bridge while S3 and S4 form the regulating half-bridge.

as primary or secondary based on their relation to the switching sequence's two-part open stage. The regulating primary switch (RP) and the nonregulating primary switch (NP) turn on or off at the transition point between the open stage's two parts (i.e., the boundary between stage 6A/6B in this example). The regulating secondary switch (RS) and the nonregulating secondary switch (NS) are off throughout the entire two-part open stage. This is further illustrated in Figure 4-2. For a given switching sequence and operating mode, the physical switches S1-S4 each take the role of one of the previously named "conceptual" switches. Throughout this chapter, we will use the "*on*" subscript to refer to a switch's on time duration and the "*dt*" subscript to refer to the dead time duration preceding a switch's on time. For example, RP_{on} indicates the on time duration of the regulating primary switch, and $S2_{dt}$ refers to the dead time duration before S2 turns on.

4.2 Regulating Half Bridge Control

As described in Section 4.1, a switching sequence's regulating half-bridge controls v_{out} with its duty cycle. However, it is not enough to control just the duty cycle; we also need to control the dead times and switching period to achieve all of the desired high efficiency behaviors described in Chapter 2.2. We present two control methods for the PR converter's regulating half-bridge, named "sensed" control and "static" control. These general control concepts may be applied to the regulating half-bridge of most six-stage switching sequences.

4.2.1 Sensed Control

Sensed control is a straightforward strategy in which both switches of the regulating half-bridge are triggered on by sensed voltage measurements and their on-times are the primary control variables. This general strategy is proposed in [22] for a switching sequence catering to high step-down ratios. RP and RS are turned on by sensing the PR node voltages v_{p1} and v_{p2}. RP is turned on when the i_L zero crossing during the two-part open stage is detected. In the design implemented in this thesis, the zero-

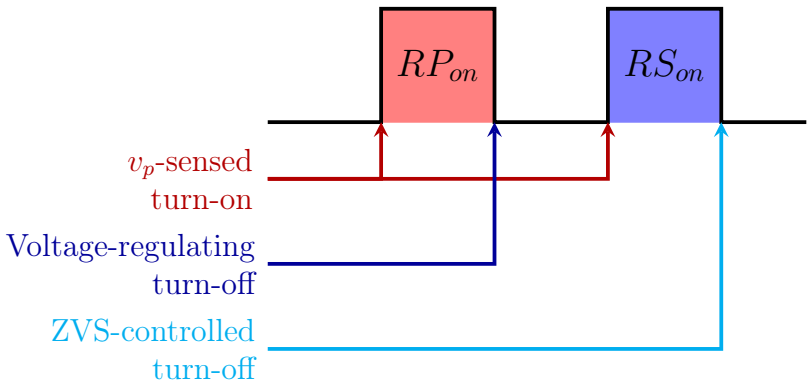


Figure 4-5: Description of switch function and control variables during sensed control. For the switching sequence of Fig. 4-3, ZVS-controlled turn off refers to controlling S2's turn off to allow resonance of v_p up to V_{in} for ZVS of S1.

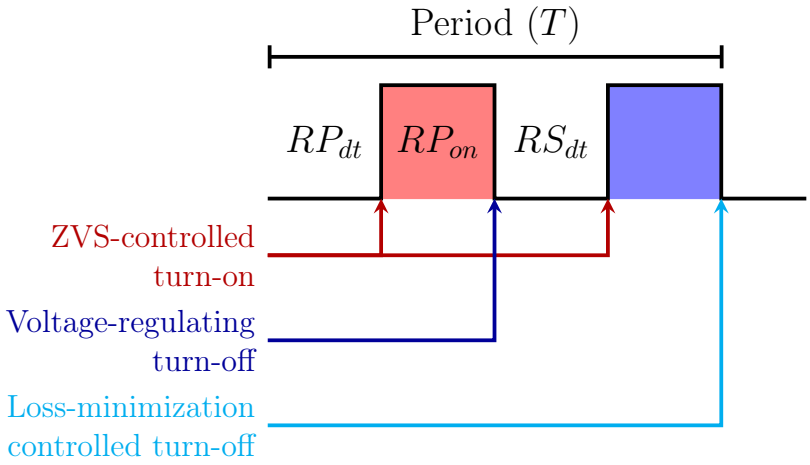


Figure 4-6: Description of switch function and control variables during static control. Loss-minimization-controlled turn off refers to maintaining both ZVS and all-positive instantaneous power transfer (to minimize circulating currents).

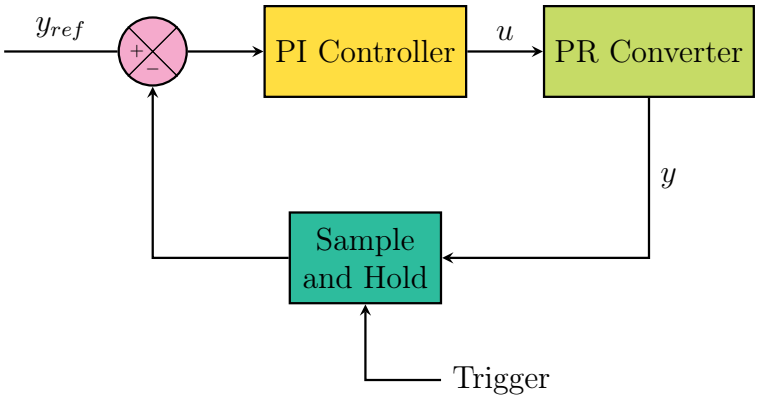


Figure 4-7: Block diagram describing the feedback loops used in both sensed and static control. Every PR cycle, the output y from the converter is sampled at the trigger point, just before the given switch turns on. See Table 4.1 for corresponding values of y, y_{ref}, and u.

current detection relies on implementing the nonregulating half-bridge with diodes, and sensing the voltage of the diode acting as NP (here, S4). RS is turned on when its drain-to-source voltage is 0, achieving ZVS of RS. RP_{on} and RS_{on} are controlled by independent feedback loops. As illustrated in Fig. 4-5, RP_{on} regulates v_{out} to the desired output voltage V_{cmd}, and RS_{on} ensures ZVS of RP and all-positive instantaneous power transfer for maximum efficiency. In periodic steady state, RP_{on} is a free control handle for regulation, but RS_{on} is constrained by the switching sequence's desired behaviors.

For the $V_{in} - V_{out}$, $Zero$, V_{out} sequence with $V_{out} < \frac{1}{2}V_{in}$, RP (S1) is turned on at the start of stage 6B, where an i_L zero crossing occurs. This zero crossing can be detected by sensing v_{p2}, which rises above zero when an assumed-diode at (NP) S4 stops conducting current. This ensures that the controller's switching signals are synchronized to the i_L cycle, which is necessary for stable control. The error between the measured v_{out} and desired output V_{cmd} drives RP_{on}'s ($S1_{on}$'s) feedback loop as illustrated in Fig. 4-7 and Table 4.1.

RS (S2) is turned on at the beginning of stage 3 when v_{p1} falls to v_{out}, or equivalently when v_p falls to 0. To control NP_{on} ($S2_{on}$) for ZVS of RP (S1), v_{p1} is measured when the i_L zero crossing determining RP (S1) turn-on occurs. The difference between that measurement and V_{in} drives $S2_{on}$'s feedback loop. This is also illustrated

Control Method	Control Variable (u)	Measured Output (y)	Desired Value (y_{ref})	Trigger (Switch Turn-on)
Sensed	RP_{on} ($S1_{on}$)	v_{out}	V_{cmd}	—
	RS_{on} ($S2_{on}$)	v_{p1}	V_{in}	RP (S1)
Static	RP_{on} ($S1_{on}$)	v_{out}	V_{cmd}	—
	RP_{dt} ($S1_{dt}$)	v_{p1}	V_{in}	RP (S1)
	RS_{dt} ($S2_{dt}$)	v_{p1}	v_{out}	RS (S2)
	T	t_{α}	$\frac{1}{2}t_{\beta}$	—
	NP_{dt} ($S4_{dt}$)	v_{p2}	0	NP (S4)

Values correspond to Fig. 4-7.

Table 4.1: Control loop variables for static and sensed control for the $V_{in}-V_{out}$, $Zero$, V_{out} sequence with $V_{out} < \frac{1}{2}V_{in}$.

Control Method	Control Variable (u)	Measured Output (y)	Desired Value (y_{ref})	Trigger (Switch Turn-on)
Static	RP_{on} ($S4_{on}$)	v_{out}	V_{cmd}	—
	RP_{dt} ($S4_{dt}$)	v_{p2}	0	$RP(S1)$
	RS_{dt} ($S3_{dt}$)	v_{p2}	v_{out}	$RS(S3)$
	T	t_{α}	$\frac{1}{2}t_{\beta}$	—
	NP_{dt} ($S1_{dt}$)	v_{p1}	V_{in}	$NP(S1)$

Values correspond to Fig. 4-7.

Table 4.2: Control loop variables for static control for the $V_{in} - V_{out}$, $Zero$, V_{out} sequence with $\frac{1}{2}V_{in} < V_{out} < V_{in}$.

in Fig. 4-7 and Table 4.1.

4.2.2 Static Control

Voltage-sensed switching is often used in high-frequency power converters (e.g., [15, 10]), but hardware limitations constrain the upper bound for switching frequency. Moreover, this approach is susceptible to high-frequency noise, and spurious modes can also interfere with voltage measurements, causing erroneous switch turn-ons. To avoid these issues, we propose static control, where the controller instead directly controls the switching period, duty cycle, and dead times for RP and RS, rather than inferring them with sensed RP and RS turn-ons.

RP_{on} is still used to regulate the output voltage as with sensed control, but in static control, the controller has three other variables as illustrated in Figure 4-6: the RP dead time RP_{dt}, the RS dead time RS_{dt}, and the switching period T.

- RP_{on} is used to regulate v_{out} and is controlled by the error between v_{out} and its desired value (V_{cmd}).

- RP_{dt} and RS_{dt} are controlled for ZVS of RP and RS, respectively. The corresponding PR node voltage is measured just before the switch turns on, and error is computed based on the voltage required for ZVS. Specific values depend on the switching sequence, operating range, and topology.

- T is controlled to align RP's turn on with the i_L zero crossing during the two-part open stage using the zero crossing alignment error as discussed in Section 4.2.3.

Because the other times are fixed, a change in period amounts to a change in RS_{on}. Figure 4-7, Table 4.1, and Table 4.2 describe the feedback loops for each control variable and the specific values used for the $V_{in} - V_{out}$, $Zero$, V_{out} sequence.

4.2.3 PR Inductor Current Zero Crossing Detection

Aligning the i_L current zero crossing with the turn on of S1 maximizes efficiency by preventing extra circulating currents and reverse power flow to the input. The i_L zero crossing can be detected without relying on sensing by measuring the width of the v_p resonant waveform in a two-part open stage (stages 6a and 6b in $V_{in} - V_{out}$, $Zero$, V_{out}). In the open-circuited PR:

$$\frac{dv_p}{dt} = -\frac{i_L}{C_p} \tag{4.1}$$

Thus, i_L transitioning from negative to positive results in a local maximum for v_p, so detecting this local maximum is equivalent to detecting the desired zero crossing. Since PRs tend to have high quality factors, v_p is approximately sinusoidal during open stages and is therefore symmetric around the local maximum. Thus, as illustrated in Figure 4-8, two points on v_p with equal voltage must be equally spaced in time from the local maximum, and the i_L zero crossing occurs temporally halfway between these points. This geometry-based strategy can be used to estimate the temporal location of the i_L zero crossing with respect to the switching sequence and detect misalignment as illustrated in Fig. 4-9. This argument assumes that the switch capacitances are negligible compared to C_p, or that they are nearly equal such that the effective capacitance in stages 6a and 6b are approximately equal.

In the $V_{in} - V_{out}$, $Zero$, V_{out} sequence with $V_{out} < \frac{1}{2}V_{in}$, the minimum voltage level common to both stage 6a and 6b is $V_{in} - V_{out}$, so the widest symmetric portion of v_p is enclosed by $v_p = V_{in} - V_{out}$. The i_L zero crossing can be approximated as occurring temporally halfway between v_p rising above $V_{in} - v_{out}$ in stage 6a and falling back to $V_{in} - v_{out}$ at the end of stage 6b. This can be practically measured using the point where v_{p1} rises above $V_{in} - v_{out}$ and the point where v_{p2} rises to v_{out} at the end of stage 6b. This method is much less sensitive to noise and interference from spurious modes than the method used with sensed control since the required comparator measurements occur where v_{p1} and v_{p2} have steeper slopes.

We note that this method for detecting the i_L zero crossing can only be used with

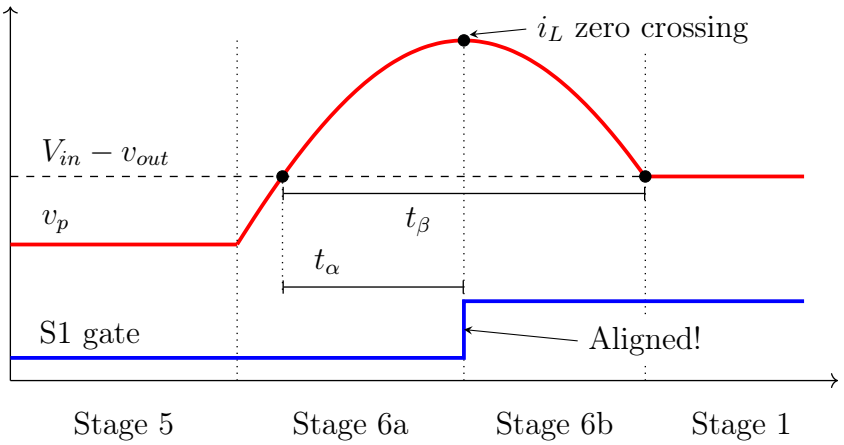


Figure 4-8: Plot illustrating how the i_L zero crossing can be detected by observing symmetry in v_p. In this example, S1's turn on is exactly aligned with the zero crossing, so we have $t_\alpha = \frac{1}{2}t_\beta$.

static control and is incompatible with sensed control. For sensed turn-on of S1, the system must know the zero crossing's temporal location as soon as the zero crossing occurs. However, the location of the zero crossing in this proposed strategy is fundamentally not known until after the zero crossing occurs. (One could instantaneously detect the zero current point by detecting the zero crossing of a differentiated version of v_p, but such a measurement is sensitive to noise.)

4.3 Non-Regulating Half-Bridge Control

For $V_{in} - V_{out}$, $Zero$, V_{out} with $V_{out} < \frac{1}{2}V_{in}$ and the chosen topology, the non-regulating half-bridge consists of S3 (NS) and S4 (NP) and can be implemented with diodes. However, to achieve maximum converter efficiency, S3 and S4 may be implemented with MOSFETs to avoid diode forward voltage drops (i.e., as "synchronous rectifiers"); this strategy is commonly utilized in resonant converters. We propose a synchronous rectifier control strategy intended to accompany static control in Section 4.2.2 such that all four switch signals are generated.

As discussed in Section 4.1, the nonregulating half-bridge operates at a duty cycle of nearly 50% since it is constrained by the i_L zero crossings. Additionally, we note that the dead times of NP and NS are approximately equal since they both occur

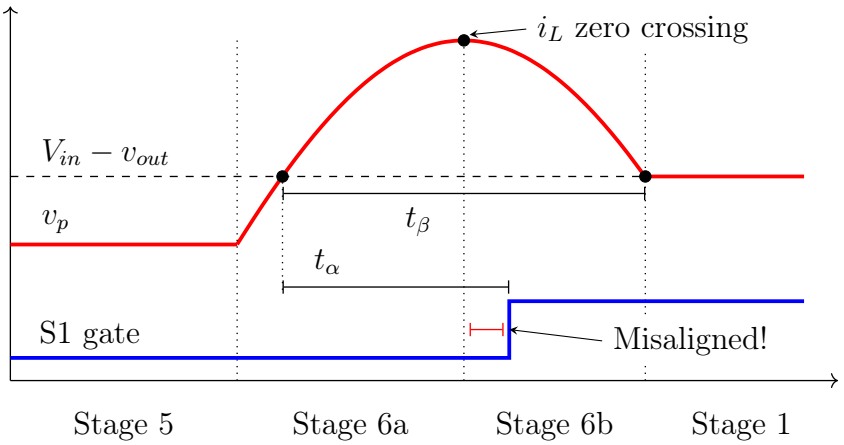


Figure 4-9: Plot illustrating how the i_L zero crossing can be detected by observing symmetry in v_p. In this example, S1's turn on is misaligned with the zero crossing and occurs late, so we have $t_\alpha > \frac{1}{2}t_\beta$. The switching period would be decreased in response to this misalignment.

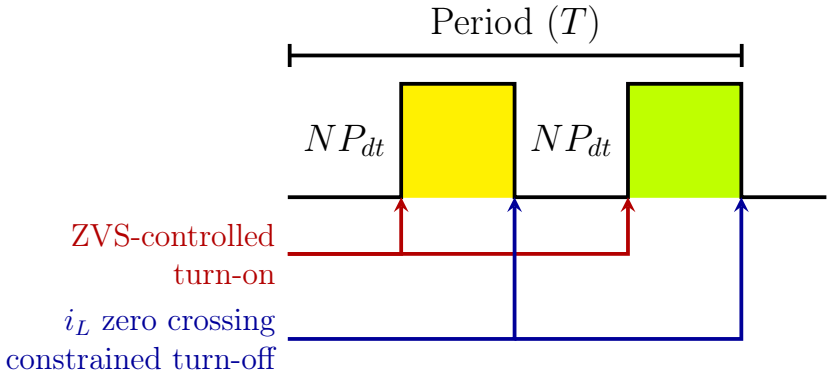


Figure 4-10: Description of switch function and control variables during static non-regulating control. Duty cycle is 50%. NP_{dt} and NS_{dt} are equal, and both expressed as NP_{dt}.

after a zero crossing and transfer the same magnitude of charge into or out of C_p. The period of the nonregulating half-bridge will be the same T already determined by regulating half-bridge control. We can also determine the phase offset of the non-regulating half-bridge from the regulating half bridge with the two-part open stage: either NP turns off when RP turns on, or vice versa, depending on the switching sequence and operating mode. Using these facts, we can fully specify the nonregulating half bridge with just one more parameter. By convention, we choose NP_{dt}, which is controlled to achieve ZVS of NP (and consequently NS).

Figure 4-10 shows diagram describing switch functions in nonregulating half-bridge

control. The specific control parameters used for the sequence $V_{in} - V_{out}$, $Zero$, V_{out} can be seen in Fig. 4-7, Table 4.1, and 4.2.

Chapter 5

PR Converter Control Simulation and Modeling

Simulations are an important tool for testing and understanding the control methods described in Chapter 4. This chapter explores several methods for simulating the dynamic response of the PR converter under feedback control, each with varying degrees of accuracy and computational complexity required. The simulations are implemented for the $V_{in} - V_{out}$, $Zero$, V_{out} switching sequence, but are easily extendable to other switching sequences.

5.1 Circuit and Feedback Simulation in Simulink

One way to model converter dynamics is to directly simulate the PR converter circuit and controller. There are many tools for simulating circuits, but we choose to use Simulink over traditional SPICE simulators because Simulink allows the circuit model to be easily connected with more general mathematical functions, allowing creation of a controller model similar to how a physical digital converter would behave. This method of simulation is the most general purpose because it simulates the circuit directly, but is the most computationally expensive as a result.

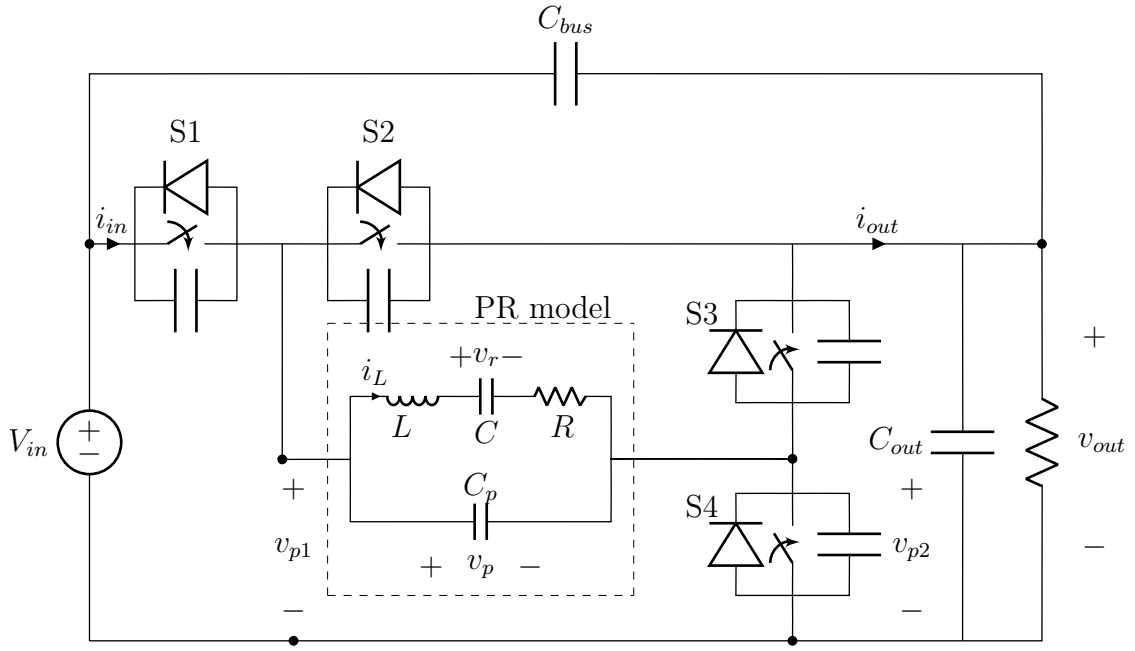


Figure 5-1: Simulink circuit representation for the converter topology implementing the $V_{in} - V_{out}$, $Zero$, V_{out} switching sequence.

5.1.1 Circuit Model

The Simulink package "Simscape" is used to create circuit models of various physical systems, including standard electrical circuits. Simscape allows the circuit to be created and connected graphically, so a given PR converter topology can be easily implemented by creating the circuit schematic accordingly. The circuit model is a Simulink continuous time system, and waveforms of the currents and voltages can be accessed by other parts of the model.

As in earlier sections, the PR is modeled with the Butterworth Van Dyke circuit representation of a capacitor in parallel with a motional RLC branch. We represent the rest of the components as follows: Switches are modeled as ideal switches with an on-state resistance, and are in parallel with a diode and a capacitor. The input is an ideal voltage source, and the output is a resistor with a capacitive filter. We also added a bus capacitor between the input and output. The simplified circuit topology can be seen in Figure 5-1, and an image as well as other specific details of the Simulink model can be seen in Appendix E.

5.1 shows the circuit parameters used. The PR values were chosen based on

Component	Property	Value		Modifiable
PR	Cp	1.41	nF	No
	Cr	510	pF	No
	L	8.73	mH	No
	R	2.3	Ω	No
Switch	Closed Resistance	0.01	Ω	No
	Open Conductance	1×10^{-8}	Ω^{-1}	No
	Threshold	0		No
Switch Parallel Capacitor	Capacitance	1	pF	No
	Series Resistance	1×10^{-6}	Ω	No
	Parallel Conductance	0	Ω^{-1}	No
Switch Parallel Diode	Forward Voltage	0.6	V	No
	On Resistance	0.3	Ω	No
	Off Conductance	1×10^{-8}	Ω^{-1}	No
Output Capacitor	Capacitance	1	μF	No
	Series Resistance	1×10^{-6}	Ω	No
	Parallel Conductance	0	Ω^{-1}	No
Bus Capacitor	Capacitance	1	μF	No
	Series Resistance	1×10^{-6}	Ω	No
	Parallel Conductance	0	Ω^{-1}	No
Input Source	DC Voltage	100	V	Yes
Load Resistor	Resistance	500	Ω	Yes

Table 5.1: Circuit component values in the Simulink Simulation.

impedance analyzer measurements of an APC International Part 1553 resonator. The values for the switch, the switch parallel capacitor, and the switch parallel diode were left as the Simulink default values. The output and bus capacitances were chosen to keep the output voltage ripple at an acceptable level. The input voltage and and load resistor are denoted as "modifiable" because the values can be changed while the simulation is running to simulate transients. The other values can only be modified before the simulation runs.

5.1.2 Controller model

To implement controllers for both sensed and static control modes, we use the control concepts developed in Chapter 4. We used Simulink's Stateflow systems to create a finite state machine that generates the switch signals with appropriate timing. Each half-bridge can be represented with a four-state finite state machine, where each state

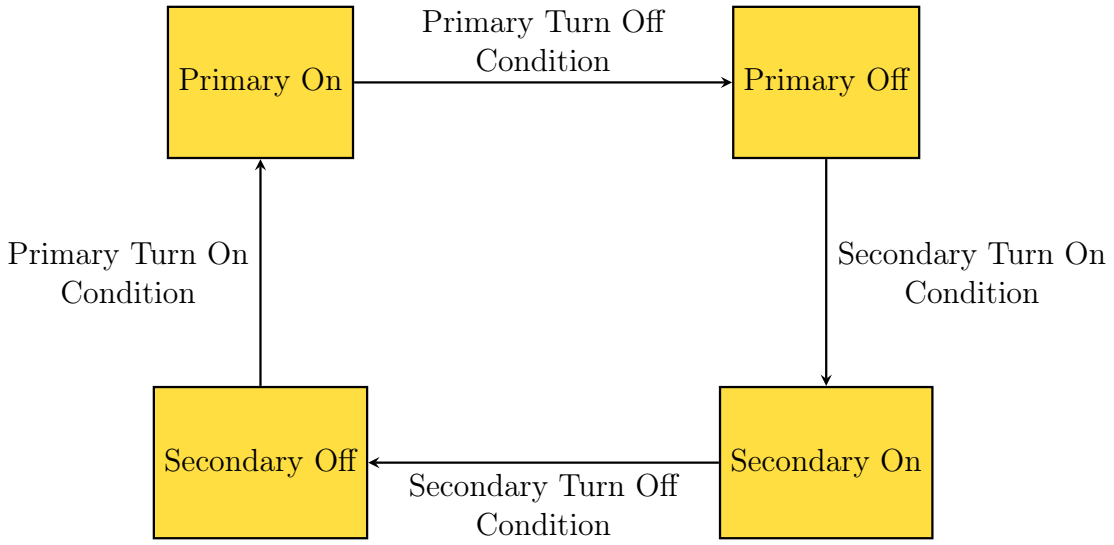


Figure 5-2: Simulink Simulation Switch FSM. Transition conditions can be seen in Table 5.2

transition represents a switch being toggled on or off. If we denote the two switches as the primary and secondary switches, then the states transition as follows:

1. Primary switch turns on
2. Primary switch turns off
3. Secondary switch turns on
4. Secondary switch turns off

The pattern then repeats. The state transitions can be driven by events in the circuit waveforms or occur after a specific amount of time has passed. Table 5.2 summarizes the state transition conditions for both sensed and static control, and a diagram can be seen in Figure 5-2. In sensed control, the switches are turned on by the PR voltage waveforms reaching certain points, while the on times are determined by the controller. In static control, all of the switch on times and dead times are determined by the controller. To start up the simulation, the FSM operates in the static control mode, except that switching times are a fixed to a predetermined PSS solution computed with the methods in Chapter 3. The simulation will run until PSS is reached, then the FSM changes operation and the feedback loop begins operation.

Control Mode	Transition	Condition
Sensed Control Regulating HB	Primary Turn On	When $v_{p2} >= 0$
	Primary Turn Off	After RP_{on}
	Secondary Turn On	When $v_{p1} <= v_{out}$
	Secondary Turn Off	After RS_{on}
Static Control Regulating HB	Primary Turn On	After RP_{dt}
	Primary Turn Off	After RP_{on}
	Secondary Turn On	After RS_{dt}
	Secondary Turn Off	After RS_{on}

Table 5.2: Simulink Controller Switch Transition Conditions

Continuous-time PI compensators are used to determine the switch on times from the circuit waveforms. The output voltage is already filtered by the output capacitors, so the output voltage error waveform can be used directly. For ZVS correction, sample-and-hold modules are used to capture the value of the a voltage waveform just before the corresponding switch turns on. The sampled-and-held ZVS error waveform is then used to drive the PI compensator. For implementing the zero crossing detector, integrators are used as timers, and the zero crossing offset error is computed every cycle, and stored with a sample and hold. The specific Simulink block diagrams for each of the switch on time feedback systems can be found in Appendix E.

5.2 Piecewise Linear Numerical Simulation

Another method for simulating converter dynamics involves modifying the CoE and CoC system of equations that were used for modeling steady state operation in Chapter 3.2. Rather than using the CoE and CoC equations to solve for a specific steady state solution, they can instead be applied to a set of initial conditions to determine the state evolution over a single PR resonant cycle, as determined by the switching sequence. This process can then be repeated for the desired number of resonant cycles to compute converter dynamics. This method of simulation is much faster than the circuit simulation techniques used in Section 5.1, however its use is more limited.

This simulation implements a modified version of sensed control where only RP_{on} control is used, and RS_{on} is automatically solved for so ZVS is perfectly reached

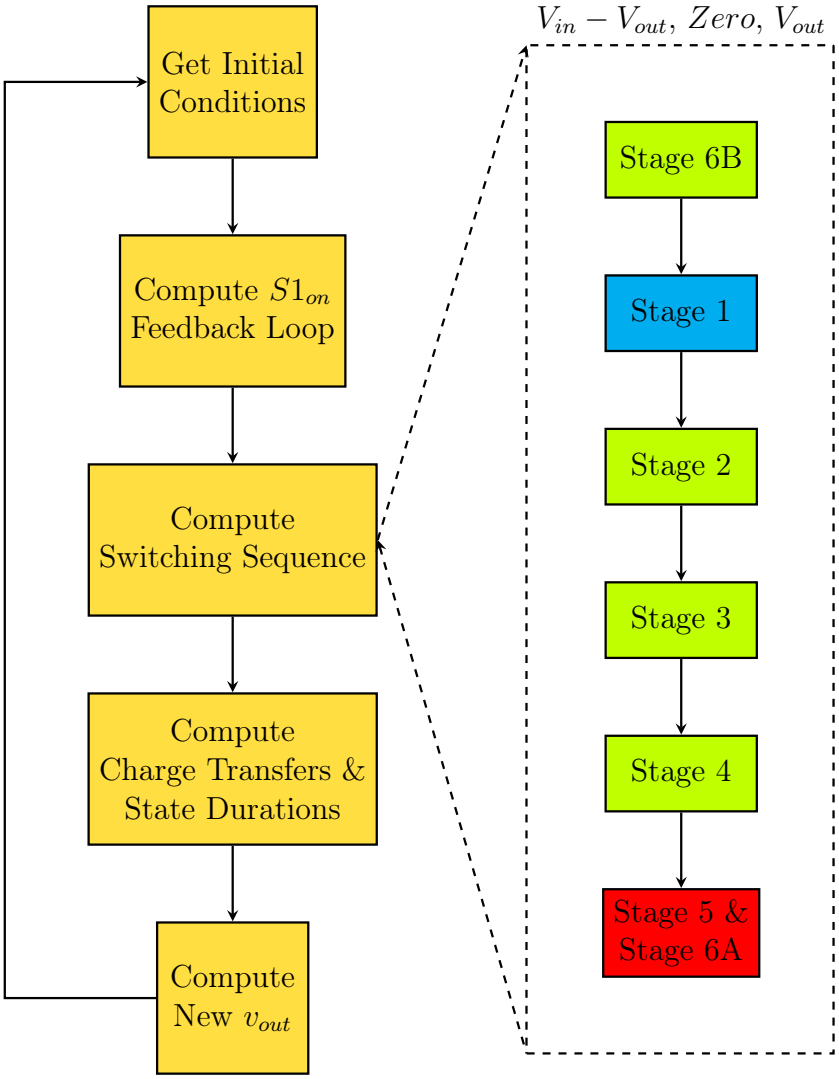


Figure 5-3: Block diagram for Piecewise Linear simulation procedure.

during the two part open stage. The evolution of the PR is based on the CoE and CoC equations that correspond to the switching sequence. The output voltage is represented by a resistive load with an output capacitor. Every cycle, RP_{on} is computed based on the error between the current and desired output voltage. During every iteration of the simulation, the following procedure, which is also illustrated in Figure 5-3, is followed:

1. Compute the current RP_{on} based on the V_{out} error.
2. Simulate the PR states across a single switching sequence using CoE and CoC.
3. Determine the connected stage charge transfers and switching times for all stages.
4. Compute the new value of V_{out} based on charge transfers and output current draw.
5. Repeat as desired.

To simulate the switching sequence over a PR cycle, we start with the corner variables for a predetermined stage, then compute the corner variables following every state until we reach the starting state again, now with an updated set of corner variables. If we choose the starting state to be one with an i_L zero crossing, then only one variable is required to represent the current PR state, v_r. Open stages provide two equations, one CoE and one CoC, so the corner variables of the following connected stage are uniquely determined. Connected stages only provide one equation, modified CoE, so another constraint is necessary to solve for the corner variables of the following open stage. The possible constraints are a time duration for the stage, an i_L zero crossing occurring at the end of the connected stage, or constraining for ZVS at the end of the following open stage. At least one connected stage must use the time duration constraint. Once the corner variables for a switching cycle or known, the charge transfers and switching times can be computed using the same methods as described in 3.2.

For a specific implementation using the $V_{in} - V_{out}$, $Zero$, V_{out} sequence with $V_{out} < \frac{1}{2}V_{in}$, we can use the following constraints:

- Starting stage is Stage 6B
- Stage 1 is constrained by time duration
- Stage 3 is constrained by an i_L zero crossing
- Stage 5 is constrained by achieving ZVS at the end of stage 6A

A feedback loop based on the V_{out} error computes $S1_{on}$ every cycle, and the t_1 can be calculated by subtracting t_{6B} from $S1_{on}$, which is used to determine stage 1. Stage 3 is determined using the fact that $i_{L,4} = 0$. Finally, Stage 5 and Stage 6A are solved in parallel so that $V_{p,6B} = V_{in}$, ensuring exact ZVS. (An alternate simulation could implement full sensed control by adding an additional feedback loop for $S2_{on}$ and constraining Stage 5 by taking $t_5 = S2_{on} - t_3 - t_4$.) Figure 5-3 further illustrates this procedure.

The following equations implement the PI compensator for $S1_{on}$, where $S1_{on,int}$ is the integral component of $S1_{on}$, K_P is the compensator proportional coefficienct, K_I is the compensator integral coefficient, T is the period of the current cycle, and V_{cmd} is the desired output voltage:

$$S1_{on,int} = S1_{on,int,prev} + K_I \times T \times (v_{out} - V_{cmd})\,; \tag{5.1}$$

$$S1_{on} = S1_{on,int} + K_P \times (v_{out} - V_{cmd})\,; \tag{5.2}$$

As mentioned earlier, the output of the converter is a resistive load and a capacitive filter, which can be represented with a single state variable. Every PR cycle, the load resistor will draw a quantity of charge based on V_{out}, and the PR will send a certain quantity of charge based on the switching sequence. For $V_{in} - V_{out}$, $Zero$, V_{out}, charge quantity $|q_1| + |q_5|$ is sent to the output, and the change in V_{out} can be computed

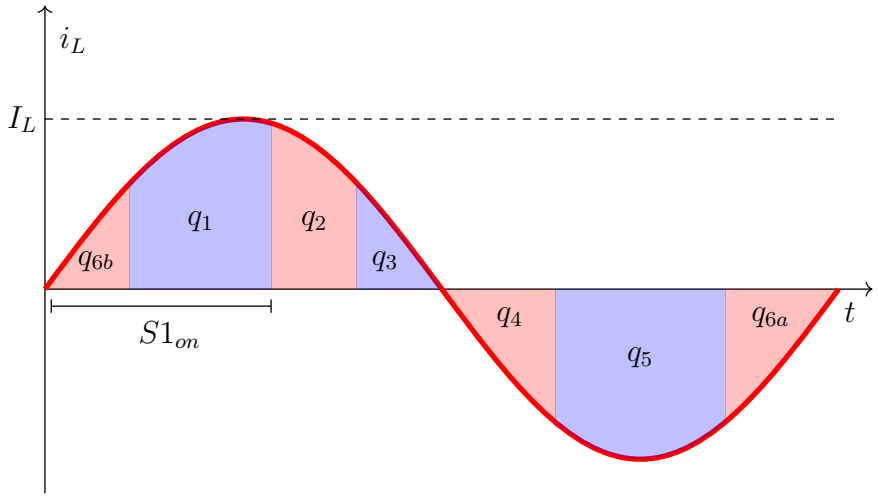


Figure 5-4: Plot of the "amplitude of resonance" approximation of i_L for the $V_{in} - V_{out}$, $Zero$, V_{out} switching sequence with $V_{out} < \frac{1}{2}V_{in}$. Each charge quantity is numbered with its corresponding stage. Open stage charge quantities are colored red. In each half period, the total charge transferred is $\frac{TI_L}{\pi}$ and the charge magnitude transferred in open stages is $C_p V_{in}$.

with the following equation, where C_{out} is the output capacitance and R_{load} is the load resistance:

$$\Delta V_{out} = \frac{1}{C_{out}} \left(|q_1| + |q_5| - \frac{V_{out}}{R_{load}} \times T \right) \tag{5.3}$$

We assume V_{out} can be approximated as constant over a single PR cycle.

This procedure can then be repeated as many times as desired to simulate converter dynamics. The number of PR cycles can be converted to time by storing the period of every cycle. To simulate the effects of a V_{cmd} or R_{load} step, the value can easily be changed at the desired point in time.

5.3 State-Space Model

Taking another approach, we can derive an approximate continuous-time state space dynamic model by using the Amplitude of Resonance (AoR) model [4]. The AoR model approximates the PR inductor current, i_L, as sinusoidal and relates the total charge magnitude transferred via i_L during the PR's resonant cycle to the amplitude

of i_L, which we denote I_L. This is illustrated in Figure 5-4, and more details about the AoR model can be found in [4].

A simplified dynamical state space model can be constructed for the PR converter with two states: v_{out} and I_L. To derive this model, we assume that the PR is ideal, RP_{on} is the only control variable, and the other switching parameters are chosen to ensure exact ZVS, soft-charging, and all-positive instantaneous power transfer. We also assume that T is a constant, which is justified because the frequency range of a PR's inductive region (i.e., the operating region for which ZVS can be obtained) is relatively narrow. (This approximation is also made in other developments using PRs, e.g., [6].) Finally, we assume that I_L and v_{out} are constant within a single PR resonant period.

5.3.1 State Equations

We will now derive the state equations governing the $V_{in} - V_{out}$, $Zero$, V_{out} switching sequence with $V_{out} < \frac{1}{2}V_{in}$. Equations (5.4) and (5.5) describe the state evolution of v_{out} and I_L:

$$\frac{dv_{out}}{dt} = \frac{1}{C_{out}}\left(\frac{|q_1| + |q_5|}{T} - \frac{v_{out}}{R_{load}}\right) \tag{5.4}$$

$$\frac{dI_L}{dt} = \frac{1}{TLI_L}\left(\left(V_{in} - v_{out}\right)|q_1| - v_{out}|q_5|\right) \tag{5.5}$$

These equations are in terms of $|q_1|$ and $|q_5|$, the charge quantities transferred by i_L during stages 1 and 5, respectively, and the following assumed constants: the output capacitance C_{out}, the load resistance R_{load}, the PR static capacitance C_p, the PR motional inductance L, and the switching period T.

(5.4) is derived from charge balance on the output capacitance. Every switching period, the PR delivers the charge quantity $|q1| + |q5|$ to the load. Thus, the average PR output current over a switching period is:

$$i_{PR,avg} = \frac{|q1| + |q5|}{T} \tag{5.6}$$

Additionally, the current drawn by the load is:

$$i_{load} = \frac{v_{out}}{R_{load}} \tag{5.7}$$

The average current into the output capacitor is then the difference between these two current quantities, giving an equation for the time derivative of v_{out}. This approximation is valid when the output capacitor is large enough that v_{out} does not change significantly during a single PR period, and when the PR output current's temporal distribution across the switching period is not relevant.

(5.5) is derived from energy balance on the PR. During stage 1, the source-load system delivers an energy quantity of $\Delta E_1 = (V_{in} - v_{out})|q_1|$ to the PR, and during stage 5, the source-load system extracts an energy quantity of $\Delta E_5 = v_{out}|q_5|$ from the PR. Thus, the time derivative of the energy stored in the PR is:

$$\frac{dE_{stored}}{dt} = \frac{(V_{in} - v_{out})|q_1| - v_{out}|q_5|}{T} \tag{5.8}$$

Assuming the PR is ideal, we can also approximate the energy stored in the PR based on the amplitude of resonance model. If the peak PR inductor current is I_L, then the peak energy stored in the PR is approximately:

$$E_{stored} = \frac{1}{2} L I_L^2 \tag{5.9}$$

The time derivative of energy can then be related to the time derivative of I_L by

$$\frac{dE_{stored}}{dt} = \frac{d(\frac{1}{2} L I_L^2)}{dt} = L I_L \frac{dI_L}{dt} \tag{5.10}$$

(5.5) is then produced by combining (5.8) and (5.10). This approximation assumes the change in energy stored in the PR capacitances is negligible compared to the change in energy stored in the PR inductor when i_L is at its peak.

5.3.2 Charge Transfer Quantities

We can derive expressions for $|q_1|$ and $|q_5|$ as functions of I_L, v_{out}, control handle $S1_{on}$, and the same constants by integrating the appropriate segments of the assumed-sinusoidal i_L waveform:

$$|q_1| = \frac{TI_L}{2\pi}\left(1 - \cos\left(\frac{2\pi}{T}S1_{on}\right)\right) - C_p v_{out} \tag{5.11}$$

$$|q_5| = \frac{TI_L}{\pi} - C_p V_{in} \tag{5.12}$$

The charge transfer required during each open stage is determined by the switching sequence.

(5.11) is derived by integrating i_L over stages 6b and 1, then subtracting the charge transfer during stage 6b to arrive at the charge transferred in stage 1. S1 turns on at the i_L zero crossing and is on for the full duration of stages 6b and 1, so this total charge transfer is a direct function of $S1_{on}$. (5.12) is derived by integrating i_L over stages 4, 5, and 6a and subtracting the open stage charge transfer. Since we assume that exact ZVS is achieved, the total open stage charge transfer in stages 4 and 6a must resonate v_p from 0 to V_{in}.

5.3.3 Model Validation

The state space equations in (5.4) and (5.5) can be linearized around an operating point to simplify their analysis. An operating point consists of an input and output voltage, a load resistance, and a switching period. The switching period can be calculated from the PSS solutions described in Chapter 3. The linear state space model can then be connected in feedback with a PI compensator to model its closed-loop behavior. The linearized equations are as follows:

$$\begin{aligned}\frac{d\widetilde{V_{out}}}{dt} =& -\left(\frac{C_p}{C_{out}T} + \frac{1}{R_{load}C_{out}}\right)\widetilde{V_{out}} \\ &+ \frac{3-\cos\left(\frac{2\pi}{T}\overline{S1_{on}}\right)}{2\pi C_{out}}\widetilde{I_L} \\ &+ \frac{\overline{I_L}}{C_{out}T}\sin\left(\frac{2\pi}{T}\overline{S1_{on}}\right)\widetilde{S1_{on}}\end{aligned} \tag{5.13}$$

$$\begin{aligned}\frac{d\widetilde{I_L}}{dt} =& \left(\frac{1}{2\pi L}\cos\left(\frac{2\pi}{T}\overline{S1_{on}}\right) - \frac{3}{2\pi L} + \frac{2C_p\overline{V_{out}}}{TL\overline{I_L}}\right)\widetilde{V_{out}} \\ &+ -\frac{C_p\overline{V_{out}}^2}{TL\overline{I_L}^2}\widetilde{I_L} \\ &+ \frac{V_{in}-\overline{V_{out}}}{TL}\sin\left(\frac{2\pi}{T}\overline{S1_{on}}\right)\widetilde{S1_{on}}\end{aligned} \tag{5.14}$$

Code that creates the linearized state space equations and connects the system in feedback can be found in Appendix G.

To validate the state space model, we compare it to the piecewise linear model described in Section 5.2. Fig. 5-5 shows a comparison of the ideal simulator to the state space model after a step in V_{cmd}. Both simulations used the same PR parameters, input and output voltages, output capacitance and load resistance, and feedback coefficients. The state space model is linearized around the starting point of V_{in} and $V_{out,i}$. These values, along with the T used with the state space model, can be found in Table 5.3.

Parameter	Value
C_p	1.41nF
C_r	510pF
L	8.73mH
V_{in}	100
$V_{out,i}$	40
$V_{out,f}$	41
C_{out}	1µF
R_{load}	244Ω
K_p	500e-6
K_i	.25e-6
T	13.1µs

Table 5.3: Values used in the Piecewise Linear Simulation and the State Space model comparison.

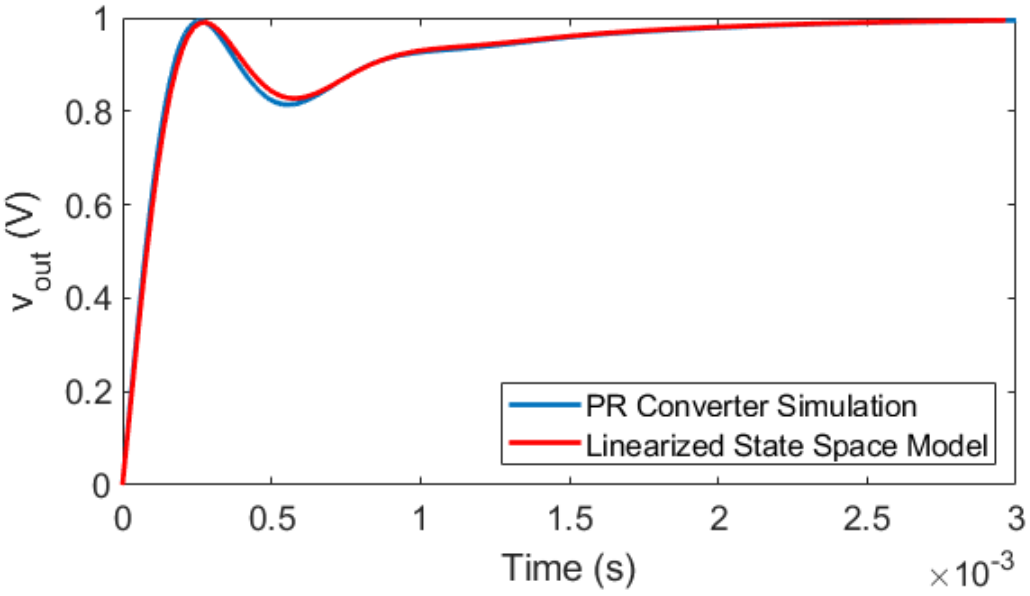


Figure 5-5: Comparison of the linearized state space model to a simulation of the PR converter with $S1_{on}$ feedback. Response to V_{cmd} step of 1V with parameters from Table 5.3.

Chapter 6

PR Converter Hardware Implementation

This chapter covers an implementation of a prototype PR-based dc-dc converter. We will discuss the circuit and its capabilities, the components used, and how to interface with the converter.

6.1 Circuit Description

We implemented the circuit topology described in Chapter 2 on a two-layer 1-oz copper PCB. This topology is primarily capable of realizing the $V_{in} - V_{out}$, $Zero$, V_{out} switching sequence, but it also supports the V_{in}, $V_{in} - V_{out}$, V_{out} switching sequence. An image of the PCB can be found in Figure 6-1, and a schematic of the primary converter circuitry can be found in Figure 6-2. A complete schematic, bill of materials and board layout is shown in Appendix B. PRs can be mounted upright or laid flat on the copper pad, depending on their size. Details of operation with a specific PR can be found in Chapter 8. S1 and S2 are implemented with GaN FETs. S3 and S4 are implemented with discrete diodes in parallel with GaN FETs, providing support for both switching sequences in their full range of operation. Based on switch tolerances, the converter supports a maximum input voltage of 400V and a maximum output voltage of 200V.

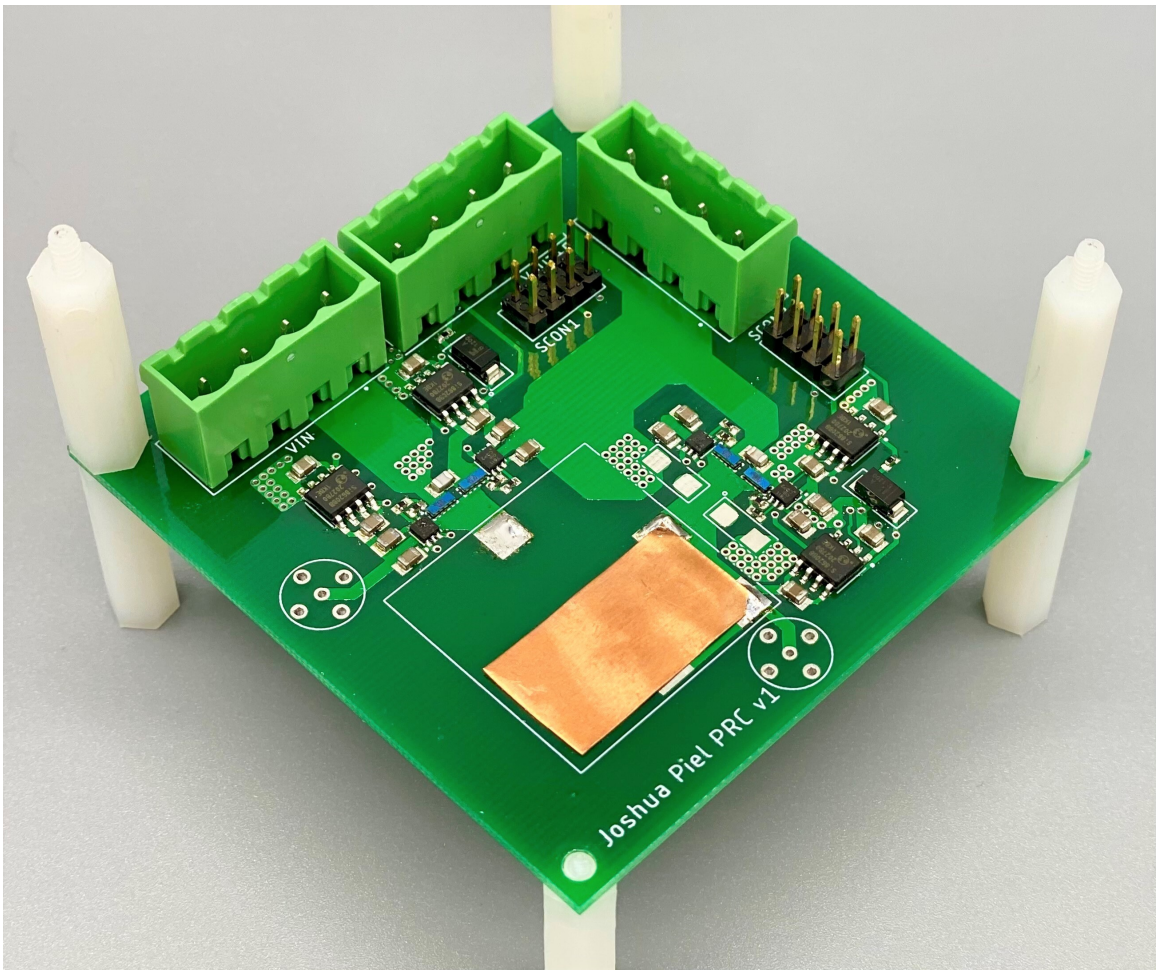


Figure 6-1: Picture the PR converter printed circuit board.

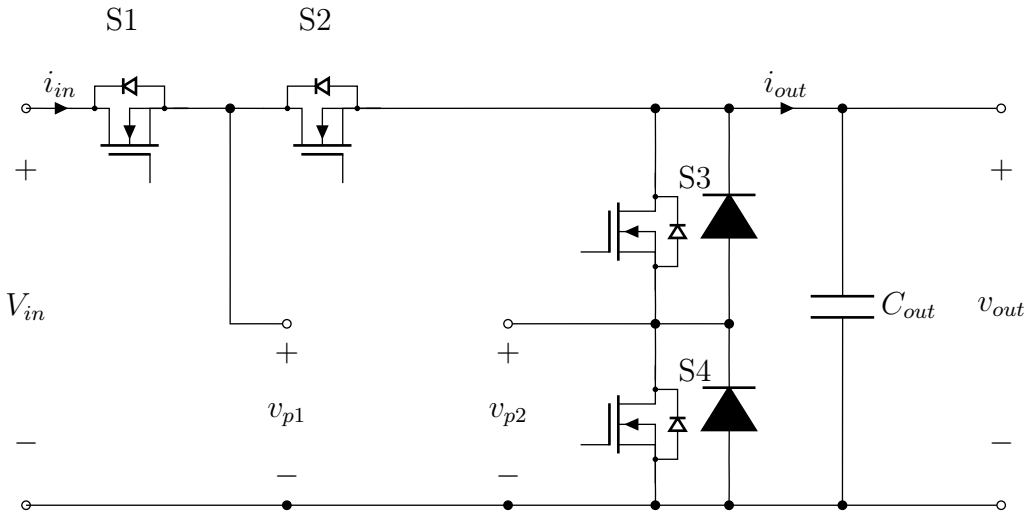


Figure 6-2: Circuit schematic of the main converter topology implemented on the prototype PCB. The PR terminals are connected to v_{p1} and v_{p2}.

Component	Value
Switches	EPC2019 GaN FETs
Diodes	On Semiconductor NSTA4100
Gate Drivers	Texas Instruments UCC27611
Input Capacitance	45µF
Output Capacitance	115µF

Table 6.1: Components used in the PR dc-dc converter prototype.

All of the switches are driven by isolated gate drivers, allowing external gate signals from a controller to be ground referenced. All gate drivers are tied to their respective switch's source node, and a boostrap diode and capacitor are used to power the S1 and S3 gate drivers. An isolated and v_{out} referenced 5V supply is required to power the S1 and S2 gate drive circuitry, and a ground referenced 5V supply is necessary to power the S3 and S4 gate drive circuitry.

A list of important components can be found in Table 6.1.

6.2 Converter Interface

There are several plugs and header pins on the board, which allow for easy connection of the converter to power supplies and controller circuitry. The three large plugs are for connecting the input power supply, the isolated 5V supply for driving S1 and S2, and the load. The header pins provide the inputs for the isolated gate signals and the 5V power supply for S3 and S4's gate drive circuitry. Appendix B.2 shows the pinouts of all ports and headers on the controller.

Chapter 7

Feedback Controller Hardware Implementation

This chapter will cover the specifics on how the control concepts from Chapter 4 can be implemented on a microcontroller and used to control the prototype PR-based dc-dc converter described in Chapter 6. First, we will introduce the important features of the microcontroller. Next, we will explore in detail how all of the hardware features of the microcontroller can be configured to set up the control system. Finally, we will explore the specifics of the code and how controller operation can be configured. The full code and configuration information can be found in Appendix H.

7.1 Microcontroller

The microcontroller used is the Texas Instruments (TI) TMDSCNCD28379D Control Card paired with the TMDSHSECDOCK docking station for easy access to the microcontroller pinout and space to add additional circuitry. An image of the microcontroller can be seen in Figure 7-1. This microcontroller was chosen because it has most of the features needed to implement a full digital controller built in. The relevant features are:

- 32-bit CPU with floating point arithmetic and other advanced math capabilities

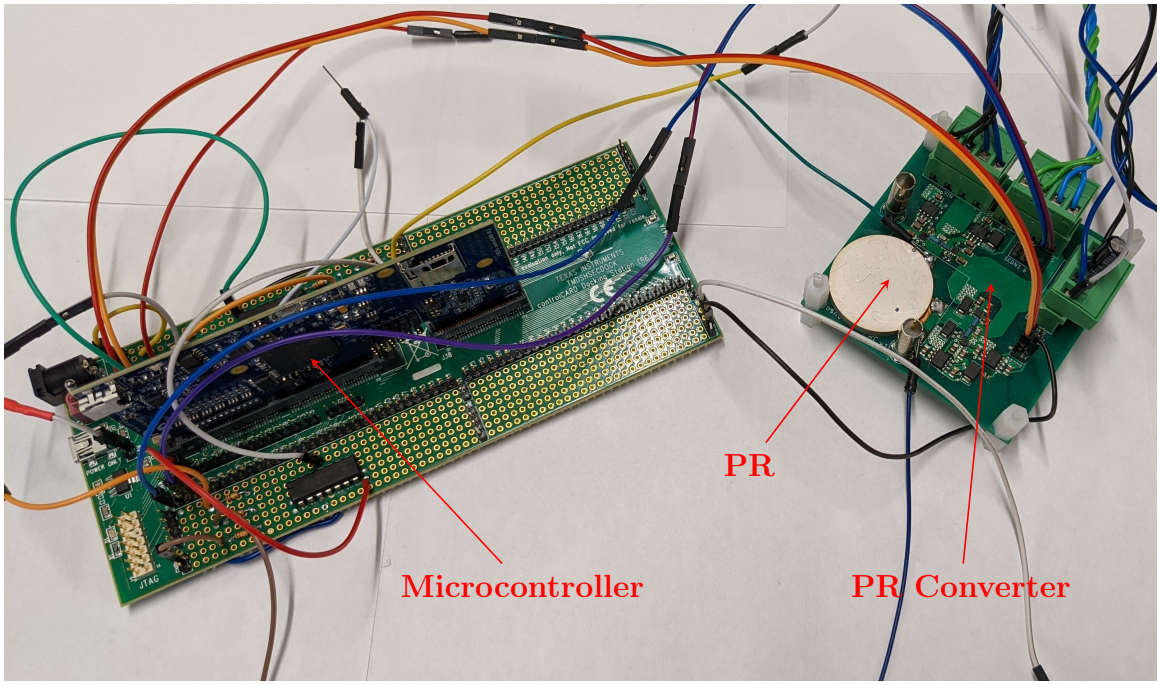


Figure 7-1: Photo of the microcontroller connected to the prototype PR converter.

- 200MHz clock frequency
- 3.3V IO
- Twelve highly configurable and independent PWM outputs
- Four 12-bit analog to digital converters (ADCs)
- Eight configurable and high speed comparators
- Four Configurable Logic Blocks (CLBs), which allow for simple custom digital logic

The microcontroller's PCB has a cartridge slot pinout on the bottom, meaning that it can be easily be connected to a custom PCB, or in our case, the docking station. The docking station was used because it makes all of the microcontroller pins easily accessible, allowing easy prototyping, interfacing with and space for implementing the required sensing circuitry, and reconfigurability if changes to the implementation are required.

The microcontroller can be programmed using the C programming language and TI's C2000Ware library. The code is written using the TI Code Composer Studio (CCS) IDE, which has built in support for C2000Ware, compiling the code, and uploading it to the microcontroller.

7.2 Gate Signal Generation

The primary input to the PR converter are the gates of the four switches. The enhanced Pulse Width Modulation (ePWM) modules on the microcontroller are used to generate these gate signals with precise timing and minimal software intervention.

The ePWMs can be thought of as advanced counters that can be used to generate an output signal with various shapes. The counters can be configured to count up, down, or up-down (where the counter counts up then down), and will increment or decrement the count every clock cycle. At a 200MHz clock frequency, this gives a time resolution of 5ns. It is most convenient to use the count-up mode because it allows the switch on-times to be most easily specified.

The ePWMs have several internal registers used for tracking and maintaining the count. The period register sets the maximum count, which determines when the count resets back to 0. This is used to directly set the switching frequency. The Counter Compare registers are used to perform actions when the count reaches specific values. This is used to set the on times of switches, and to generate signals at specific times to trigger other hardware, like ADC measurements.

Since we need to use four ePWMs, one for each switch on the converter, we need to ensure the gate signals are properly offset in time for correct converter operation and to avoid issues like switch shoot-through. We can accomplish this using the sync and phase shift features of the ePWMs. Each ePWM has a number assigned to it. The twelve ePWMs are labelled ePWM1, ePWM2, and so on up to ePWM12. The ePWMs are split into four groups of three modules, and the first in each group (1, 4, 7, 10) can generate sync signals which pass to all higher numbered ePWMs. When an ePWM receives a sync signal, it will reset its counter to its phase value. This allows

Counter Compare	Action
CTR = 0	Set Output High
CTR = CMPA	Set Output Low
CTR = CMPC	ADC Start Of Conversion

Table 7.1: Static Control ePWM Counter Compare Actions

multiple ePWMs to be offset in phase from each other while ensuring they do not drift over time.

Additionally, the ePWMs support shadowing, which means that when the ePWM register values are changed, the values are first loaded into a second set of registers which buffer the values until a specific condition is reached. This allows consistent updating of the ePWM values from the code without having to worry about when exactly the code executes relative to the ePWMs. The ePWMs are configured to load from the shadow registers either when the count overflows or a sync signal occurs.

The following subsections detail the specific ePWM configurations necessary to implement both static and sensed control. Each control mode has specific ways of configuring three main aspects: the Counter Compare actions, Counter Compare registers, and sync behaviors. Code for both static and sensed control can be found in Appendix H.

7.2.1 Static Control Configuration

Static control uses the counter compare actions found in Table 7.1 for all ePWM modules. Because the ePWMs are configured in count-up mode, the output will be set high when the counter equals 0, and low when the counter later reaches CMPA. This means the switch on-time is configured by setting CMPA to the desired value. When the counter equals CMPC, the ePWM creates an internal signal that can trigger an ADC module conversion. This is used to take voltage measurements at precise points in the switching sequence.

To fully specify the output signal timing for each ePWM, we need to configure the period, the CMPA register, and the Phase register. The five parameters available for configuring the ePWMs are T, RP_{on}, RP_{dt}, RS_{dt}, and NP_{dt}. (See Chapter 4 for

Conceptual Switch	Physical Switch	
	($V_{out} < 1/2V_{in}$)	($V_{out} > 1/2V_{in}$)
RP	ePWM1 (S1)	ePWM4 (S4)
RS	ePWM2 (S2)	ePWM3 (S3)
NP	ePWM4 (S4)	ePWM1 (S1)
NS	ePWM3 (S3)	ePWM2 (S2)

Table 7.2: Static control switch functions for the $V_{in} - V_{out}$, $Zero$, V_{out} sequence.

more information on these parameters.) For simplicity, we choose to align all of the timing with one of the switch transitions at the stage 6A/6B boundary during the two-part open stage. This allows each the timing of each half bridge to be specified only with its own parameters and T. Additionally, there are two possible modes that the switch waveforms can take, and they are applicable to many sequences in [4] with two-part open stages. Both modes are controlled identically, and only dictate the relative offsets of the regulating and nonregulating half-bridges. In Mode 1, RP turns on during the two part open stage (e.g., $V_{in} - V_{out}$, $Zero$, V_{out} with $V_{out} < \frac{1}{2}V_{in}$, in which RP is S1), and in Mode 2, RP turns off during the two part open stage (e.g., $V_{in} - V_{out}$, $Zero$, V_{out} with $V_{in} > V_{out} > \frac{1}{2}V_{in}$, in which RP is S4). Mode 1 corresponds to Figure 7-2 illustrates the switch waveforms in Mode 1, and Figure 7-3 illustrates the switch waveforms in Mode 2.

The above configuration can easily be used with the $V_{in} - V_{out}$, $Zero$, V_{out} switching sequence with both $V_{out} < \frac{1}{2}V_{in}$ and $V_{out} > \frac{1}{2}V_{in}$. When $V_{out} < \frac{1}{2}V_{in}$, the gate signals should be in Mode 1, and when $V_{out} > \frac{1}{2}V_{in}$ the gate signals should be in Mode 2. This is because the two half bridges exchange regulatory roles across this boundary. Table 7.2 defines what role each physical switch takes when using this switching sequence. We chose to connect each ePWM to the correspondingly numbered physical switch for simplicity. Because S1 always turns on during the two-part open stage, ePWM1 can be configured to always output its sync signal when its counter equals 0, ensuring proper switch alignment.

The ePWM register values can be found in Table 7.3 for Mode 1 and in Table 7.4 for Mode 2. As mentioned earlier, CMPA is configured with the switch on-time. Phase is configured to offset the switch turn-on points from each other appropriately.

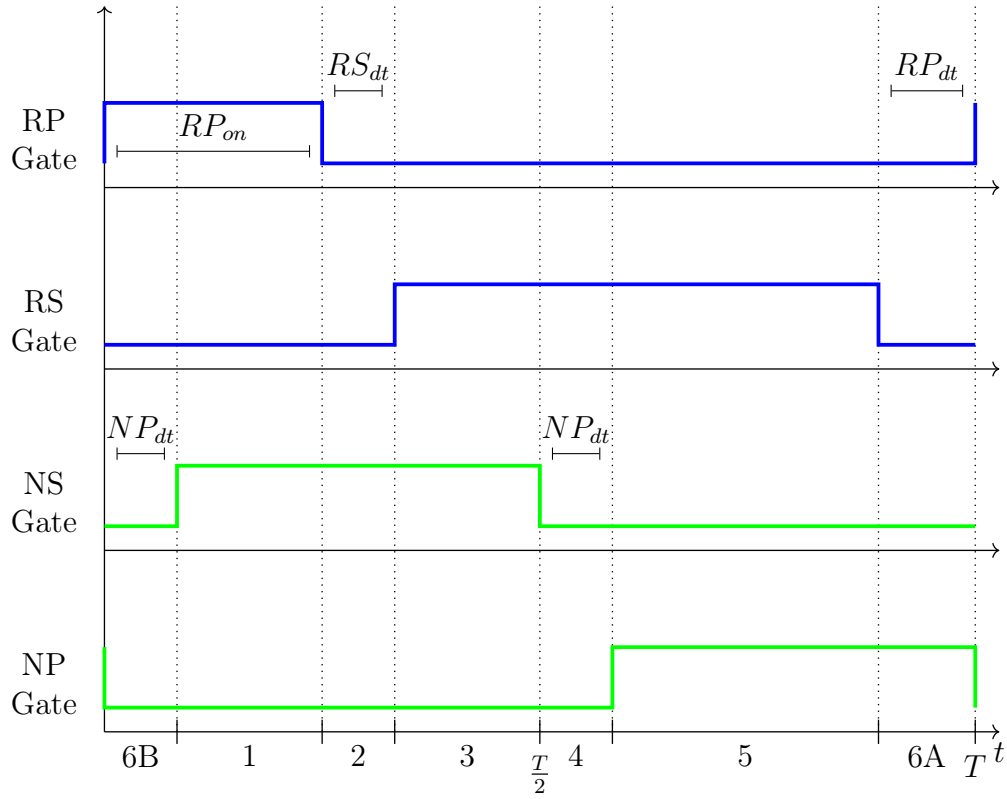


Figure 7-2: Static Mode 1 Switch Waveforms. Used with $V_{in} - V_{out}$, $Zero$, V_{out} with $V_{out} < \frac{1}{2}V_{in}$. The switch transition between stages 6A and 6B during the two-part open stage occurs at the left and right edges of the plot.

Switch	CMPA	Phase
RP	RP_{on}	0
RS	$T - RP_{on} - RP_{dt} - RS_{dt}$	$T - RP_{on} - RS_{dt}$
NP	$T/2 - NP_{dt}$	$T/2 - NP_{dt}$
NS	$T/2 - NP_{dt}$	$T - NP_{dt}$

Table 7.3: Static Control ePWM Mode 1 Register Configurations

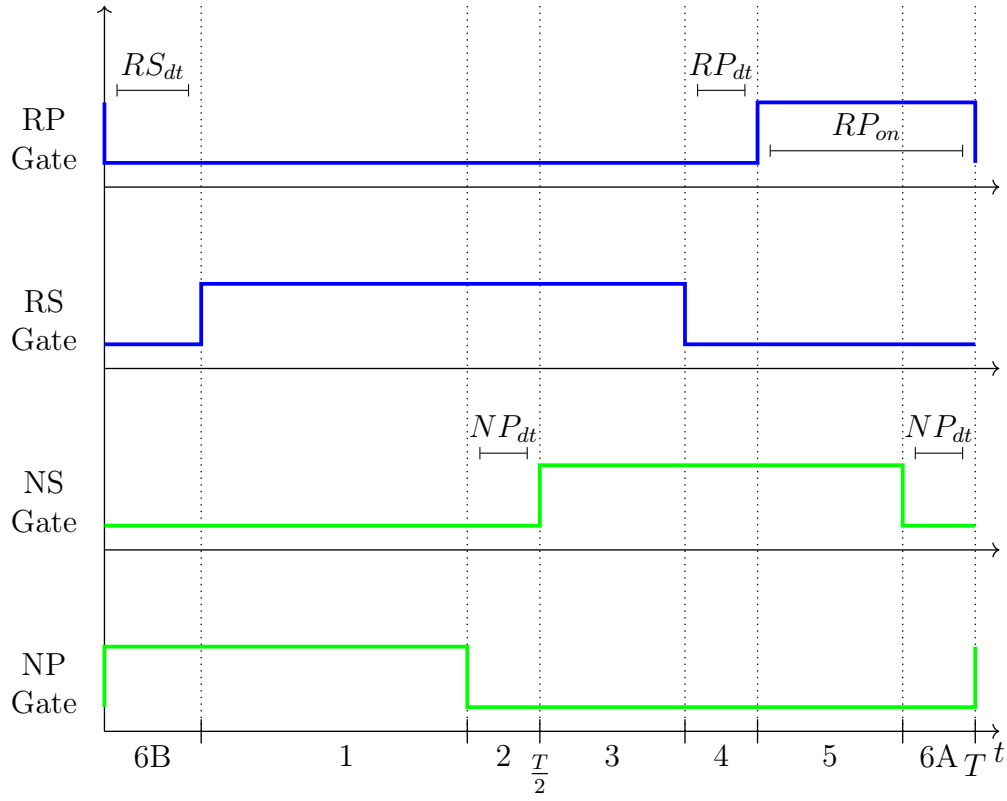


Figure 7-3: Static Mode 2 Switch Waveforms. Used with $V_{in} - V_{out}$, $Zero$, V_{out} with $V_{out} > \frac{1}{2}V_{in}$. The switch transition between stages 6A and 6B during the two-part open stage occurs at the left and right edges of the plot.

If the switch is supposed to turn on x amount of time after the two-part open stage switch transition, then phase should be set to $T - x$. This is because when the sync occurs, Phase is loaded into the counter. The counter will then count up to T after $T - (T - x) = x$ time, causing the counter to reset to 0 and the output to be set high.

The sync mechanism is primarily used to maintain the relative timings of all of the ePWMs. No external syncing is used or required. Since sync signals can only be passed from lower numbered ePWMs to higher numbered ePWMs, ePWM1 is used

Switch	CMPA	Phase
RP	RP_{on}	RP_{on}
RS	$T - RP_{on} - RP_{dt} - RS_{dt}$	$T - RS_{dt}$
NP	$T/2 - NP_{dt}$	0
NS	$T/2 - NP_{dt}$	$T/2$

Table 7.4: Static Control ePWM Mode 2 Register Configurations

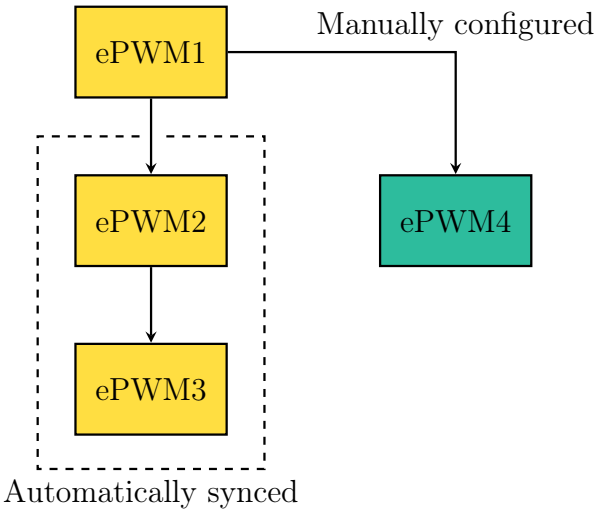


Figure 7-4: Static Sync Diagram

as the sync signal generator. Due to the way the ePWM modules were designed, ePWMs 2 and 3 automatically accept the sync signal from ePWM1, and ePWM4 needs to be specifically configured to respond to ePWM1's sync signal. To ensure proper alignment with the Phase configuration above, ePWM1 should be configured to output its sync signal exactly when the switch transition during the two-part open stage occurs. The sync signal configuration can also be seen in Figure 7-4.

7.2.2 Sensed Control Configuration

Sensed control uses the counter compare actions found in Table 7.5 for all ePWM modules. Similarly to static control, the on-time of switches is configured by setting the CMPA register. However, the output will only be set to high upon a sync event; the output will remain low if no sync event occurs. The Phase register is always set to 0, so that sync events reset the counter to 0. This allows the ePWM modules to act in a "one-shot" mode, where they turn on for a fixed on-time upon a sync event. By configuring the sync events to be triggered by other hardware, sensed switch turn on can be achieved. Figure 7-5 illustrates this sync signal configuration.

Sensed control can be used with the $V_{in} - V_{out}$, $Zero$, V_{out} switching sequence only with $V_{out} < \frac{1}{2}V_{in}$. S3 and S4 must also be implemented with diodes. Table 7.6 defines

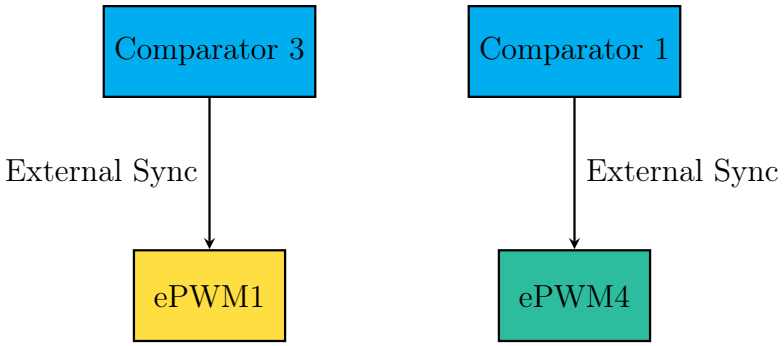


Figure 7-5: Sensed Sync Diagram

Counter Compare	Action
CTR = PERIOD	Set Output Low
CTR = CMPA	Set Output Low
Sync Event	Set Output High

Table 7.5: Sensed Control ePWM Counter Compare Actions

what role each physical switch takes when using this switching sequence.

Because the ePWMs are independent and triggered on-the-fly in sensed control, some additional considerations are required. First, protection needs to be added to prevent the ePWMs from both triggering at the same time, causing switch shoot-through. This can be accomplished using ePWM TripZone features to shut off the ePWMs if shoot-through occurs and the microcontroller's Configurable Logic Block to enforce proper switch triggering. Additionally, the ADC measurement required for ZVS control requires a feed-forward timing prediction. This can be accomplished by measuring the current switching period and predicting that the period will remain nearly constant between consecutive cycles. This method can be subject to errors from noisy triggering and during transients.

Conceptual Switch	Physical Switch ($V_{out} < 1/2V_{in}$)
RP	ePWM1 (S1)
RS	ePWM2 (S2)

Table 7.6: Sensed control switch functions for the $V_{in} - V_{out}$, $Zero$, V_{out} sequence.

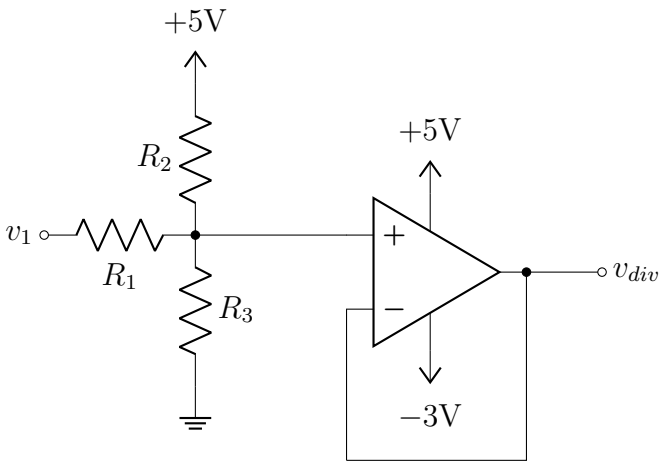


Figure 7-6: Sensing buffer circuitry, implemented with a TL974IN op amp. See Table 7.7 for component values.

7.3 Sensing Implementation

The microcontroller also needs to know the current state of the PR converter to implement the full feedback loop. To accomplish this, the three main outputs from the PR converter, v_{out}, v_{p1}, and v_{p2} need to be scaled down to the 0V-3.3V range to work properly with the ADCs and comparators on the microcontroller.

7.3.1 Buffer Circuitry

It is important that the buffer circuitry has minimal impact on the operation of the PR converter while still being powerful enough to drive the capacitances of the ADCs and comparators. To accomplish this, we use a buffer-connected op-amp driven by a resistive divider to scale down the voltages from the PR converter. The positive supply of the op-amp is +5V, and the negative supply of the op-amp is -3V. The resistive divider was originally designed as a three way resistive divider between the input, 0V, and +5V, which adds a fixed offset to the output of the divider. This was intended to avoid saturation of the negative supply rail at 0V, but since the op-amp saturated regardless, the negative supply was lowered to -3V, meaning the modified resistive divider is no longer necessary. The circuit diagram of the buffer circuit can be seen in Figure 7-6.

Component	Value
R_1	180kΩ
R_2	100kΩ
R_3	12kΩ
$R_{LP,1}$	15kΩ
$R_{LP,2}$	9.1kΩ
$C_{LP,1}$	1000pF
$C_{LP,2}$	470pF

Table 7.7: Component values used in the sensing buffer and low-pass filter circuitry.

The op-amp used is the Texas Instruments TL974IN op-amp, which has a gain-bandwidth product of 12MHz. This op-amp has a low output impedance suitable for driving the microcontroller ADCs and is capable of effectively buffering the v_{p1} and v_{p2} waveforms up to 500 kHz.

The output of the resistive divider will be of the following form:

$$v_{div} = \left(\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3}\right)^{-1} \left(\frac{v_1}{R_1} + \frac{5\text{V}}{R_2}\right) \tag{7.1}$$

If the resistive connection to $V_2 = +5$V is omitted, then the formula simplifies to

$$v_{div} = \frac{R_3}{R_1 + R_3} v_1 \tag{7.2}$$

The sensing circuitry was initially designed to work with a maximum voltage of 50V. The resistor values used can be found in Table 7.7. The same resistor dividers were used for v_{out}, v_{p1}, and v_{p2} so that they would have identical scales from the perspective of the microcontroller. These resistor values give the following gain equation:

$$V_{div} = 0.0561 v_1 + .505\text{V} \tag{7.3}$$

These resistor values give a minimum voltage of .505V and a maximum of 3.31V when $V_1 = 50$V. Additionally, the magnitudes of the resistors were made large enough so that the power dissipation from the PR converter would be negligible. The power drawn from each switch node is the following:

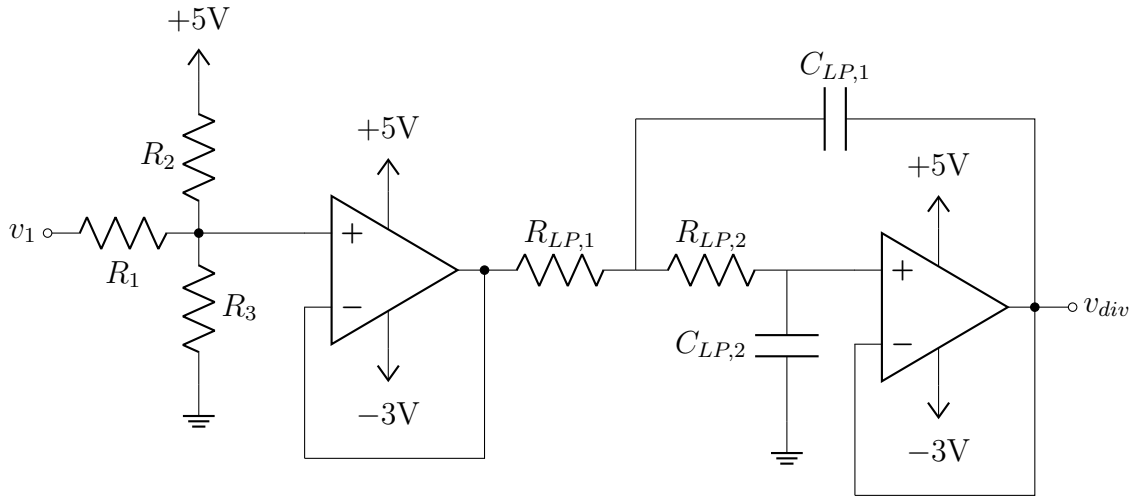


Figure 7-7: Sensing buffer circuitry with low-pass filter, implemented with a TL974IN op amp. See Table 7.7 for component values.

$$P_{diss} = v_1 \frac{v_1 - v_{div}}{R_1} \tag{7.4}$$

The maximum instantaneous power draw with $V_1 = 50$V is 13mW. Actual average power draw can be computed from the v_{p1} and v_{p2} waveforms and Equation 7.4. Average power draw for 30V-18V operation is approximately 4mW.

The output of the buffer for v_{out} is further filtered to remove high frequency noise, including switching ripple. A Sallen-Key low pass filter topology was used. This topology is an active, second order filter, and it was designed to have a cutoff frequency of 19.9kHz and a quality factor of $\frac{1}{\sqrt{2}}$. This cutoff frequency is well below the minimum switching frequency we used, or about 75kHz. A schematic of the low pass filter buffer circuit can be seen in Figure 7-7, and the component values can be found in Table 7.7.

Three sensing buffer circuits were implemented on the microcontroller docking station protoboard. The buffer circuit for v_{out} uses the low-pass filter buffer, while the buffer circuits for v_{p1} and v_{p2} do not. Figures 7-8 and 7-9 show images of the front and back of the docking station, respectively.

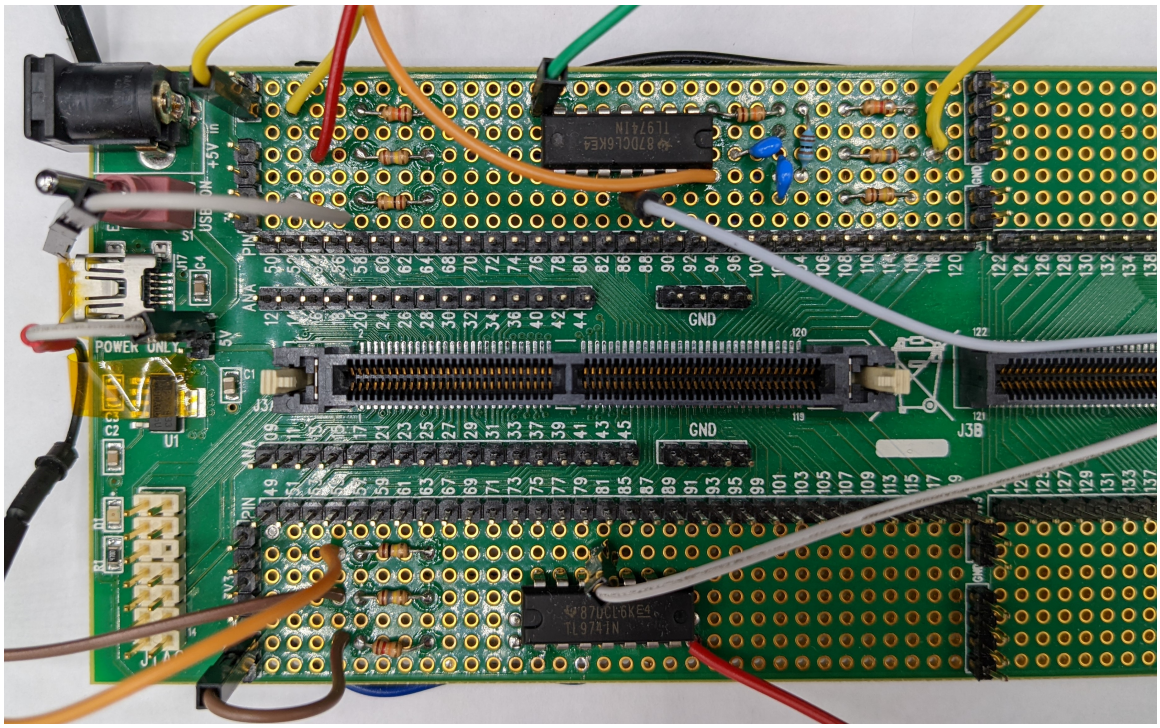


Figure 7-8: Picture of the sensing circuitry, front side.

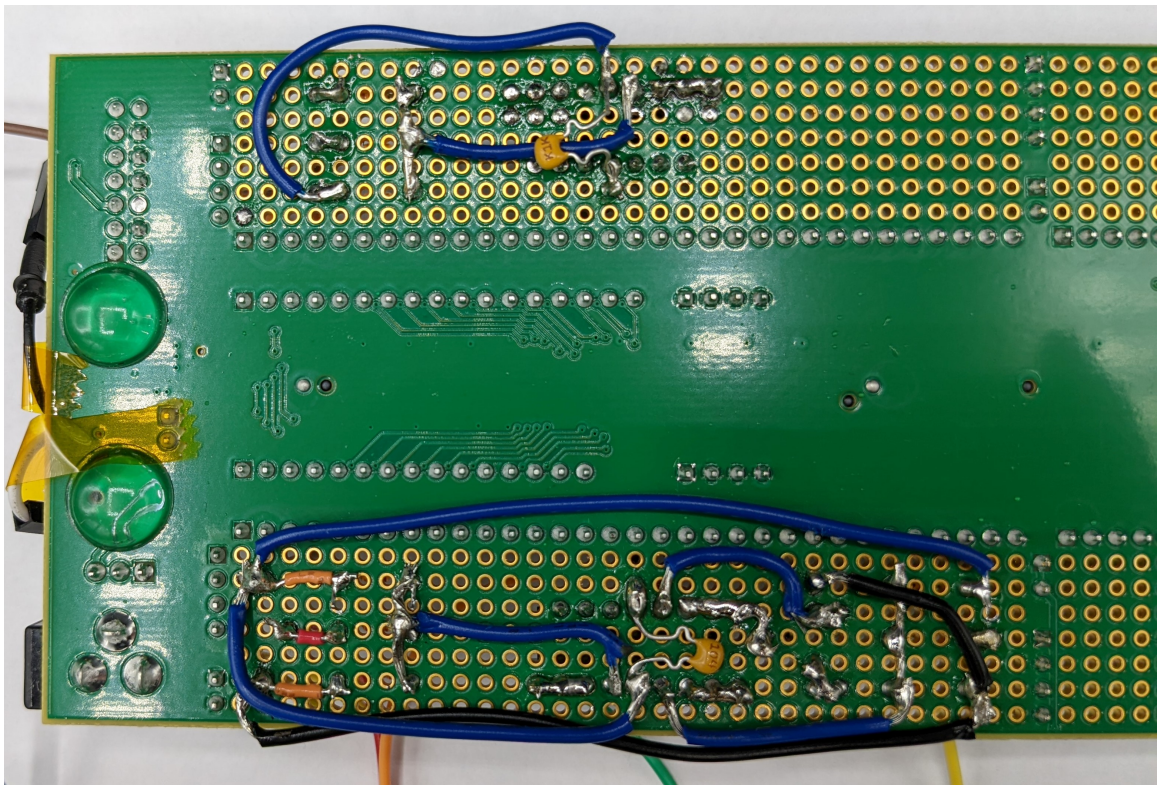


Figure 7-9: Picture of the sensing circuitry, back side.

7.3.2 ADC and Comparator Configuration

As part of the microcontroller's analog subsystem, there are four ADCs and eight comparator units. The ADCs and comparators are connected so that an ADC and a comparator can both read from the same pin simultaneously. The outputs of the three buffers for v_{out}, v_{p1}, and v_{p2} are connected to an ADC and comparator in this fashion, though the comparators are only actively used for v_{p1} and v_{p2}.

As described in Section 7.2, ADC conversions are automatically triggered at specific points within the switching cycle. To implement the ZVS correction feedback loops, the ADCs are used to measure v_{p1} and v_{p2} "just before" the switches turn on. Here, we can more precisely define "just before" by starting the conversion such that the sample and hold interval will complete one clock cycle before the ePWM turns its switch on. The measurement of V_{out} is configured to occur at the same "just before" S1 turns on used to correct for ZVS of S1. It should be noted that where exactly the measurement of v_{out} is made is arbitrary since the waveform is heavily filtered. The microcontroller manual indicates that, when using multiple ADCs simultaneously, higher performance is achieved when the ADC operations overlap exactly. For this reason, v_{out}'s ADC is configured to overlap exactly with another measurement.

The comparators are used to quickly respond to the v_{p1} and v_{p2} waveforms reaching certain points. The comparator outputs are used with the Zero-Crossing Detector in static control and to trigger switch turn-ons in sensed control. The positive terminals of the comparators are connected to v_{p1} and v_{p2}, and the negative terminals are configured to use the internal comparator DACs. These DACs allow the comparison voltage to be set internally with software, allowing for easy modification of the comparator functions. Additionally, the comparators are configured to have hysteresis and a digital filter. The digital filter will only change the output if a certain threshold of comparator samples are a high or low. To balance between speed and noise elimination, we set the digital filter to use the threshold of two out of the last three samples.

7.4 Zero Crossing Detector

As described in Chapter 4, the zero crossing detector (ZCD) needs to measure the width of the two part open stage (t_β) and the point where the switch transition occurs (t_α). A robust method to measure these two quantities is necessary for effective frequency control of the PR converter. Since the time instances can all be represented with either a comparator rising edge or and ePWM counter compare signal, we can use the microcontroller's Configurable Logic Block (CLB) to implement the ZCD. The CLB can be used to implement custom but quite limited digital logic functions. There are four CLB "tiles," and each has three counter modules, three finite state machine (FSM) modules, three input lookup tables (LUTs), and 8 output lookup tables. Additionally, there is a high level controller (HLC) module which can be programmed with up to 8 instructions which run after certain events are triggered. The precise CLB configuration used to implement the ZCD can be found in Appendix H.

To implement the ZCD, two counter modules are used as timers, and the finite state machine modules are used to control the inputs to the counters. The counters will increment their count every clock cycle where they are enabled. To measure the time interval between two events, we use the finite state machines to enable the counters when one event occurs then disable the counters when the next occurs. Finally, the HLC will copy the final counts from the counter modules and place them in code-accesible registers. A block diagram of the system can be seen in Figure 7-10.

There are five inputs to the ZCD:

1. Start Pulse
2. α Pulse
3. β Pulse
4. Reset Pulse
5. Latch Pulse

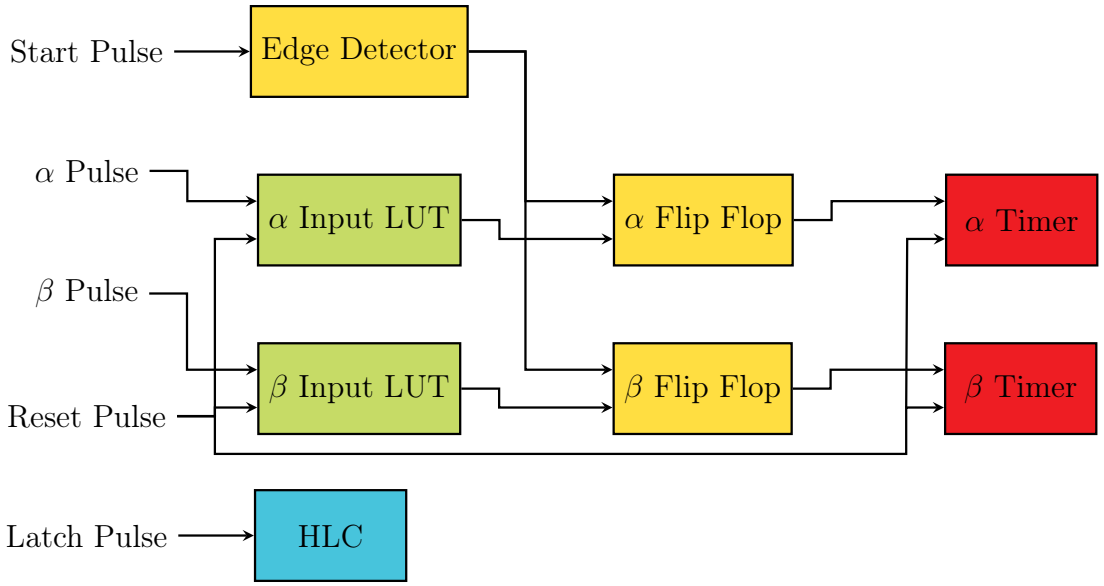


Figure 7-10: Block diagram describing the implementation of the ZCD using the CLB.

Each pulse either comes from a comparator output or an ePWM counter compare. These signals can be internally configured as inputs to then CLB. Comparator output signals are necessary for determining when v_p crosses a certain voltage threshold, and will remain high for some time after triggering. ePWM counter compare pulses are used to determine the time when a switch either turns on or turns off, and are logically high for exactly one clock cycle.

There are two outputs: t_α and t_β. t_α is defined as the time between the start pulse and the α pulse, and similarly t_β is defined as the time between the start pulse and the β pulse. These are both measured in units of 10ns.

There are three main components of the ZCD: the input logic, the counting logic, and the output logic. The input logic consists of an edge detector, implemented with FSM 0, and two logical OR gates, each implemented with an input LUT. FSM 0 must be used because the CLB has hardware limitations to prevent internal feedback loops, and the counting logic must have access to the outputs of the input logic. The edge detector is used to filter the start pulse, preventing the counters from erroneously resuming after an α or β pulse. This issue only affects the start pulse, and it can only occur when the start pulse is driven by a comparator. Through the α and β input LUTs, the α and β pulses are logically ORed with the reset pulse, ensuring the

counting logic is fully turned off in the case where the α or β pulse does not occur.

The counting logic consists of two logical flip flops, implemented using the two remaining FSMs, and two timers, implemented using counter modules. One pair of flip flop and timer measures t_α, and the other measures t_β, and each pair is identical. Each flip flop has two inputs and 2 possible states, on and off. The α flip flop enters the on state when it receives a logical high from the edge-filtered start pulse, and enters the off state when it receives a logical high from the α LUT. The β flip flop behaves similarly, except it responds to the β LUT. The counters are configured in count-up mode and are enabled when their respective flip flops are in the on state. This means the counters will increment every clock cycle that the flip flops are on, effectively timing the intervals between a start pulse and an α orβ pulse. Additionally, the counters load 0 into their accumulators upon a reset pulse.

The output logic uses the HLC to copy the current values stored in the timer registers to the code-accessible registers. The HLC does this as a response to an event triggered by the latch pulse.

The desired order of pulses and operation of the ZCD is as follows:

1. The start pulse occurs. Both flip flops turn on, causing both counters to start incrementing every clock cycle.

2. The α Pulse occurs. The α flip flop turns off, causing the α counter to stop incrementing, retaining its current value.

3. The β Pulse occurs. The β flip flop turns off, causing the β counter to stop incrementing, retaining its current value. At this point, measurement is complete.

4. The latch pulse occurs. The HLC copies the current counts from both counters to the code accessible registers, storing the measurement.

5. The reset pulse occurs. Both flip flops are set to off and both counters are reset to 0.

Pulse	Standard Control	Synchronous Control
Start	$v_{p1} > V_{in} - v_{out}$ (CMPSS3)	$v_{p1} > V_{in} - v_{out}$ (CMPSS3)
α	S1 Turn On (ePWM1 CTR=Zero)	S1 Turn On (ePWM1 CTR=Zero)
β	$v_{p2} > v_{out}$ (CMPSS1)	S3 Turn On (ePWM3 CTR=Zero)
Reset	S2 Turn Off (ePWM2 CTR=CMPA)	S2 Turn Off (ePWM2 CTR=CMPA)
Latch	S1 Turn Off (ePWM1 CTR=CMPA)	S1 Turn Off (ePWM1 CTR=CMPA)

Table 7.8: List of ZCD configurations for the $V_{in} - V_{out}$, $Zero$, V_{out} switching sequence in the $\frac{1}{2}V_{in} > V_{out} > 0$ operating region, both with and without synchronous rectifier control.

Figure 7-11 shows a graphical representation of how the ZCD responds to input signals. By convention, the α pulse occurs before the β pulse, though the operation would be identical if the β pulse occurs first.

7.4.1 Example Implementation

We will use the $V_{in} - V_{out}$, $Zero$, V_{out} switching sequence in both operating regions ($\frac{1}{2}V_{in} > V_{out} > 0$ and $V_{in} > V_{out} > \frac{1}{2}V_{in}$) to describe how the ZCD can be appropriately configured. We assume the topology and hardware configuration described in Section 7.2.1. Recall that the purpose of the ZCD is to control the switch transition within the two-part open stage to line up exactly with the corresponding i_L zero crossing.

For the $\frac{1}{2}V_{in} > V_{out} > 0$ region, the two part open stage will have the form seen in Figure 7-12 and the pulse configuration seen in Table 7.8. The minimum voltage common to both halves of the open stage is $V_{in} - v_{out}$, so we measure around that voltage level. The starting point is when v_{p1} reaches $V_{in} - v_{out}$, and we need a comparator to create the start pulse since this point is not tied to a switch transition. CMPSS3 is configured to compare v_{p1} with $V_{in} - v_{out}$. The α pulse occurs when S1 turns on, which is when ePWM1's counter is 0.

The β pulse occurs when v_{p2} reaches v_{out}, which is also when S3 turns on. When

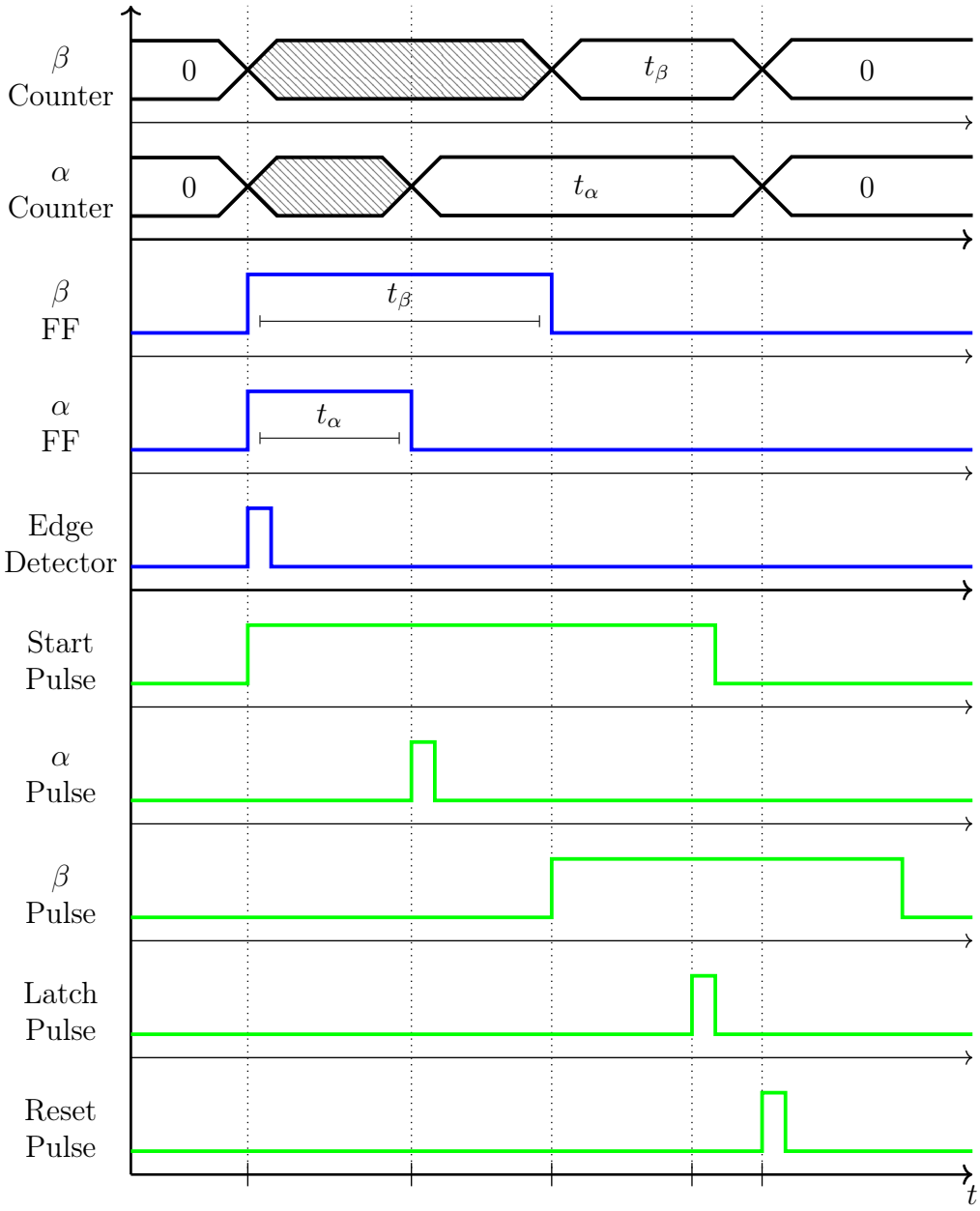


Figure 7-11: ZCD Timing Diagram

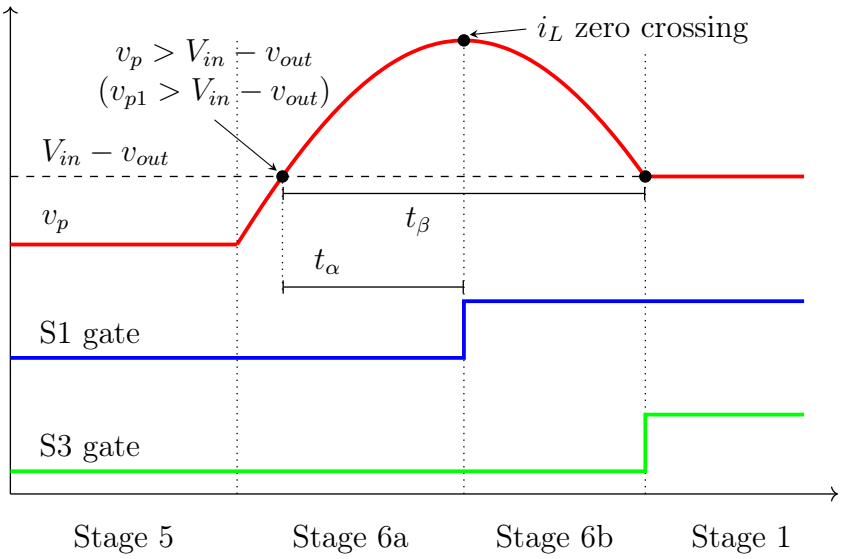


Figure 7-12: ZC Waveform for $V_{out} < 1/2V_{in}$. Plot illustrating how the i_L zero crossing can be detected by observing symmetry in v_p. In this example, S1's turn off is exactly aligned with the zero crossing, so we have $t_\alpha = \frac{1}{2}t_\beta$.

using diodes at S3 and S4, this point is not immediately known by the controller, so a comparator must be used. CMPSS1 is configured to compare v_{p2} with v_{out}. However, when using synchronous rectifier control, S3's turn on is set by ePWM3, so we can use when ePWM3's counter is 0 to determine the β pulse. Care should be taken with the measured data if the ZVS error at S3's turn on is not close to 0, as this means v_{p2} is being hard-switched instead of resonating properly, and the ZCD symmetry assumption is no longer valid.

The exact configuration of the latch and reset signals is not as critical as the start, α, and β pulses. The only requirements are that the latch pulse occurs after the β pulse, the reset pulse occurs after the latch pulse, and the reset pulse occurs before the next start pulse. Under standard operation, S1 can be safely assumed to turn off after the β pulse, and afterwards, S2 must turn off before v_{p2} can rise above $V_{in} - v_{out}$.

The $V_{in} > v_{out} > \frac{1}{2}V_{in}$ region is similar to the $\frac{1}{2}V_{in} > v_{out} > 0$ region, except that the two part open stage is effectively mirrored horizontally. This can be seen in Figure 7-13, and the pulse configuration can be seen in Table 7.9. Now, the minimum common voltage is v_{out} and the start pulse is generated when S2 turns off, or when ePWM2's counter is CMPA. The α pulse still occurs when S1 turns on, or when

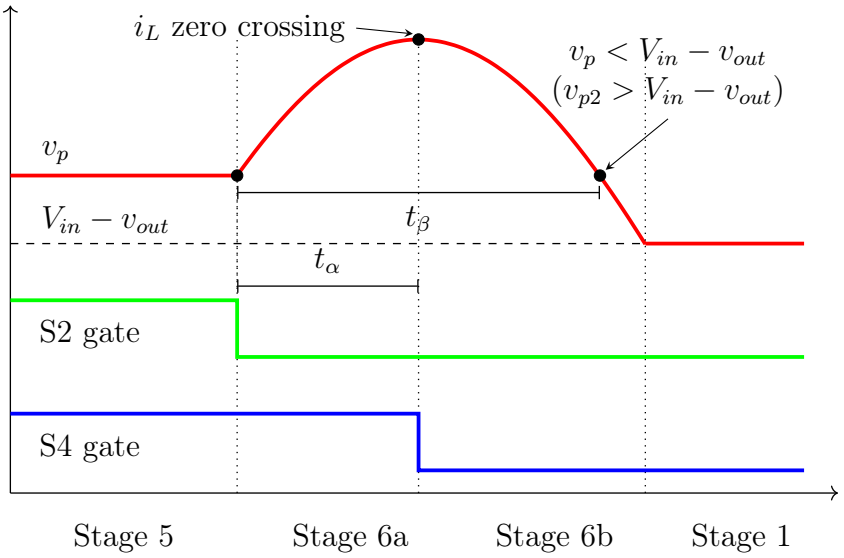


Figure 7-13: ZC Waveform for $V_{out} > 1/2V_{in}$. Plot illustrating how the i_L zero crossing can be detected by observing symmetry in v_p. In this example, S4's turn off (also S1's turn on) is exactly aligned with the zero crossing, so we have $t_\alpha = \frac{1}{2}t_\beta$.

Pulse	Synchronous Control
Start	S2 Turn Off (ePWM2 CTR=CMPA)
α	S4 Turn Off (ePWM1 CTR=Zero)
β	$v_{p2} > V_{in} - v_{out}$ (CMPSS1)
Reset	S4 Turn On (ePWM4 CTR=Zero)
Latch	S3 Turn On (ePWM3 CTR=Zero)

Table 7.9: List of ZCD configurations for the $V_{in} - V_{out}$, $Zero$, V_{out} switching sequence in the $V_{in} > V_{out} > \frac{1}{2}V_{in}$ operating region.

ePWM1's counter is 0. The β pulse occurs when v_{p2} reaches $V_{in} - v_{out}$, which is determined by CMPSS1. The latch pulse is set to when S3 turns on, which is after the beta pulse in standard conditions. Finally, the reset pulse is when S4 turns on, which occurs after the latch and before the start pulse when S4 turns off.

7.5 Code Feedback Loop

To complete the feedback controller, the measurements from the ADCs and the ZCD are used to update the switching times. Discrete time PI compensators are computed for every controllable switching time, and updated every cycle (or after a fixed amount of cycles, which is necessary at higher frequencies). Every measurement has a desired value, which is either a constant or based on other measurements.

The feedback code is contained within an interrupt routine which is called every PR resonant cycle. The interrupt routine performs the following procedure:

1. Load all of the measurement data out of the ADC and ZCD registers. (Also done while disabled)

2. Update the comparator DACs based on measurement data. (Also done while disabled)

3. Compute the error terms for each measurement.

4. Use the error terms to compute the proportional terms and update the integral terms.

5. Compute the new switching times from the proportional and integral terms.

6. Check to ensure the switching times do not exceed set bounds. Cap the switching times at the bounds if necessary.

7. Reconfigure the ePWM modules with the new switching times.

8. Wait for new trigger from the switching cycle.

First, all of the most recent ADC and ZCD data needs to be accessed to compute the new error terms. This process is straightforward, and only requires invoking library functions to copy the data from hardware registers. The measurement of v_{out} is also used to update the comparator DACs at this point in time, ensuring that the comparators correctly respond to v_{p1} and v_{p2}, even during transients. Correct comparator thresholds are necessary for the ZCD to function properly.

To compute the switching parameters controlled by feedback loops (RP_{on}, RP_{dt}, RS_{dt}, NP_{dt}, and T), we need the corresponding error term, integral term, and feedback coefficients. We can compute the error terms directly from the current ADC and ZCD measurements, along with any necessary constants. The three types of errors are v_{out} error, ZVS error, and zero-crossing offset error. One ADC measures the value of v_{out} every cycle. v_{out} error is the difference between the currently measured v_{out} and the desired v_{out}, which is a constant specified by the user or software. The remaining ADCs measure the values of v_{p1} and v_{p2} just before their switches turn on. ZVS error is the difference between these measured values and the value of the node after the switch turns on. During connected stages, v_{p1} and v_{p2} can be one of three possible voltages: V_{in}, v_{out}, and 0. As such, these are the only desired values for ZVS error. When the desired value is v_{out}, the currently measured value should be used. Finally, the ZCD provides t_α and t_β for computing the zero crossing offset error. This error is simply the difference of $t_\alpha - \frac{1}{2}t_\beta$.

Once we have the error terms, we can use them to compute the new switching parameters for this cycle, and we implement the feedback loop with discrete time PI compensators. The switching parameters have two main terms, the proportional term and the integral term. The proportional term is the product of the proportional feedback coefficient and the current error term. The integral term is the product of the integral feedback coefficient and the total sum of the error from all previous cycles. The current error is then added to the integral term. Thus, the proportional and integral terms can be summed to give the switching time. One consideration is that the ePWM registers support 16 bit integers for all register fields. To have higher precision than just 16 bits when computing switching times in the feedback

loop, we store switching times with signed 32-bit fixed point numbers, where the most significant 16 bits are taken as the integer part and used to program the ePWMs.

The final step is to ensure the computed switching times do not exceed bounds that would prevent forming a valid switching sequence for that cycle. For example, this could occur when there is a transient with large feedback coefficients. Dead times are configured to have minimum (DT_{min}) and maximum values (DT_{max}). These values can be determined empirically by observing dead time lengths during normal operation. We choose a DT_{min} to be about 5% of T and DT_{max} to be about 22% of T. We also bounded RP_{on} to have a minimum of DT_{min} and a maximum of $\frac{T}{2} - DT_{min}$. This ensures that RP does not stay on past the following i_L zero crossing, and that there is always some dead time between RP being turned on and both surrounding i_L zero crossings. If the switching times do exceed the bounds one way or the other, then the bound will be used instead. The integral term will also not be updated when the bound is exceeded.

Once the switching times have been computed, they can be programmed to the ePWMs. The ePWM shadowing feature will ensure the new switching times are all properly loaded at the same time. The interrupt routine will now exit, and the processor will wait for the next switching cycle to occur before running the feedback code again.

7.6 Startup

The feedback control procedures outlined in this thesis assume that the PR converter is already within some "reasonable" state to properly make corrections to its operation. Thus, it is necessary that the startup procedure puts the PR converter into a "reasonable" state. A simple way to accomplish this is to output predetermined switching times initially on startup before enabling the feedback control. Upon power-up, the specific initial switching times are used to configure the ePWMs and the initial integrator values. After a fixed time interval (about 3-5 seconds is practical), the feedback control loop will begin updating the switching times. The initial switching frequency

should be within the inductive region of the PR, or between its series and parallel resonant frequencies, so that ZVS is possible and the ZCD will properly detect the zero crossing location. The initial switching times can be estimated for the specific PR being used using the PSS solutions described in Chapter 3.

Chapter 8

Experimental Results

In this chapter, we explore the rest of the experimental setup that was used to test the feedback controller. Then, we will discuss the experimental results obtained.

8.1 Experimental Setup

To perform experiments, the controller described in Chapter 7 was connected to the PR-based dc-dc converter described in Chapter 6. The converter outputs v_{p1}, v_{p2}, and v_{out} were connected to the inputs of the controller's sensing buffers, and the controller's ePWM outputs were connected to the converter's isolated switch inputs. For the following experiments, we used an APC International part 1553 PR, whose parameters can be found in Table 8.1.

The final component needed to perform experiments is a suitable load for the con-

Parameter	Value
C_p	1.41nF
C_r	510pF
L	8.73mH
R	2.3Ω
Series Frequency	75.4 kHz
Parallel Frequency	88.0 kHz

Table 8.1: Table of PR parameters for the specific APC International Part 1553 PR used during experiments. Parameters were extracted using an impedance analyzer.

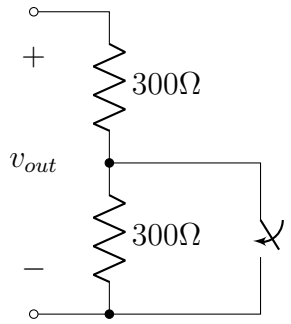


Figure 8-1: Converter load circuitry. The load resistance is 600Ω when the switch is open, and 300Ω when the switch is closed. The switch is implemented as an IRF740 MOSFET.

verter. We used a resistive load because it is a passive device where power dissipation is a function of the output voltage. We also designed the load to be tapped in the center, allowing half of the resistance to shorted out with a switch. We implemented the switch as a discrete through-hole IRF740 MOSFET (note that this part is capable of handling much higher powers than is necessary here, any MOSFET would work), and applying voltage to the gate is used to change the load resistance. When the gate is high (+5V), then the load resistance will be 300Ω, and when the gate is low, the load resistance will be 600Ω. A pulldown resistor ties the gate to ground, preventing the gate from floating when disconnected. A circuit diagram of the load can be found in Figure 8-1, and an image of the load setup can be found in Figure 8-2.

8.2 Experimental Results

We first did an experiment to confirm that the controller is able to meet the desired high efficiency behaviors. We used the switching sequence $V_{in} - V_{out}$, $Zero$, V_{out} with $V_{out} < \frac{1}{2}V_{in}$ and ran the converter with static regulating half-bridge control. The nonregulating half-bridge was controlled passively with diodes at S3 and S4. The feedback coefficients used were small to ensure the most stable waveform, and can be found in Table 8.2. Figure 8-3 shows the PR converter waveforms across several switching cycles. It is clear that the PR is being soft charged since, v_p is able to resonate exactly to the following connected stage voltage during open stages. It is

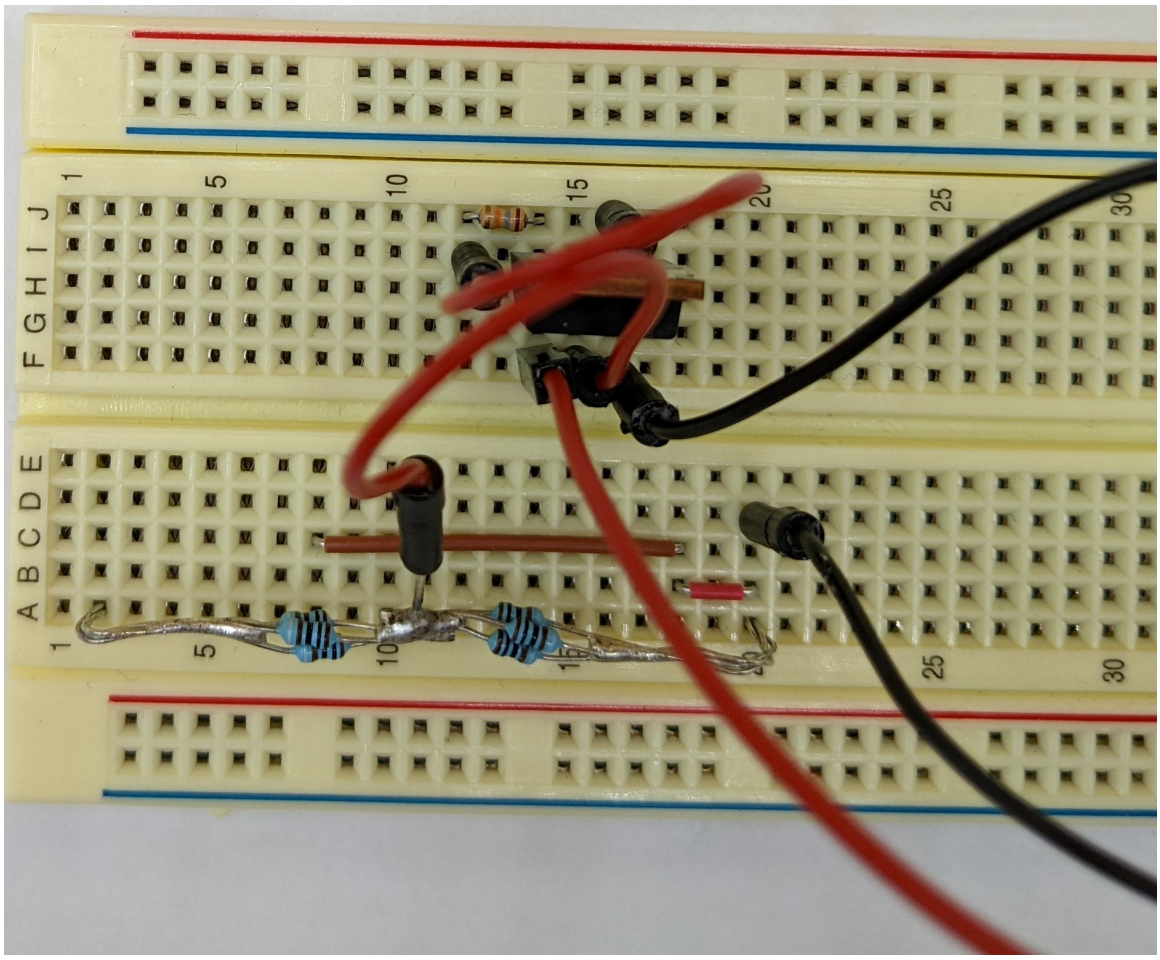


Figure 8-2: Photo of the load circuitry used during experiments.

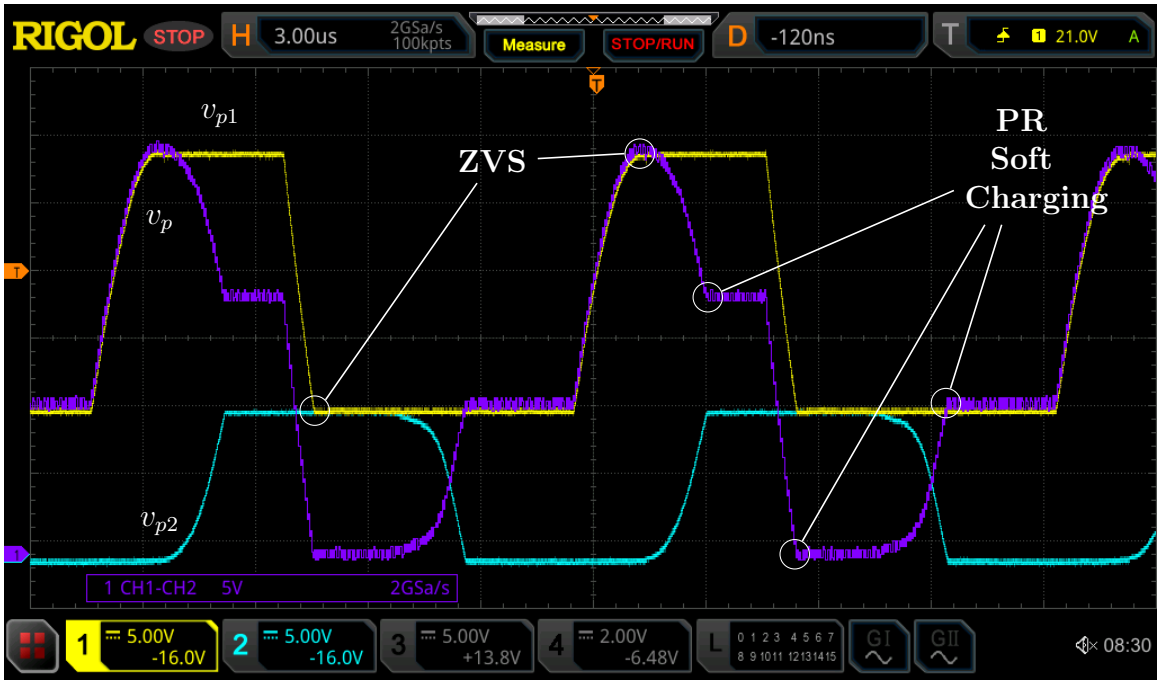


Figure 8-3: Zoomed in view of the PR waveforms v_{p1}, v_{p2}, and v_p, showing that ZVS and soft charging are achieved with the feedback controller active. $V_{in} = 30V$, $v_{out} = 10.4V$, and $R_{load} = 600\Omega$.

Feedback Loop	K_p	K_i
$v_o ut$	0	-10
ZVS (all)	0	10
ZC Offset	0	-10

Table 8.2: Feedback coefficients used to test high efficiency behaviors and nonregulating half-bridge control.

also clear that ZVS is achieved since v_p1 and v_p2 also resonate enough to allow S1 and S2 to turn on with 0 drain-to-source voltage.

Once we knew the regulating half-bridge control was working, we enabled nonregulating half-bridge control to confirm that we could achieve synchronous rectification of S3 and S4, which is illustrated in Figure 8-4. The slight shift in voltage indicates that the MOSFETs in parallel with diodes at S3 and S4 have turned on, and the loss from the diode forward drop is mitigated. The feedback coefficients in Table 8.2 were also used in this experiment.

Next, we tested the response of the converter and controller system after a step in load resistance. As described in Section 8.1, our load circuit allows steps from

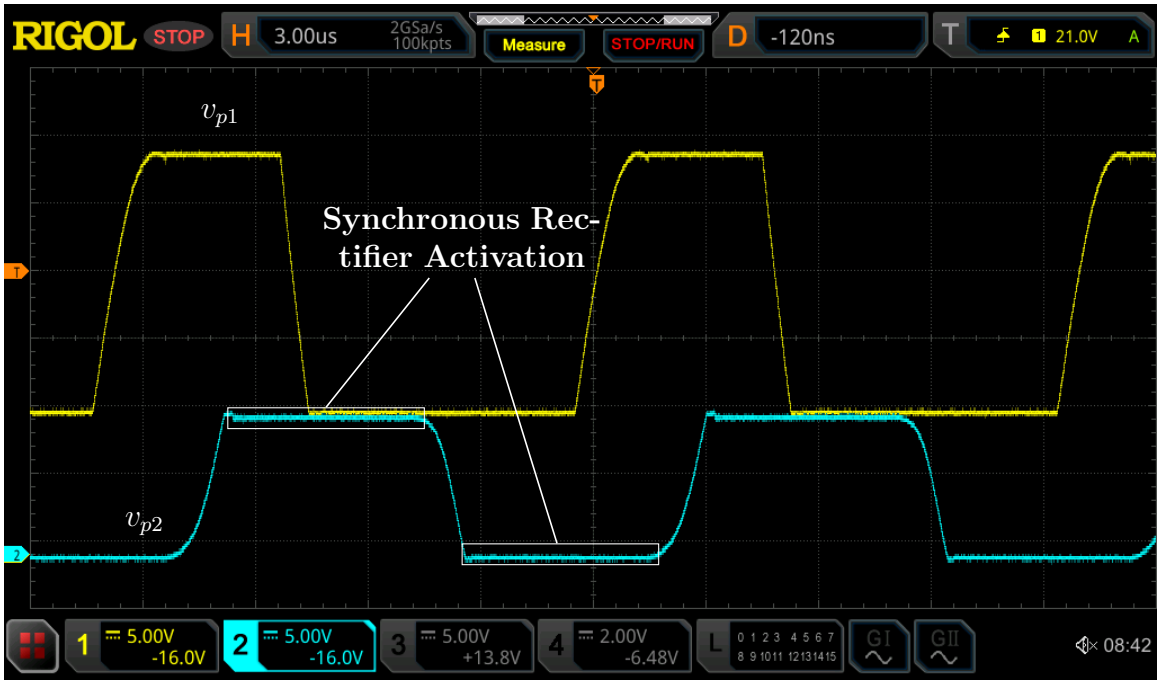


Figure 8-4: Zoomed in view of the PR waveforms v_{p1} and v_{p2} with synchronous rectifier control enabled. $V_{in} = 30V$, $v_{out} = 10.4V$, and $R_{load} = 600\Omega$.

600Ω to 300Ω and from 300Ω to 600Ω. The step was performed by changing the gate voltage v_{gate} of the load MOSFET, the transient waveform was captured on the oscilloscope by configuring it to single trigger off of an edge on v_{gate}. We evaluated the transient response based on the settling time and peak deviation of v_{out}. Our goal was to minimize the settling time, defined as the time taken for v_{out} to settle within 2% of the steady state output voltage, while keeping the peak voltage deviation within 10% of the steady state output voltage.

The response to the 600Ω -> 300Ω step can be seen in Figure 8-5, with a settling time of 14.6ms and a peak deviation of 7.5%. Likewise, the response to the 300Ω -> 600Ω step can be seen in Figure 8-6, with a settling time of 18.4ms and a peak deviation of 5.8%.

The feedback coefficients used for both load resistance step experiments can be found in Table 8.3. These coefficients were selected using the following procedure:

1. Start with known stable feedback coefficients (see Table 8.2).

2. Increase the v_{out} coefficients until control becomes unstable or no improvement

Figure 8-5: Response to R_{load} step from 600Ω to 300Ω with $V_{in} = 30V$ and $V_{out} = 10.4V$. The peak deviation from steady state is $770mV$, or 7.5% of the output voltage. The output voltage settles to within 2% after $14.6ms$.

Figure 8-6: Response to R_{load} step from 300Ω to 600Ω with $V_{in} = 30V$ and $V_{out} = 10.4V$. The peak deviation from steady state is $600mV$, or 5.8% of the output voltage. The output voltage settles to within 2% of steady state after $18.4ms$.

Feedback Loop	K_p	K_i
$v_o ut$	-30000	-40
ZVS (S1)	0	45
ZVS (S2)	0	10
ZC Offset	-30000	-500

Table 8.3: Feedback coefficients used to test transient response after a step in load resistance.

in the transient response is seen.

3. Increase the ZC-offset coefficients until control becomes unstable or no improvement in the transient response is seen.

4. Increase the ZVS coefficients until control becomes unstable or no improvement in the transient response is seen.

5. Repeat until no improvement is seen.

Chapter 9

Conclusion

Creating a feedback control system for PR-based dc-dc power converters that achieves all of the desired high efficiency behaviors is challenging because it requires aspects of duty cycle, dead time, and frequency control. In this thesis, we derive the regulation capabilities of six stage sequences for PR-based converters and derive the regulating and nonregulating half-bridges. We then propose two control methods for the regulating half-bridge, sensed control and static control. We also propose a control method for the nonregulating half bridge for use with static control. We then experimentally validate the proposed control scheme with a prototype PR dc-dc converter and a microcontroller-based feedback controller implementation. We also present several analysis methods and models for the operation of PR-based dc-dc converters, both steady state and dynamic.

The proposed control is advantageous because it was implemented on a microcontroller and only relies on voltage sensing techniques. The implementation uses only simple feedback loops that can be easily computed on a microcontroller, and requires only ADCs and comparators for measurements without having to sense any currents directly. The control also successfully achieves all desired high efficiency behaviors, PR soft charging, ZVS of all switches, and all-instantaneous power transfer, while being capable of regulating to a range of output voltages and responding to load resistance transients.

Piezoelectric resonators are promising alternatives to magnetic energy storage for

miniaturization in power electronics owing to their high quality factors and power density capabilities. The proposed feedback control scheme is simple and robust, and paves the way for enabling use of small and efficient PR-based dc-dc power converters in a wide range of real world applications.

9.1 Future Work

One area for future work is an expansion of the dynamic modeling techniques discussed in Chapter 5. Currently, the modeling methods that do not rely on a direct circuit simulation make use of heavy simplifying assumptions about the feedback controller. While the models have good agreement with each other, more work needs to be done to better model the feedback loops that are present in static control so that more accurate predictions of the dynamic response of the converter on real hardware can be made.

Another important area of future work will be expanding the controller to work with higher frequency PRs. The frequency of the PR used in this thesis is in the 78-85 kHz range, while the frequencies of PRs with high power densities tend to be in the 500kHz to low MHz range. Testing with a larger and low frequency PR is acceptable for initial validation, but some aspects of the controller may need to be changed to get good performance at higher frequencies. First the method used for implementing ZVS feedback loops should be revisited, since ADCs cannot make accurate measurements when the PR waveforms change too quickly. It will be also necessary to ensure that the gate signal generators and the zero-crossing detector have adequate time resolution at high frequencies, since the PR is sensitive to small changes in switching times, especially with the switching period. Finally, the controller should be implemented on a PCB rather than relying on proto-boarded sensing circuitry and jumper wires for connections.

Appendix A

Full Steady State Solutions

A.1 Ideal Steady State Solution Example Equations

This appendix presents the full set of CoC and Coe equations describing a PSS solution for a general six-stage sequence with a two-part open stage. The switching sequence has connected stage voltages V_a, V_b, V_c, where $v_p = V_a$ in stage 1,$v_p = V_b$ in stage 3, and $v_p = V_c$ in stage 5. It must resonate v_p to V_d at the start of stage 6B during the two-part open stage to achieve ZVS. The corner variables are denoted v_{rx} and i_{Lx}, where "x" is the stage number. The PR used has parameters C_p, C_r, and L. i_L zero crossing constraints can be specified by adding the equation $i_{lx} = 0$. If the switching sequence has its two-part open stage in stage 2 or 4, the equations can be modified or the switching sequence can be rotated so the two-part open stage is stage 6.

The following equations implement the PSS solution:

$$C_r(v_{r1} - V_a)^2 + Li_{L1}^2 = C_r(v_{r2} - V_a)^2 + Li_{L2}^2 \tag{A.1}$$

$$C_pV_a^2 + C_rv_{r2}^2 + Li_{L2}^2 = C_pV_b^2 + Cv_{r3}^2 + Li_{L3}^2 \tag{A.2}$$

$$C_p(V_b - V_a) = -C(v_{r3} - v_{r2}) \tag{A.3}$$

$$C_r(v_{r3} - V_z)^2 + Li_{L3}^2 = C_r(v_{r4} - V_b)^2 + Li_{L4}^2 \tag{A.4}$$

$$C_p(V_b)^2 + C_rv_{r4}^2 + Li_{L4}^2 = C_pV_c^2 + C_rv_{r5}^2 + Li_{L5}^2 \tag{A.5}$$

$$C_p(V_c - V_b) = -C_r(v_{r5} - v_{r4}) \tag{A.6}$$

$$C_r(v_{r5} - V_c)^2 + Li_{L5}^2 = C_r(v_{r6a} - V_c)^2 + Li_{L6a}^2 \tag{A.7}$$

$$C_pV_c^2 + C_rv_{r6a}^2 + Li_{L6a}^2 = C_p(V_d)^2 + C_rv_{r6b}^2 + Li_{L6b}^2 \tag{A.8}$$

$$C_p(V_d - V_c) = -C_r(v_{r6b} - v_{r6a}) \tag{A.9}$$

$$C_pV_d^2 + C_rv_{r6b}^2 + Li_{L6b}^2 = C_p(V_a)^2 + C_rv_{r1}^2 + Li_{L1}^2 \tag{A.10}$$

$$C_p(V_a - V_d) = -C_r(v_{r1} - v_{r6b}) \tag{A.11}$$

Appendix B

PR Converter PCB Technical Information

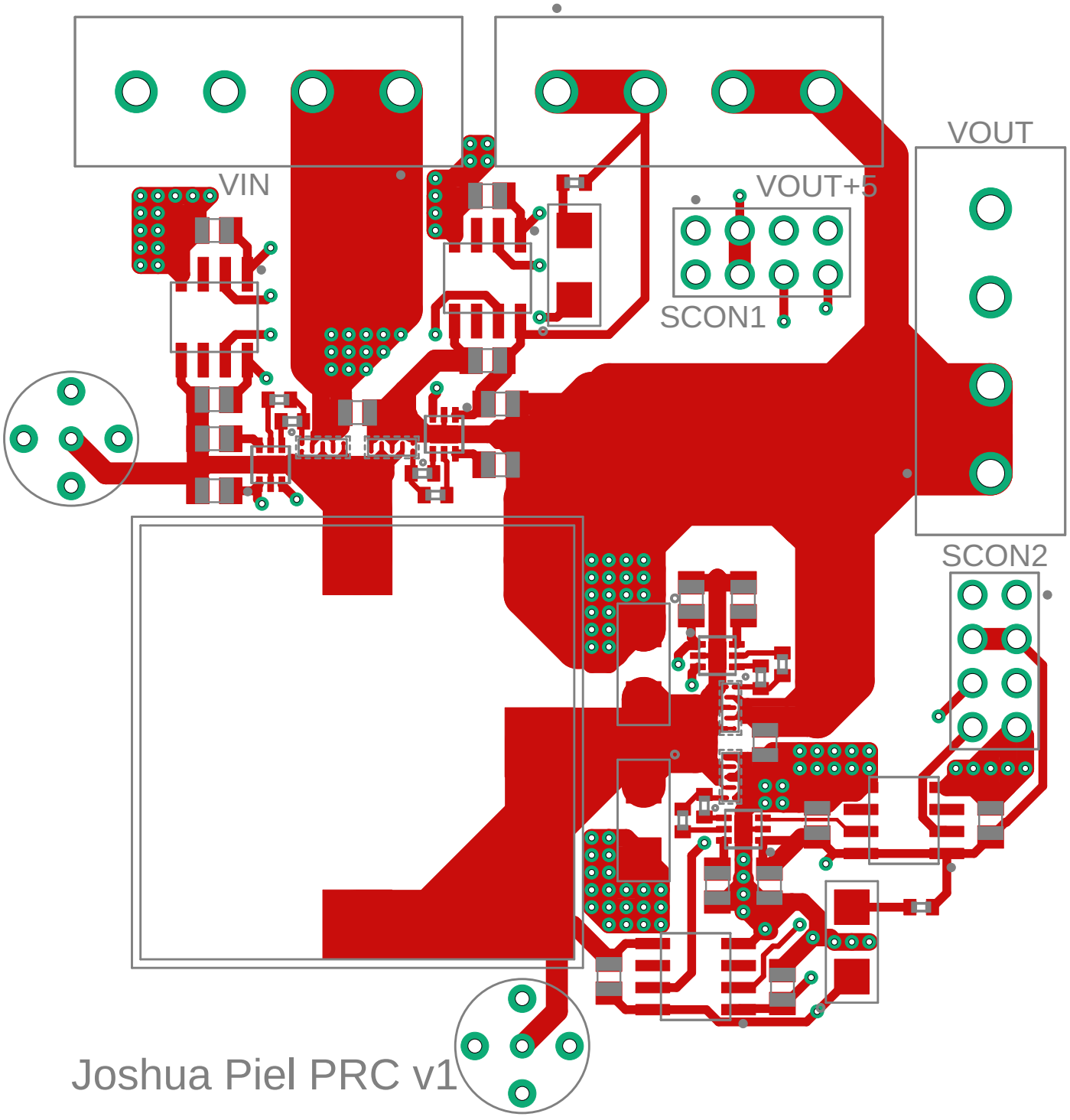
This appendix presents the full details of the PCB layout, schematic, bill of materials, and PCB header pinout for the prototype PR-based dc-dc converter implemented in this thesis.

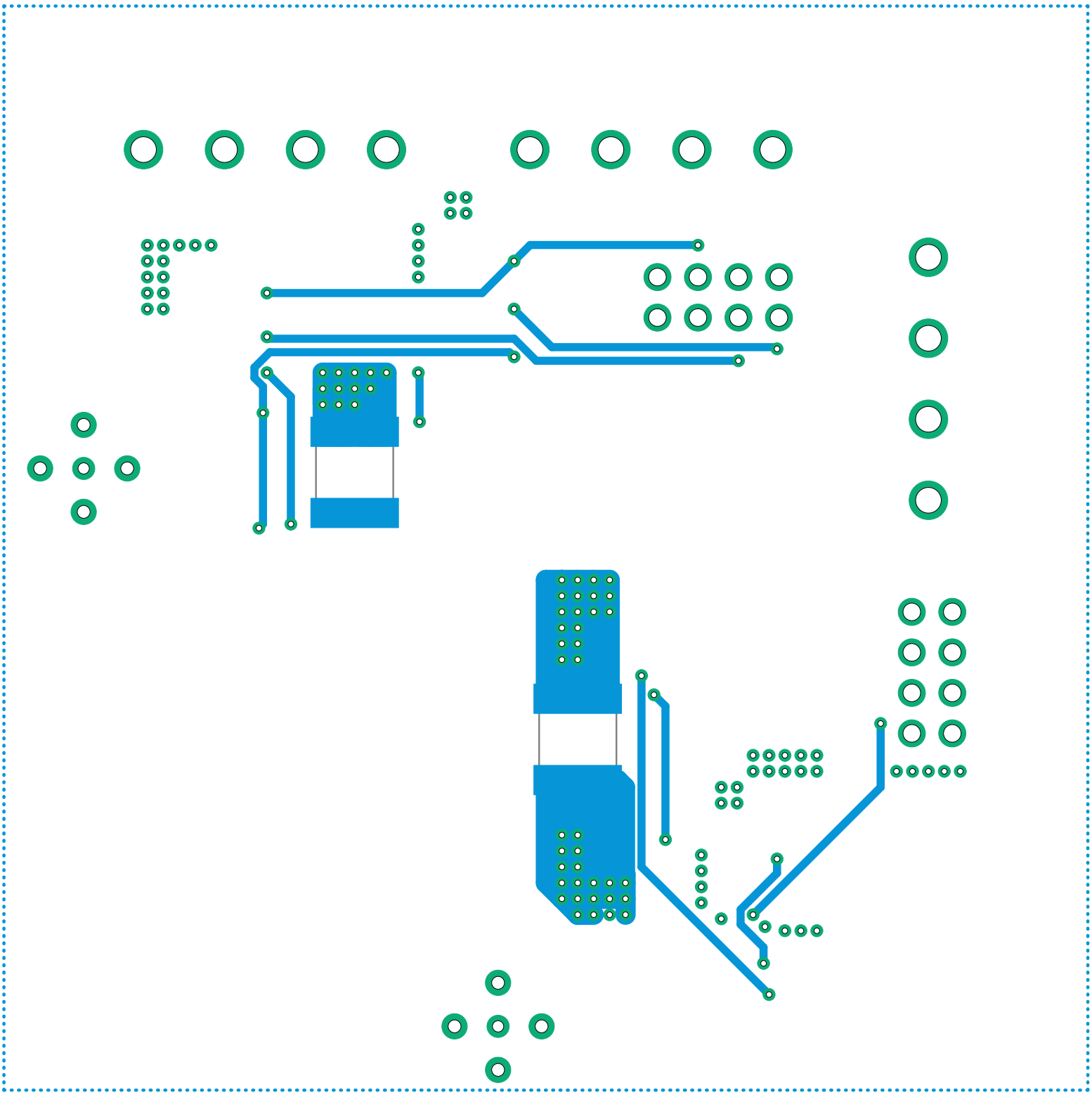
The PCB layout files are at 3x scale. For example, a line with length 3 inches on this document corresponds to a length of 1 inch on the actual PCB. This document has dimensions 8.5 inch by 11 inch.

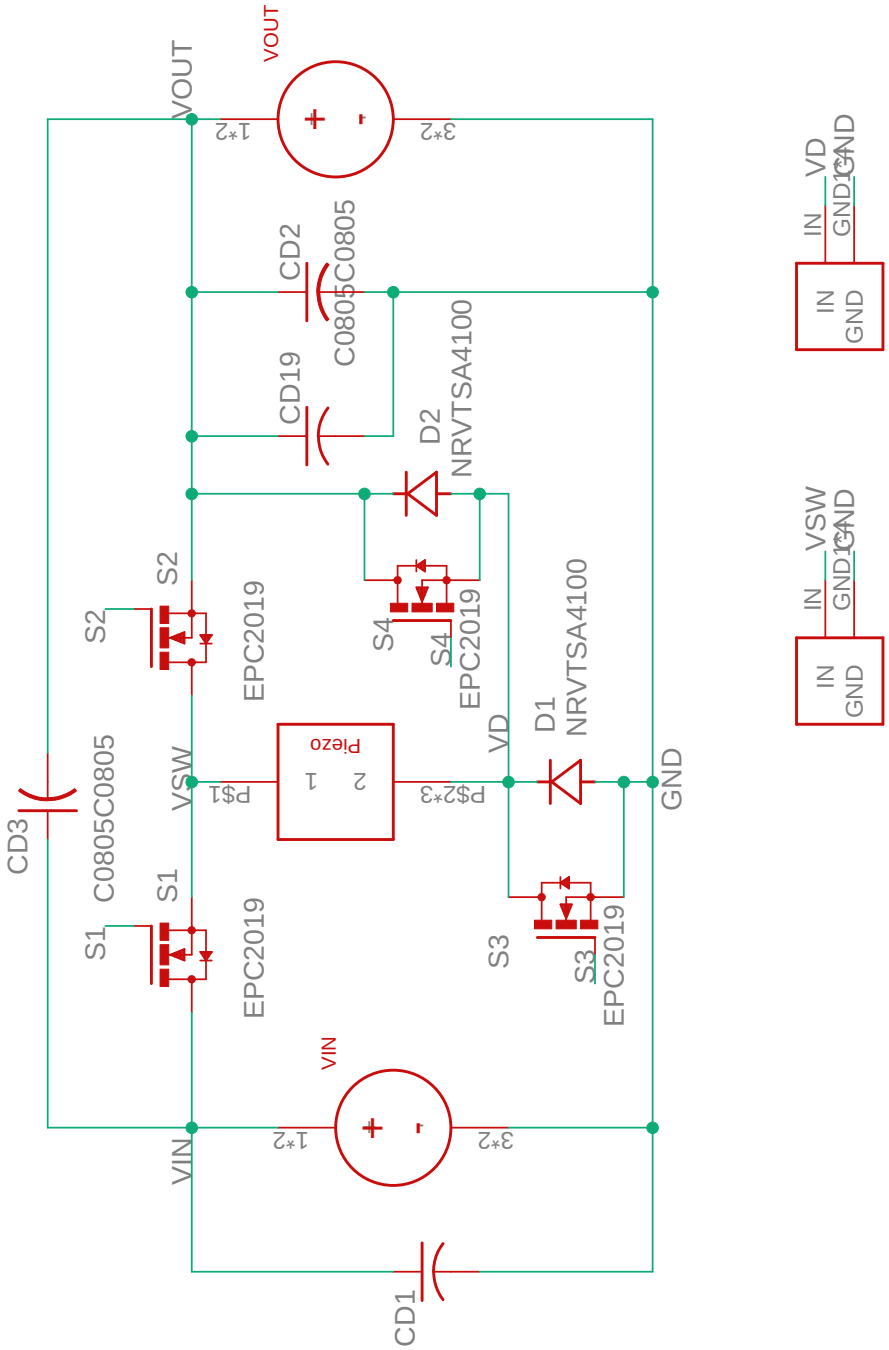
The schematic files show the schematics of both the main power conversion stage as well as all of the gate drive circuitry used to drive the switches.

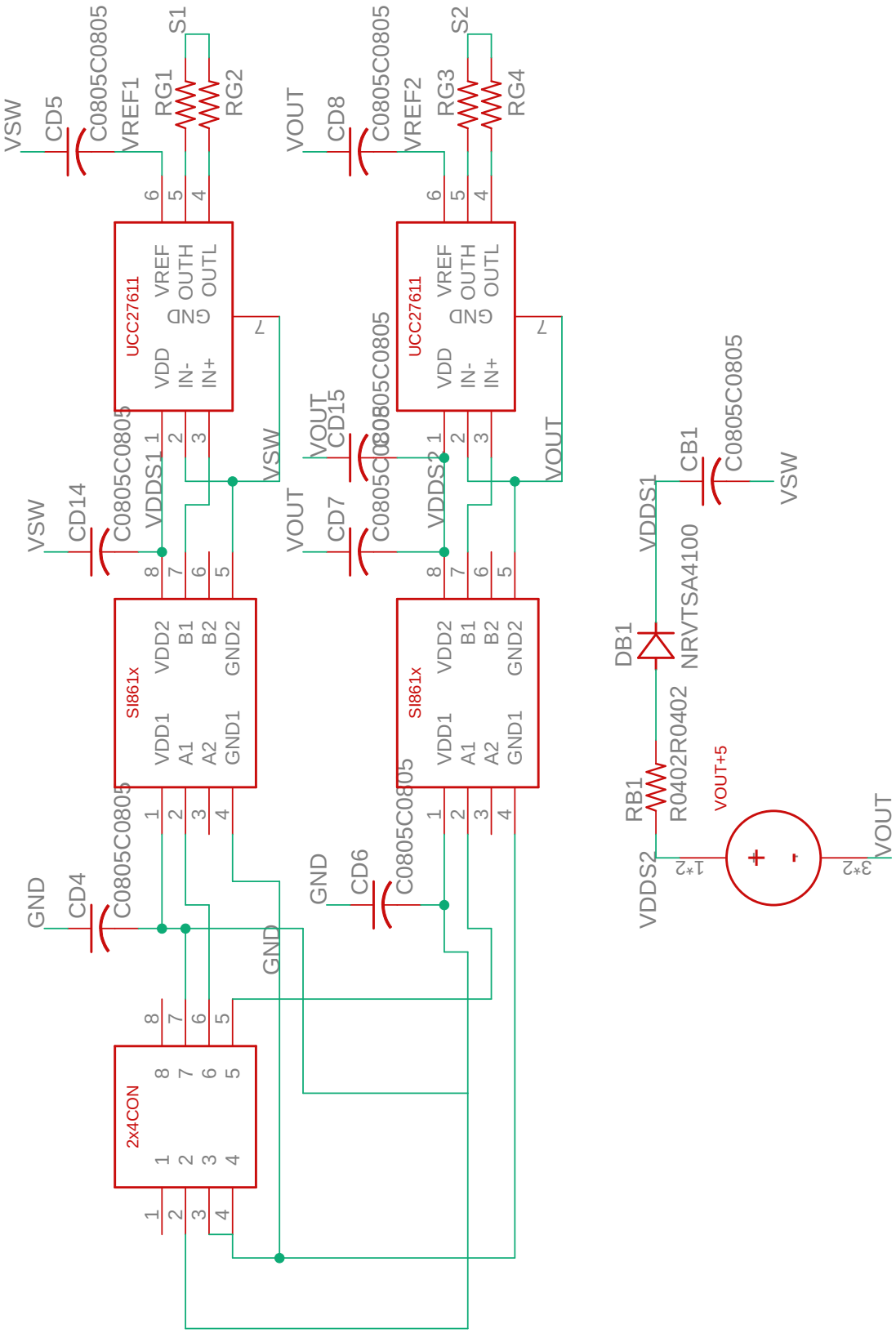
The bill of materials presents all of the parts used, along with their description.

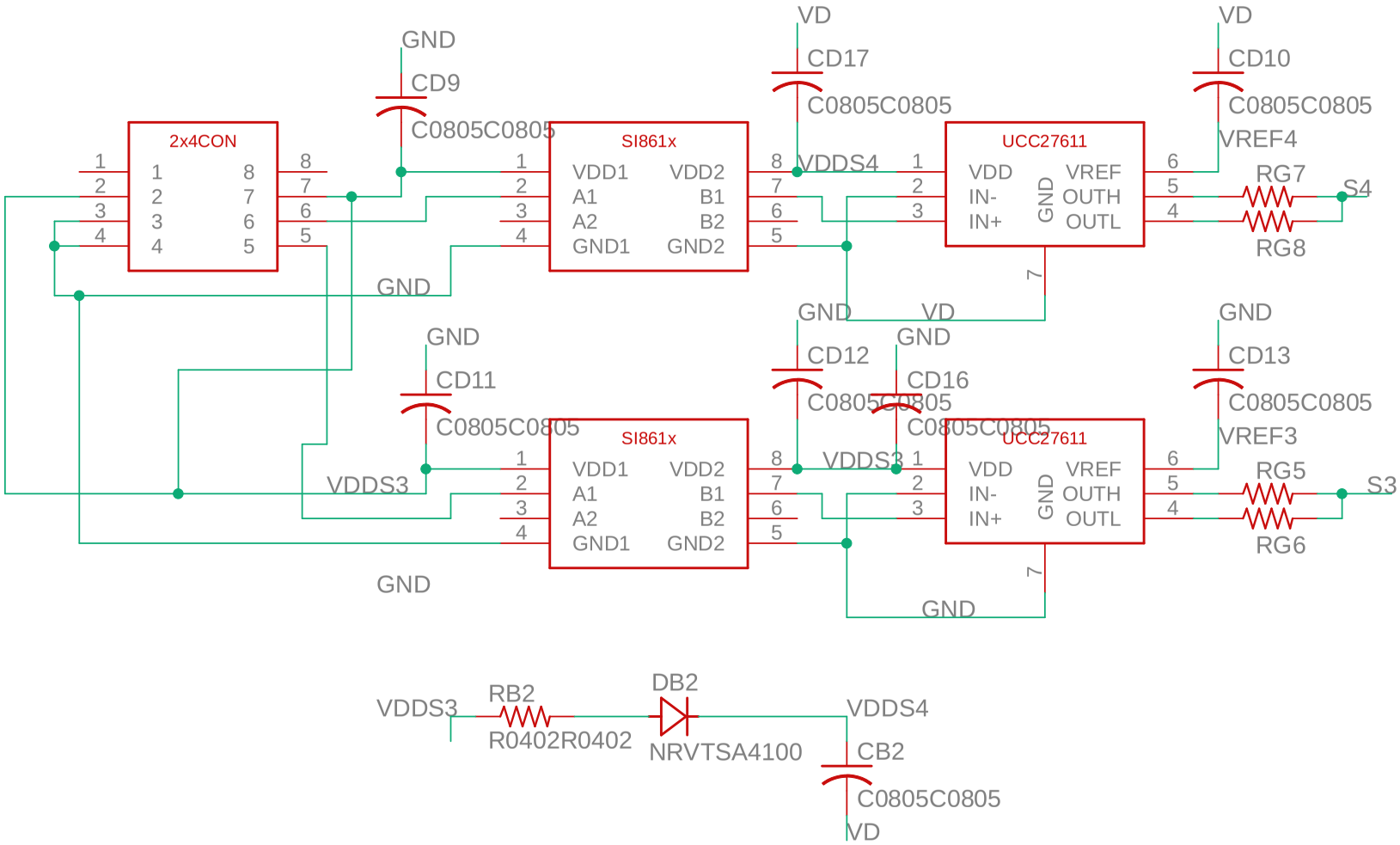
The PCB header pinout presents the values of the individual pins for all of the connectors and headers on the PCB. The tables are oriented in the same way as they are present on the PCB layout, and the * corresponds to the orientation dot on the PCB.


VOUT
VIN
VOUT+5
SCON1
SCON2
Joshua Piel PRC v1




VOUT
VOUT
1*2
3*2
CD2
C0805
C0805
CD19
D2
NRVTSA4100
S2
S2
EPC2019
S4
S4
EPC2019
D1
NRVTSA4100
VD
VSW
P$1
P$2*3
Piezo
2 1
GND
CD3
C0805
C0805
S1
S1
EPC2019
S3
S3
EPC2019
VIN
VIN
1*2
3*2
CD1
IN
VD
GND
GND
IN
GND
IN
VSW
GND
GND
IN
GND


VSW
CD5
C0805C0805
VREF1
RG1
RG2
S1
VOUT
CD8
VREF2
RG3
RG4
S2
UCC27611
VDD
IN-
IN+
GND
VREF
OUTH
OUTL
VSW
CD14
VDDS1
VOUT
CD15
CD7
VDDS2
SI861x
VDD1
A1
A2
GND1
VDD2
B1
B2
GND2
GND
CD4
CD6
2x4CON
DB1
NRVTSA4100
RB1
R0402R0402
VDDS2
CB1
VOUT+5
1*2
3*2


2x4CON
SI861x
UCC27611
GND
CD9
C0805C0805
VD
CD17
VDDS4
CD10
VREF4
RG7
RG8
S4
CD11
VDDS3
CD12
CD16
CD13
VREF3
RG5
RG6
S3
RB2
R0402R0402
DB2
NRVTSA4100
CB2

B.1 Bill of Materials

Description	Part #	Ratings/Info
Piezo	APC International 790	"Disc (diameter 19.8mm, thickness 0.8mm), Material 844"
MOSFET	EPC 2019	"GaN, 200V, 8.5A, BUMPED DIE"
Diode	ON Semiconductor NRVTSA4100T3G	"SCHOTTKY, 4A, 100V, SMA-2"
Terminal Plug	Phoenix Contact 1757035	"PLUG, 4POS, STR, 5.08MM"
Terminal Header	Phoenix Contact 1755752	"HDR, 4POS, VERT, 5.08MM"
Input Bus Capacitor		"3x CERAMIC, 1UF, 450V, X7T, 2220"
Output Bus Capacitor		"CERAMIC, 15UF, 100V, X7S, 2220"
Output Half Bridge Decoupling Capacitor		"CERAMIC, 1000PF, 250V, C0G, 0805 "
Inter-Bus Half Bridge Decoupling Capacitor		"CERAMIC, 1000PF, 250V, C0G, 0805 "
Gate Driver	UCC27611DRVT	IC GATE DRIVER 6SON
Isolator	SI8620BB-B-ISR	DGTL ISO 2.5KV GEN PURP 8SOIC
Gate Resistors		"2-4 ohm, 0402"
Decoupling Capacitors		"CERAMIC, 22UF, 25V, X5R, 0805"
Bootstrap Capacitor		"CERAMIC, 22UF, 25V, X5R, 0805"
Bootstrap Diode	ST Microelectroncis STPS2200U	"SCHOTTKY, 200V, 2A, SMB"
Boostrap Resistor		"2-4 ohm, 0402"
Isolated Supply Term. Plug	Phoenix Contact 1757035	"PLUG, 4POS, STR, 5.08MM"
Isolated Supply Term. Header	Phoenix Contact 1755752	"HDR, 4POS, VERT, 5.08MM"
Control Terminal Header Pins	FCI 67997-208HLF	"HDR, VERT, 8POS, 2.54MM"
Standoffs		"HEX STANDOFF 4-40 NYLON 1/2"""

B.2 PCB Header Pinout

GND	GND	V_{in}	V_{in}

*

Table B.1: V_{in} supply input

*

$V_{out}+5$	$V_{out}+5$	V_{out}	V_{out}

Table B.2: $V_{out}+5$ Supply input

	GND
	GND
	V_{out}
*	V_{out}

Table B.3: V_{out} Load/Output

*

NC	+5V in	GND	GND
NC	+5V in	S1 in	S2 in

Table B.4: SCON1 inputs

NC	NC	*
+5V in	+5V in	
S3 in	GND	
S4 in	GND	

Table B.5: SCON2 inputs

Appendix C

Microcontroller and Sensing Circuit Bill of Materials

This appendix presents the bill of materials for all parts used for constructing the closed-loop controller prototype and the sensing buffer circuitry. The table includes the part numbers as well as a description of each part's purpose.

Item	Qty	Description
296-46777-ND Texas Instruments Control Card TMS320F28379D EVAL	1	Microcontroller
296-52312-ND Texas Instruments Docking Station TMDSHSECDOCK	1	Microcontroller Docking Station
296-39237-5-ND IC OPAMP GP 4 CIRCUIT 14DIP	5	Buffer Op-amp
445-173244-1-ND CAP CER 4.7PF 100V C0G RADIAL	10	Filter cap (not used)
445-175548-ND CAP CER 10PF 100V C0G RADIAL	10	Filter cap (not used)
445-FA18C0G2A471JNU00-ND CAP CER 470PF 100V C0G RADIAL	10	Filter cap
445-174252-1-ND CAP CER 1000PF 250V X7R RADIAL	10	Filter cap
399-14094-1-ND CAP CER 0.1UF 100V X7R RADIAL	10	Op-amp decoupling cap
RS112-KIT-ND RESISTOR KIT 1-1M 1/6W 365PCS	1	Resistor kit for making resistor divider
S910CACT-ND RES 910 OHM 1/4W 1% AXIAL	10	Filter resistor (not used)
RNMF14FAD9K10CT-ND RES 9.1K OHM 1/4W 1% AXIAL	10	Filter resistor

Appendix D

Steady State Solution Code

This appendix presents the MATLAB code used to compute the nonideal PSS solution for a PR given the PR parameters (C_p, C_r, L, R), the input and output voltages (V_{in}, V_{out}), the switching sequence description ("Topology" matrix, see code comments), and the initial condition V_{r1}. This voltage should be negative and is usually larger in magnitude than V_{in} and V_{out}. The script will output the full PSS solution with all corner variables and switching times, and it will also display a state space plot of the solution.

First, the script solves for the ideal PSS solution using the ideal PR parameters. Then, it computes the switching times from the ideal corner variables. Next, it creates the differential equations governing open and connected stages and solves them symbolically in terms of their initial conditions. Then, it creates a system of equations using the time domain waveforms to solve for the nonideal corner variables. The ideal solution is used as the starting point in the MATLAB function vpasolve.

```
function PiezoConverterNew_iL6B(Co, C, L, R, Vin, Vout, top, Vs)
%% INPUT DOCUMENTATION

% Topology
% 2x6 matrix
%
% 1st row - choose switching cycle in terms of Vin and Vout
% Voltage of stage is linear combination of Vin and Vout based on
```

```
% multiplier in matrix
%
% 1st stage | 3rd stage | 5th stage | SS stage
% Vin  Vout | Vin  Vout | Vin  Vout | Vin Vout
%
% Example: for Vin-Vout,-Vout,Zero,Vss=Vin
% [1 -1 0 -1 0 0 1 0]
%
% 2nd Row - current constraints
% Sets current constraints
% kth column is -1 if iLk <= 0, 1 if iLk >= 0, 0 if iLk == 0
%
% Example: iL1, iL2, iL3 < 0; iL5, iL6a > 0; iL4, iL6b = 0
% [1 1 1 0 -1 -1 0 0]



%% IDEAL SOLVER

%clear all

% % Parameters
% Co = 710e-12;
% L = .158;
% C = 35e-12;
% R = 20;
% Vin = 100;
% Vout = 45;
Ceff = C*Co/(C+Co);
Ts = 2*pi*sqrt(L*C);
Vs1_n = Vs;

V = [Vin, Vout];

% Voltage states
```

```
Va = dot(V,top(1,1:2));
Vz = dot(V,top(1,3:4));
Vb = dot(V,top(1,5:6));
Vss = dot(V,top(1,7:8));

% Variables
syms  iL1 iL2 iL3 iL4 iL5 iL6a iL6b vr1 vr2 vr3 vr4 vr5 vr6a vr6b a
    b


vr = [vr1 vr2 vr3 vr4 vr5 vr6a vr6b];
iL = [iL1 iL2 iL3 iL4 iL5 iL6a iL6b];
assume(vr1,'real');
assume(iL.*top(2,1:7) > 0);

fixed_i = iL(top(2,1:7)==0);
free_i = iL(top(2,1:7)~=0);
free_v = vr((4-length(fixed_i)):7);
fixed_v = vr(1:(3-length(fixed_i)));
fixed_sv = [fixed_i fixed_v];

dE1 = 1/2*C*(vr2^2-vr1^2)+1/2*L*(iL2^2-iL1^2);
dE2 = 1/2*C*(vr4^2-vr3^2)+1/2*L*(iL4^2-iL3^2);
dE3 = 1/2*C*(vr6a^2-vr5^2)+1/2*L*(iL6a^2-iL5^2);
Eout = top(1,2)*Vout/Va*dE1+top(1,4)*Vout/Vz*dE2+top(1,6)*Vout/Vb*
    dE3;

eqns = [
            % Equations

            (vr1-Va)^2 + (L/C)*iL1^2 == (vr2 - Va)^2 + (L/C)*iL2
    ^2,... % 1
            ...
            Co*(Va)^2 + C*vr2^2 + L*iL2^2 == Co*Vz^2 + C*vr3^2 + L*
    iL3^2, ... % 2
            Co*(Vz-Va) ==  -C*(vr3 - vr2), ...
```

```
            ...
            (vr3-Vz)^2 + (L/C)*iL3^2 == (vr4 - Vz)^2 + (L/C)*iL4^2,
    ... % 3
            ...
            Co*(Vz)^2 + C*vr4^2 + L*iL4^2 == Co*Vb^2 + C*vr5^2 + L*
    iL5^2, ...  % 4
            Co*(Vb-Vz) ==  -C*(vr5 - vr4), ...
            ...
            (vr5-Vb)^2 + (L/C)*iL5^2 == (vr6a - Vb)^2 + (L/C)*iL6a
    ^2, ... % 5
            ...
            Co*Vb^2 + C*vr6a^2 + L*iL6a^2 == Co*(Vss)^2 + C*vr6b^2 +
     L*iL6b^2, ... % 6a
            Co*(Vss-Vb) == -C*(vr6b - vr6a), ...
            ...
            Co*Vss^2 + C*vr6b^2 + L*iL6b^2 == Co*(Va)^2 + C*vr1^2 +
    L*iL1^2, ... % 6b
            Co*(Va-Vss) == -C*(vr1 - vr6b), ...
            ...
            ...
            ...% Initial conditions and constraints
            ...
            vr1 == Vs1_n, ...
            ...%-Eout/Ts == power,...
            fixed_i == 0 ...
            ];
assumptions

% Solve equations
vars = [vr1 vr2 vr3 vr4 vr5 vr6a vr6b iL1 iL2 iL3 iL4 iL5 iL6a iL6b
    ];
[svr1, svr2, svr3, svr4, svr5, svr6a, svr6b, siL1, siL2, siL3, siL4,
     siL5, siL6a, siL6b] = solve(eqns,vars);


```

```
svr = [svr1,svr2,svr3,svr4,svr5,svr6a,svr6b];
siL = [siL1, siL2, siL3, siL4, siL5, siL6a, siL6b];


% Solve for resonant angles
theta1 = atan2(norm(cross([(svr(1)-Va),siL(1)*sqrt(L/C),0],[(svr(2)-
    Va),siL(2)*sqrt(L/C),0])),dot([(svr(1)-Va),siL(1)*sqrt(L/C),0],[(
    svr(2)-Va),siL(2)*sqrt(L/C),0]));
theta2 = atan2(norm(cross([(Va-svr(2)),siL(2)*sqrt(L/Ceff),0],[(Vz-
    svr(3)),siL(3)*sqrt(L/Ceff),0])),dot([(Va-svr(2)),siL(2)*sqrt(L/
    Ceff),0],[(Vz-svr(3)),siL(3)*sqrt(L/Ceff),0]));
theta3 = atan2(norm(cross([(svr(3)-Vz),siL(3)*sqrt(L/C),0],[(svr(4)-
    Vz),siL(4)*sqrt(L/C),0])),dot([(svr(3)-Vz),siL(3)*sqrt(L/C),0],[(
    svr(4)-Vz),siL(4)*sqrt(L/C),0]));
theta4 = atan2(norm(cross([(Vz-svr(4)),siL(4)*sqrt(L/Ceff),0],[(Vb-
    svr(5)),siL(5)*sqrt(L/Ceff),0])),dot([(Vz-svr(4)),siL(4)*sqrt(L/
    Ceff),0],[(Vb-svr(5)),siL(5)*sqrt(L/Ceff),0]));
theta5 = atan2(norm(cross([(svr(5)-Vb),siL(5)*sqrt(L/C),0],[(svr(6)-
    Vb),siL(6)*sqrt(L/C),0])),dot([(svr(5)-Vb),siL(5)*sqrt(L/C),0],[(
    svr(6)-Vb),siL(6)*sqrt(L/C),0]));
theta6a = atan2(norm(cross([(Vb-svr(6)),siL(6)*sqrt(L/Ceff),0],[(Vss
    -svr(7)),siL(7)*sqrt(L/Ceff),0])),dot([(Vb-svr(6)),siL(6)*sqrt(L/
    Ceff),0],[(Vss-svr(7)),siL(7)*sqrt(L/Ceff),0]));
theta6b = atan2(norm(cross([(Vss-svr(7)),siL(7)*sqrt(L/Ceff),0],[(Va
    -svr(1)),siL(1)*sqrt(L/Ceff),0])),dot([(Vss-svr(7)),siL(7)*sqrt(L
    /Ceff),0],[(Va-svr(1)),siL(1)*sqrt(L/Ceff),0]));

theta = [theta1,theta2,theta3,theta4,theta5,theta6a,theta6b];
% Solve for times
t1 = theta1*sqrt(L*C);
t2 = theta2*sqrt(L*Ceff);
t3 = theta3*sqrt(L*C);
t4 = theta4*sqrt(L*Ceff);
t5 = theta5*sqrt(L*C);
t6a = theta6a*sqrt(L*Ceff);
t6b = theta6b*sqrt(L*Ceff);
```

```
128
129 fnew = 1./(t1+t2+t3+t4+t5+t6a+t6b);
130 T = [t1, t2, t3, t4, t5, t6a, t6b];
131
132
133 % Display
134 % vpa(svr)
135 % vpa(siL)
136 % vpa(T)
137
138
139
140  %% LOSSY SOLVER
141  fprintf("Starting non-ideal solver")
142
143 syms Vp(t) Vs(t) iL(t) VinS1 Vs0S1 iL0S1 VinS2 Vs0S2 iL0S2 CpS CsS
       LSS VS RS VaS VbS VzS VssS;
144
145 assume([CpS, CsS, LSS, RS] > 0);
146 assumeAlso([VinS1 Vs0S1 iL0S1 VinS2 Vs0S2 iL0S2 CpS CsS LSS VS RS],
       'real');
147
148 Cp = Co;
149 Cs = C;
150
151
152
153
154 %Solve diffeq for stage 1 and 3
155
156 %Lossless
157 %eqns1 = [diff(Vp,t) == 0, diff(Vs,t) == iL/CsS, diff(iL, t) == Vp/
       LSS - Vs/LSS];
158 %cond1 = [Vp(0) == VinS1, Vs(0) == Vs0S1, iL(0) == iL0S1];
159
160 %Lossy
```

```
161 eqns1 = [diff(Vp,t) == 0, diff(Vs,t) == iL/CsS, diff(iL, t) == Vp/
      LSS - Vs/LSS - RS/LSS*iL];
162 cond1 = [Vp(0) == VinS1, Vs(0) == Vs0S1, iL(0) == iL0S1];
163 stage1d = dsolve(eqns1, cond1);
164
165 %{
166 pretty(stage1.Vp)
167 pretty(stage1.Vs)
168 pretty(stage1.iL)
169 %}
170
171
172 %Solve diffeq for stage 2 and 4
173 eqns2 = [diff(Vp,t) == -iL/CpS, diff(Vs,t) == iL/CsS, diff(iL,t) ==
      Vp/LSS-Vs/LSS-RS/LSS*iL];
174 cond2 = [Vp(0) == VinS2, Vs(0) == Vs0S2, iL(0) == iL0S2];
175 stage2d = dsolve(eqns2, cond2);
176
177 %Rloss1 = int(stage1d.iL^2*RS, t, 0, t);
178 %Rloss2 = int(stage2d.iL^2*RS, t, 0, t);
179
180 %{
181 pretty(stage2.Vp)
182 pretty(stage2.Vs)
183 pretty(stage2.iL)
184 %}
185
186 Rloss1 = int(stage1d.iL^2*RS, t, 0, t);
187 Rloss2 = int(stage2d.iL^2*RS, t, 0, t);
188
189 VsS = [vr1 vr2 vr3 vr4 vr5 vr6a vr6b]
190 iLS = [iL1 iL2 iL3 iL4 iL5 iL6a iL6b]
191 tS = sym('t',[1,7]);
192
193 eqns = zeros(1,18,'sym');
194
```

```
eqns(1) = VsS(2) - subs(stage1d.Vs,[Vs0S1, iL0S1, VinS1,t],[VsS(1), iLS(1), VaS, tS(1)]);
eqns(2) = iLS(2) - subs(stage1d.iL,[Vs0S1, iL0S1, VinS1,t],[VsS(1), iLS(1), VaS, tS(1)]);

eqns(3) = VsS(3) - subs(stage2d.Vs,[Vs0S2, iL0S2, VinS2,t],[VsS(2), iLS(2), VaS, tS(2)]);
eqns(4) = iLS(3) - subs(stage2d.iL,[Vs0S2, iL0S2, VinS2,t],[VsS(2), iLS(2), VaS, tS(2)]);
eqns(5) = VzS - subs(stage2d.Vp,[Vs0S2, iL0S2, VinS2,t],[VsS(2), iLS(2), VaS, tS(2)]);

eqns(6) = VsS(4) - subs(stage1d.Vs,[Vs0S1, iL0S1, VinS1,t],[VsS(3), iLS(3), VzS, tS(3)]);
eqns(7) = iLS(4) - subs(stage1d.iL,[Vs0S1, iL0S1, VinS1,t],[VsS(3), iLS(3), VzS, tS(3)]);

eqns(8) = VsS(5) - subs(stage2d.Vs,[Vs0S2, iL0S2, VinS2,t],[VsS(4), iLS(4), VzS, tS(4)]);
eqns(9) = iLS(5) - subs(stage2d.iL,[Vs0S2, iL0S2, VinS2,t],[VsS(4), iLS(4), VzS, tS(4)]);
eqns(10) = VbS - subs(stage2d.Vp,[Vs0S2, iL0S2, VinS2,t],[VsS(4), iLS(4), VzS, tS(4)]);

eqns(11) = VsS(6) - subs(stage1d.Vs,[Vs0S1, iL0S1, VinS1,t],[VsS(5), iLS(5), VbS, tS(5)]);
eqns(12) = iLS(6) - subs(stage1d.iL,[Vs0S1, iL0S1, VinS1,t],[VsS(5), iLS(5), VbS, tS(5)]);

eqns(13) = VsS(7) - subs(stage2d.Vs,[Vs0S2, iL0S2, VinS2,t],[VsS(6), iLS(6), VbS, tS(6)]);
eqns(14) = iLS(7) - subs(stage2d.iL,[Vs0S2, iL0S2, VinS2,t],[VsS(6), iLS(6), VbS, tS(6)]);
eqns(15) = VssS - subs(stage2d.Vp,[Vs0S2, iL0S2, VinS2,t],[VsS(6), iLS(6), VbS, tS(6)]);

```

```
eqns(16) = VsS(1) - subs(stage2d.Vs,[Vs0S2, iL0S2, VinS2,t],[VsS(7),
    iLS(7), VssS, tS(7)]);
eqns(17) = iLS(1) - subs(stage2d.iL,[Vs0S2, iL0S2, VinS2,t],[VsS(7),
    iLS(7), VssS, tS(7)]);
eqns(18) = VaS - subs(stage2d.Vp,[Vs0S2, iL0S2, VinS2,t],[VsS(7),
   iLS(7), VssS, tS(7)]);

sfixed_i = siL(top(2,1:7)==0)
sfixed_v = svr(1:(3-length(sfixed_i)))
sfixed_sv = [sfixed_i sfixed_v]

sfree_i = vpa(siL(top(2,1:7)~=0))
sfree_v = vpa(svr((4-length(fixed_i)):7))


eqns_n = subs(eqns,[CpS CsS LSS RS VaS VzS VbS VssS fixed_sv],[Co C
   L R Va Vz Vb Vss sfixed_sv])

symvar(eqns_n)

guess = [sfree_v sfree_i T]
solve_vars = [free_v free_i tS]

out = vpasolve(eqns_n,solve_vars,guess)

%Voltage
if in(vr1,fixed_sv)
    Vs0 = svr(1);
else
    Vs0 = out.vr1;
end

if in(vr2,fixed_sv)
    Vs1 = svr(2);
else
    Vs1 = out.vr2;
```

```
248 end
249
250 if in(vr3,fixed_sv)
251     Vs2 = svr(3);
252 else
253     Vs2 = out.vr3;
254 end
255
256 if in(vr4,fixed_sv)
257     Vs3 = svr(4);
258 else
259     Vs3 = out.vr4;
260 end
261
262 if in(vr5,fixed_sv)
263     Vs4 = svr(5);
264 else
265     Vs4 = out.vr5;
266 end
267
268 if in(vr6a,fixed_sv)
269     Vs5 = svr(6);
270 else
271     Vs5 = out.vr6a;
272 end
273
274 if in(vr6b,fixed_sv)
275     Vs5b = svr(7);
276 else
277     Vs5b = out.vr6b;
278 end
279
280 %Current
281 if in(iL1,fixed_sv)
282     iL0 = siL(1);
283 else
```

```
284     iL0 = out.iL1;
285 end
286 
287 if in(iL2,fixed_sv)
288     iL1 = siL(2);
289 else
290     iL1 = out.iL2;
291 end
292 
293 if in(iL3,fixed_sv)
294     iL2 = siL(3);
295 else
296     iL2 = out.iL3;
297 end
298 
299 if in(iL4,fixed_sv)
300     iL3 = siL(4);
301 else
302     iL3 = out.iL4;
303 end
304 
305 if in(iL5,fixed_sv)
306     iL4 = siL(5);
307 else
308     iL4 = out.iL5;
309 end
310 
311 if in(iL6a,fixed_sv)
312     iL5 = siL(6);
313 else
314     iL5 = out.iL6a;
315 end
316 
317 if in(iL6b,fixed_sv)
318     iL5b = siL(7);
319 else
```

```
320     iL5b = out.iL6b;
321 end
322
323 t1 = out.t1;
324 t2 = out.t2;
325 t3 = out.t3;
326 t4 = out.t4;
327 t5 = out.t5;
328 t6a = out.t6;
329 t6b = out.t7;
330
331 Vs_n = [Vs0, Vs1, Vs2, Vs3, Vs4, Vs5, Vs5b]
332 iL_n = [iL0, iL1, iL2, iL3, iL4, iL5, iL5b]
333 t_n = [t1, t2, t3 , t4, t5, t6a, t6b]
334
335 E0 = 1/2*Cp*Va^2+1/2*Cs*Vs0^2+1/2*L*iL0^2;
336 E1 = 1/2*Cp*Va^2+1/2*Cs*Vs1^2+1/2*L*iL1^2;
337 E3 = 1/2*Cp*Vz^2+1/2*Cs*Vs3^2+1/2*L*iL3^2;
338 E4 = 1/2*Cp*Vb^2+1/2*Cs*Vs4^2+1/2*L*iL4^2;
339 E5 = 1/2*Cp*Vb^2+1/2*Cs*Vs5^2+1/2*L*iL5^2;
340
341
342 %%{
343 figure(1)
344
345
346
347 fplot(Va, subs(stage1d.iL,[VinS1, Vs0S1, iL0S1, CsS, LSS, RS], [Va,
       Vs0, iL0, Cs, L, R]),[0, double(t1)])
348
349 hold on
350 fplot(subs(stage2d.Vp,[VinS2, Vs0S2, iL0S2, CpS, CsS, LSS, RS], [Va,
        Vs1, iL1, Cp, Cs, L, R]), subs(stage2d.iL,[VinS2, Vs0S2, iL0S2,
       CpS, CsS, LSS, RS], [Va, Vs1, iL1, Cp, Cs, L, R]), [0,double(t2)
       ]);
351 fplot(Vz, subs(stage1d.iL,[VinS1, Vs0S1, iL0S1, CpS, CsS, LSS, RS],
```

```
    [Vz, Vs2, iL2, Cp, Cs, L, R]),[0, double(t3)])
352
353 fplot(subs(stage2d.Vp,[VinS2, Vs0S2, iL0S2, CpS, CsS, LSS, RS], [Vz,
     Vs3, iL3, Cp, Cs, L, R]), subs(stage2d.iL,[VinS2, Vs0S2, iL0S2,
    CpS, CsS, LSS, RS], [Vz, Vs3, iL3, Cp, Cs, L, R]), [0,double(t4)
    ]);
354 fplot(Vb, subs(stage1d.iL,[VinS1, Vs0S1, iL0S1, CpS, CsS, LSS, RS],
    [Vb, Vs4, iL4, Cp, Cs, L, R]),[0, double(t5)])
355 fplot(subs(stage2d.Vp,[VinS2, Vs0S2, iL0S2, CpS, CsS, LSS, RS], [Vb,
     Vs5, iL5, Cp, Cs, L, R]), subs(stage2d.iL,[VinS2, Vs0S2, iL0S2,
    CpS, CsS, LSS, RS], [Vb, Vs5, iL5, Cp, Cs, L, R]), [0,double(t6a+
    t6b)],'Color','red');
356
357 hold off
358
359 figure(2)
360
361 fplot(subs(stage1d.Vs,[VinS1, Vs0S1, iL0S1, CsS, LSS, RS], [Va, Vs0,
     iL0, Cs, L, R]), subs(stage1d.iL,[VinS1, Vs0S1, iL0S1, CsS, LSS,
     RS], [Va, Vs0, iL0, Cs, L, R]),[0, double(t1)])
362
363
364 hold on
365 fplot(subs(stage2d.Vs,[VinS2, Vs0S2, iL0S2, CpS, CsS, LSS, RS], [Va,
     Vs1, iL1, Cp, Cs, L, R]), subs(stage2d.iL,[VinS2, Vs0S2, iL0S2,
    CpS, CsS, LSS, RS], [Va, Vs1, iL1, Cp, Cs, L, R]), [0,double(t2)
    ]);
366 fplot(subs(stage1d.Vs,[VinS1, Vs0S1, iL0S1, CpS, CsS, LSS, RS], [Vz,
     Vs2, iL2, Cp, Cs, L, R]), subs(stage1d.iL,[VinS1, Vs0S1, iL0S1,
    CpS, CsS, LSS, RS], [Vz, Vs2, iL2, Cp, Cs, L, R]),[0, double(t3)
    ])
367
368 fplot(subs(stage2d.Vs,[VinS2, Vs0S2, iL0S2, CpS, CsS, LSS, RS], [Vz,
     Vs3, iL3, Cp, Cs, L, R]), subs(stage2d.iL,[VinS2, Vs0S2, iL0S2,
    CpS, CsS, LSS, RS], [Vz, Vs3, iL3, Cp, Cs, L, R]), [0,double(t4)
    ]);
```

```
fplot(subs(stage1d.Vs,[VinS1, Vs0S1, iL0S1, CpS, CsS, LSS, RS], [Vb, Vs4, iL4, Cp, Cs, L, R]), subs(stage1d.iL,[VinS1, Vs0S1, iL0S1, CpS, CsS, LSS, RS], [Vb, Vs4, iL4, Cp, Cs, L, R]),[0, double(t5)])
fplot(subs(stage2d.Vs,[VinS2, Vs0S2, iL0S2, CpS, CsS, LSS, RS], [Vb, Vs5, iL5, Cp, Cs, L, R]), subs(stage2d.iL,[VinS2, Vs0S2, iL0S2, CpS, CsS, LSS, RS], [Vb, Vs5, iL5, Cp, Cs, L, R]), [0,double(t6a+t6b)],'Color','red');


hold off

%Output scatter plot data to csv file
%TODO - make general method for switch nodes
%TODO - Do substitutions once so this code runs a lot faster, right now it
%does the same substitutions repeatedly

% scatter_step = 100;
%
% fprintf('Generating scatter plots\n');
%
% scatter1 = [subs([Va, subs(stage1d.iL,[VinS1, Vs0S1, iL0S1, CsS, LSS, RS], [Va, Vs0, iL0, Cs, L, R])], t, linspace(0, double(t1), scatter_step)');...
%     subs([subs(stage2d.Vp,[VinS2, Vs0S2, iL0S2, CpS, CsS, LSS, RS], [Va, Vs1, iL1, Cp, Cs, L, R]), subs(stage2d.iL,[VinS2, Vs0S2, iL0S2, CpS, CsS, LSS, RS], [Va, Vs1, iL1, Cp, Cs, L, R])], t, linspace(0,double(t2),scatter_step)');...
%     subs([Vz, subs(stage1d.iL,[VinS1, Vs0S1, iL0S1, CpS, CsS, LSS, RS], [Vz, Vs2, iL2, Cp, Cs, L, R])], t, linspace(0, double(t3), scatter_step)');...
%     subs([subs(stage2d.Vp,[VinS2, Vs0S2, iL0S2, CpS, CsS, LSS, RS], [Vz, Vs3, iL3, Cp, Cs, L, R]), subs(stage2d.iL,[VinS2, Vs0S2, iL0S2, CpS, CsS, LSS, RS], [Vz, Vs3, iL3, Cp, Cs, L, R])], t, linspace(0,double(t4),scatter_step)');...
```

```
388 %     subs([Vb, subs(stage1d.iL,[VinS1, Vs0S1, iL0S1, CpS, CsS, LSS,
      RS], [Vb, Vs4, iL4, Cp, Cs, L, R])], t, linspace(0, double(t5),
    scatter_step)');...
389 %     subs([subs(stage2d.Vp,[VinS2, Vs0S2, iL0S2, CpS, CsS, LSS, RS
    ], [Vb, Vs5, iL5, Cp, Cs, L, R]), subs(stage2d.iL,[VinS2, Vs0S2,
    iL0S2, CpS, CsS, LSS, RS], [Vb, Vs5, iL5, Cp, Cs, L, R])], t,
    linspace(0,double(t6a+t6b),scatter_step)')];
390 %
391 %
392 % scatter2 = [subs([subs(stage1d.Vs,[VinS1, Vs0S1, iL0S1, CsS, LSS,
    RS], [Va, Vs0, iL0, Cs, L, R]), subs(stage1d.iL,[VinS1, Vs0S1,
    iL0S1, CsS, LSS, RS], [Va, Vs0, iL0, Cs, L, R])], t, linspace(0,
    double(t1),scatter_step)');...
393 %     subs([subs(stage2d.Vs,[VinS2, Vs0S2, iL0S2, CpS, CsS, LSS, RS
    ], [Va, Vs1, iL1, Cp, Cs, L, R]), subs(stage2d.iL,[VinS2, Vs0S2,
    iL0S2, CpS, CsS, LSS, RS], [Va, Vs1, iL1, Cp, Cs, L, R])], t,
    linspace(0,double(t2),scatter_step)');...
394 %     subs([subs(stage1d.Vs,[VinS1, Vs0S1, iL0S1, CpS, CsS, LSS, RS
    ], [Vz, Vs2, iL2, Cp, Cs, L, R]), subs(stage1d.iL,[VinS1, Vs0S1,
    iL0S1, CpS, CsS, LSS, RS], [Vz, Vs2, iL2, Cp, Cs, L, R])], t,
    linspace(0, double(t3),scatter_step)');...
395 %     subs([subs(stage2d.Vs,[VinS2, Vs0S2, iL0S2, CpS, CsS, LSS, RS
    ], [Vz, Vs3, iL3, Cp, Cs, L, R]), subs(stage2d.iL,[VinS2, Vs0S2,
    iL0S2, CpS, CsS, LSS, RS], [Vz, Vs3, iL3, Cp, Cs, L, R])], t,
    linspace(0,double(t4),scatter_step)');...
396 %     subs([subs(stage1d.Vs,[VinS1, Vs0S1, iL0S1, CpS, CsS, LSS, RS
    ], [Vb, Vs4, iL4, Cp, Cs, L, R]), subs(stage1d.iL,[VinS1, Vs0S1,
    iL0S1, CpS, CsS, LSS, RS], [Vb, Vs4, iL4, Cp, Cs, L, R])], t,
    linspace(0, double(t5),scatter_step)');...
397 %     subs([subs(stage2d.Vs,[VinS2, Vs0S2, iL0S2, CpS, CsS, LSS, RS
    ], [Vb, Vs5, iL5, Cp, Cs, L, R]), subs(stage2d.iL,[VinS2, Vs0S2,
    iL0S2, CpS, CsS, LSS, RS], [Vb, Vs5, iL5, Cp, Cs, L, R])], t,
    linspace(0,double(t6a+t6b),scatter_step)')];
398 %
399 % scatter3 = [subs([t, subs(stage1d.Vp,[VinS1, Vs0S1, iL0S1, CsS,
    LSS, RS], [Va, Vs0, iL0, Cs, L, R])], t, linspace(0, double(t1),
```

```
    scatter_step)');...
400 %     subs([t+t1, subs(stage2d.Vp,[VinS2, Vs0S2, iL0S2, CpS, CsS,
    LSS, RS], [Va, Vs1, iL1, Cp, Cs, L, R])], t, linspace(0,double(t2
    ),scatter_step)');...
401 %     subs([t+t1+t2, subs(stage1d.Vp,[VinS1, Vs0S1, iL0S1, CpS, CsS,
     LSS, RS], [Vz, Vs2, iL2, Cp, Cs, L, R])], t, linspace(0, double(
    t3),scatter_step)');...
402 %     subs([t+t1+t2+t3, subs(stage2d.Vp,[VinS2, Vs0S2, iL0S2, CpS,
    CsS, LSS, RS], [Vz, Vs3, iL3, Cp, Cs, L, R])], t, linspace(0,
    double(t4),scatter_step)');...
403 %     subs([t+t1+t2+t3+t4, subs(stage1d.Vp,[VinS1, Vs0S1, iL0S1, CpS
    , CsS, LSS, RS], [Vb, Vs4, iL4, Cp, Cs, L, R])], t, linspace(0,
    double(t5),scatter_step)');...
404 %     subs([t+t1+t2+t3+t4+t5, subs(stage2d.Vp,[VinS2, Vs0S2, iL0S2,
    CpS, CsS, LSS, RS], [Vb, Vs5, iL5, Cp, Cs, L, R])], t, linspace
    (0,double(t6a+t6b),scatter_step)')];
405 %
406 % scatter4 = [subs([t, subs(stage1d.Vs,[VinS1, Vs0S1, iL0S1, CsS,
    LSS, RS], [Va, Vs0, iL0, Cs, L, R])], t, linspace(0, double(t1),
    scatter_step)');...
407 %     subs([t+t1, subs(stage2d.Vs,[VinS2, Vs0S2, iL0S2, CpS, CsS,
    LSS, RS], [Va, Vs1, iL1, Cp, Cs, L, R])], t, linspace(0,double(t2
    ),scatter_step)');...
408 %     subs([t+t1+t2, subs(stage1d.Vs,[VinS1, Vs0S1, iL0S1, CpS, CsS,
     LSS, RS], [Vz, Vs2, iL2, Cp, Cs, L, R])], t, linspace(0, double(
    t3),scatter_step)');...
409 %     subs([t+t1+t2+t3, subs(stage2d.Vs,[VinS2, Vs0S2, iL0S2, CpS,
    CsS, LSS, RS], [Vz, Vs3, iL3, Cp, Cs, L, R])], t, linspace(0,
    double(t4),scatter_step)');...
410 %     subs([t+t1+t2+t3+t4, subs(stage1d.Vs,[VinS1, Vs0S1, iL0S1, CpS
    , CsS, LSS, RS], [Vb, Vs4, iL4, Cp, Cs, L, R])], t, linspace(0,
    double(t5),scatter_step)');...
411 %     subs([t+t1+t2+t3+t4+t5, subs(stage2d.Vs,[VinS2, Vs0S2, iL0S2,
    CpS, CsS, LSS, RS], [Vb, Vs5, iL5, Cp, Cs, L, R])], t, linspace
    (0,double(t6a+t6b),scatter_step)')];
412 %
```

```
413 % scatter5 = [subs([t, subs(stage1d.iL,[VinS1, Vs0S1, iL0S1, CsS,
    LSS, RS], [Va, Vs0, iL0, Cs, L, R])], t, linspace(0, double(t1),
    scatter_step)');...
414 %     subs([t+t1, subs(stage2d.iL,[VinS2, Vs0S2, iL0S2, CpS, CsS,
    LSS, RS], [Va, Vs1, iL1, Cp, Cs, L, R])], t, linspace(0,double(t2
    ),scatter_step)');...
415 %     subs([t+t1+t2, subs(stage1d.iL,[VinS1, Vs0S1, iL0S1, CpS, CsS,
     LSS, RS], [Vz, Vs2, iL2, Cp, Cs, L, R])], t, linspace(0, double(
    t3),scatter_step)');...
416 %     subs([t+t1+t2+t3, subs(stage2d.iL,[VinS2, Vs0S2, iL0S2, CpS,
    CsS, LSS, RS], [Vz, Vs3, iL3, Cp, Cs, L, R])], t, linspace(0,
    double(t4),scatter_step)');...
417 %     subs([t+t1+t2+t3+t4, subs(stage1d.iL,[VinS1, Vs0S1, iL0S1, CpS
    , CsS, LSS, RS], [Vb, Vs4, iL4, Cp, Cs, L, R])], t, linspace(0,
    double(t5),scatter_step)');...
418 %     subs([t+t1+t2+t3+t4+t5, subs(stage2d.iL,[VinS2, Vs0S2, iL0S2,
    CpS, CsS, LSS, RS], [Vb, Vs5, iL5, Cp, Cs, L, R])], t, linspace
    (0,double(t6a+t6b),scatter_step)')];
419 %
420 % scatter6 = [subs([t, subs(top(1,1)*stage1d.iL,[VinS1, Vs0S1, iL0S1
    , CsS, LSS, RS], [Va, Vs0, iL0, Cs, L, R])], t, linspace(0,
    double(t1),scatter_step)');...
421 %     subs([t+t1, 0], t, linspace(0,double(t2),scatter_step)');...
422 %     subs([t+t1+t2, subs(top(1,3)*stage1d.iL,[VinS1, Vs0S1, iL0S1,
    CpS, CsS, LSS, RS], [Vz, Vs2, iL2, Cp, Cs, L, R])], t, linspace
    (0, double(t3),scatter_step)');...
423 %     subs([t+t1+t2+t3, 0], t, linspace(0,double(t4),scatter_step)')
    ;...
424 %     subs([t+t1+t2+t3+t4, subs(top(1,5)*stage1d.iL,[VinS1, Vs0S1,
    iL0S1, CpS, CsS, LSS, RS], [Vb, Vs4, iL4, Cp, Cs, L, R])], t,
    linspace(0, double(t5),scatter_step)');...
425 %     subs([t+t1+t2+t3+t4+t5, 0], t, linspace(0,double(t6a+t6b),
    scatter_step)')];
426 %
427 % scatter7 = [subs([t, -subs(top(1,2)*stage1d.iL,[VinS1, Vs0S1,
    iL0S1, CsS, LSS, RS], [Va, Vs0, iL0, Cs, L, R])], t, linspace(0,
```

```
    double(t1),scatter_step)');...
%     subs([t+t1, 0], t, linspace(0,double(t2),scatter_step)');...
%     subs([t+t1+t2, -subs(top(1,4)*stage1d.iL,[VinS1, Vs0S1, iL0S1,
     CpS, CsS, LSS, RS], [Vz, Vs2, iL2, Cp, Cs, L, R])], t, linspace
    (0, double(t3),scatter_step)');...
%     subs([t+t1+t2+t3, 0], t, linspace(0,double(t4),scatter_step)')
    ;...
%     subs([t+t1+t2+t3+t4, -subs(top(1,6)*stage1d.iL,[VinS1, Vs0S1,
    iL0S1, CpS, CsS, LSS, RS], [Vb, Vs4, iL4, Cp, Cs, L, R])], t,
    linspace(0, double(t5),scatter_step)');...
%     subs([t+t1+t2+t3+t4+t5, 0], t, linspace(0,double(t6a+t6b),
    scatter_step)')];
%
%
% %SCATTER8 AND SCATTER9 ARE SPECIFIC TO VIN-VOUT,ZERO,VOUT
% scatter8 = [subs([t, Vin], t, linspace(0, double(t1),scatter_step)
    ');...
%     subs([t+t1, Vout+subs(stage2d.Vp,[VinS2, Vs0S2, iL0S2, CpS,
    CsS, LSS, RS], [Va, Vs1, iL1, Cp, Cs, L, R])], t, linspace(0,
    double(t2),scatter_step)');...
%     subs([t+t1+t2, Vout], t, linspace(0, double(t3),scatter_step)
    ');...
%     subs([t+t1+t2+t3, Vout], t, linspace(0,double(t4),scatter_step
    )');...
%     subs([t+t1+t2+t3+t4, Vout], t, linspace(0, double(t5),
    scatter_step)');...
%     subs([t+t1+t2+t3+t4+t5, subs(stage2d.Vp,[VinS2, Vs0S2, iL0S2,
    CpS, CsS, LSS, RS], [Vb, Vs5, iL5, Cp, Cs, L, R])], t, linspace
    (0,double(t6a),scatter_step)');...
%     subs([t+t1+t2+t3+t4+t5, Vin], t, linspace(double(t6a), double(
    t6a+t6b),scatter_step)')];
%
% scatter9 = [subs([t, Vout], t, linspace(0, double(t1),scatter_step
    )');...
%     subs([t+t1, Vout], t, linspace(0,double(t2),scatter_step)')
    ;...
```

```
%     subs([t+t1+t2, Vout], t, linspace(0, double(t3),scatter_step)
    ');...
%     subs([t+t1+t2+t3, Vout-subs(stage2d.Vp,[VinS2, Vs0S2, iL0S2,
    CpS, CsS, LSS, RS], [Vz, Vs3, iL3, Cp, Cs, L, R])], t, linspace
    (0,double(t4),scatter_step)');...
%     subs([t+t1+t2+t3+t4, 0], t, linspace(0, double(t5),
    scatter_step)');...
%     subs([t+t1+t2+t3+t4+t5, 0], t, linspace(0,double(t6a),
    scatter_step)');...
%     subs([t+t1+t2+t3+t4+t5, Vin-subs(stage2d.Vp,[VinS2, Vs0S2,
    iL0S2, CpS, CsS, LSS, RS], [Vb, Vs5, iL5, Cp, Cs, L, R])], t,
    linspace(double(t6a),double(t6a+t6b),scatter_step)')];
%
%
% csvwrite("VpvsiL.csv",double(scatter1));
% csvwrite("VrvsiL.csv",double(scatter2));
% csvwrite("Vpvst.csv",double(scatter3));
% csvwrite("Vrvst.csv",double(scatter4));
% csvwrite("iLvst.csv",double(scatter5));
% csvwrite("iinvst.csv",double(scatter6));
% csvwrite("ioutvst.csv",double(scatter7));
% csvwrite("Vsw1vst for VIN-VOUT ZERO VOUT.csv",double(scatter8));
% csvwrite("Vsw2vst for VIN-VOUT ZERO VOUT.csv",double(scatter9));


%}

%This area calculates information about converter operation,
    including
%   frequency, power, efficiency, Q, for the different converter
%   topologies. This is because a single state plane could represent
%   "different" physical (ie where the power goes) operation of the
%   different converter types

%In each case, energy transferred/dissipated is calculated from the
%   equations and time/period is known from the time variables to
```

```
    switch
%   calculated earlier

pd = t1+t2+t3+t4+t5+t6a+t6b;
fprintf('PERIOD: %e\n', pd)
fprintf('FREQUENCY: %e\n\n', 1/pd)

Eloss = zeros(1,6);
Eloss(1) = vpa(subs(Rloss1,[VinS1, Vs0S1, iL0S1, CpS, CsS, LSS, RS,
    t], [Va, Vs0, iL0, Cp, Cs, L, R, t1]));
Eloss(2) = vpa(subs(Rloss2,[VinS2, Vs0S2, iL0S2, CpS, CsS, LSS, RS,
    t], [Va, Vs1, iL1, Cp, Cs, L, R, t2]));
Eloss(3) = vpa(subs(Rloss1,[VinS1, Vs0S1, iL0S1, CpS, CsS, LSS, RS,
    t], [Vz, Vs2, iL2, Cp, Cs, L, R, t3]));

Eloss(4) = vpa(subs(Rloss2,[VinS2, Vs0S2, iL0S2, CpS, CsS, LSS, RS,
    t], [Vz, Vs3, iL3, Cp, Cs, L, R, t4]));
Eloss(5) = vpa(subs(Rloss1,[VinS1, Vs0S1, iL0S1, CpS, CsS, LSS, RS,
    t], [Vb, Vs4, iL4, Cp, Cs, L, R, t5]));
Eloss(6) = vpa(subs(Rloss2,[VinS2, Vs0S2, iL0S2, CpS, CsS, LSS, RS,
    t], [Vb, Vs5, iL5, Cp, Cs, L, R, t6a+t6b]));

i_int = int(stage1d.iL,t,0,t);
I1 = subs(i_int,[VinS1, Vs0S1, iL0S1, CpS, CsS, LSS, RS, t], [Va,
    Vs0, iL0, Cp, Cs, L, R, t1]);
I3 = subs(i_int,[VinS1, Vs0S1, iL0S1, CpS, CsS, LSS, RS, t], [Vz,
    Vs2, iL2, Cp, Cs, L, R, t3]);
I5 = subs(i_int,[VinS1, Vs0S1, iL0S1, CpS, CsS, LSS, RS, t], [Vb,
    Vs4, iL4, Cp, Cs, L, R, t5]);

%TOPOLOGY SPECIFIC
fprintf("Converter Stats @ Vin = %d, Vout = %d\n",Vin,Vout)

Eout = Vout*dot(top(1,:),[0 I1 0 I3 0 I5 0 0]);
fprintf('Eout = %e\n', Eout);

```

```
pwr = -Eout/pd;
fprintf('Power = %e\n', pwr)

Es = max([E1,E3,E5]);
Q = vpa(2*pi*Es/sum(Eloss));
fprintf('Q = %e\n', Q)

Et_Es = vpa(-Eout/Es);
fprintf('Et/Es = %e\n', Et_Es)
eff0 = Q*Et_Es/(2*pi);
eff1 = eff0/(1+eff0);
fprintf('Eff = %e\n\n', eff1)

%{
Eloss

swloss = (1/R).*Eloss.*[.05+.4 0 .05+.4 0 .05+.4 0]
sum(swloss)/pd
sum(Eloss)/pd
sum(swloss+Eloss)/pd

vpa(-Eout/(-Eout+sum(Eloss)+sum(swloss)))

%Eloss
%}


%TUNING PARAMETER .9546

fprintf('period = %d\n', pd/1.04e-9);
fprintf('J/K total high side (t1+t6a+t6b) = %d\n', (t1+t6a+t6b)/1.04
   e-9);
fprintf('high side on time (t1+t6b) = %d\n', (t1+t6b)/1.04e-9);
fprintf('low side on time (t3+t4+t5) = %d\n', (t3+t4+t5)/1.04e-9);
fprintf('O/P low side dead time(t2) = %d\n', t2/1.04e-9);
fprintf('Z/X high side dead time(t6) = %d\n', t6a/1.04e-9);
```

```
535
536 for i = 1:length(t_n)
537     fprintf('.param t%d = %e\n', i, t_n(i))
538 end
539
540 fprintf('\nFPGA\n')
541 fprintf('period = %d\n', pd/1e-8);
542 fprintf('sw1 on time (t1+t6b) = %d\n', (t1+t6b)/1e-8);
543 fprintf('low side on time (t3+t4+t5) = %d\n', (t3+t4+t5)/1e-8);
544 fprintf('Phase/dead time = %d\n', t2/1e-8);
545
546 end
547
548 function b = in(v,a)
549 b = any(v==a);
550 end
```

Appendix E

Simulink Simulation Model

This appendix presents the block diagrams and schematics used in the Simulink dynamic circuit simulation. Simulink projects are not code, but graphical, so this section will display images of the various subsystems with explanations of their functions. Implementations for sensed and static control are given. Both versions implement simulations of the $V_{in} - V_{out}$, $Zero$, V_{out} switching sequence with $V_{out} < \frac{1}{2}V_{in}$. More details can be found in Chapter 5.1. The embedded images are high resolution, so zooming in to view the details is recommended.

E.1 Sensed Control Simulink Simulation

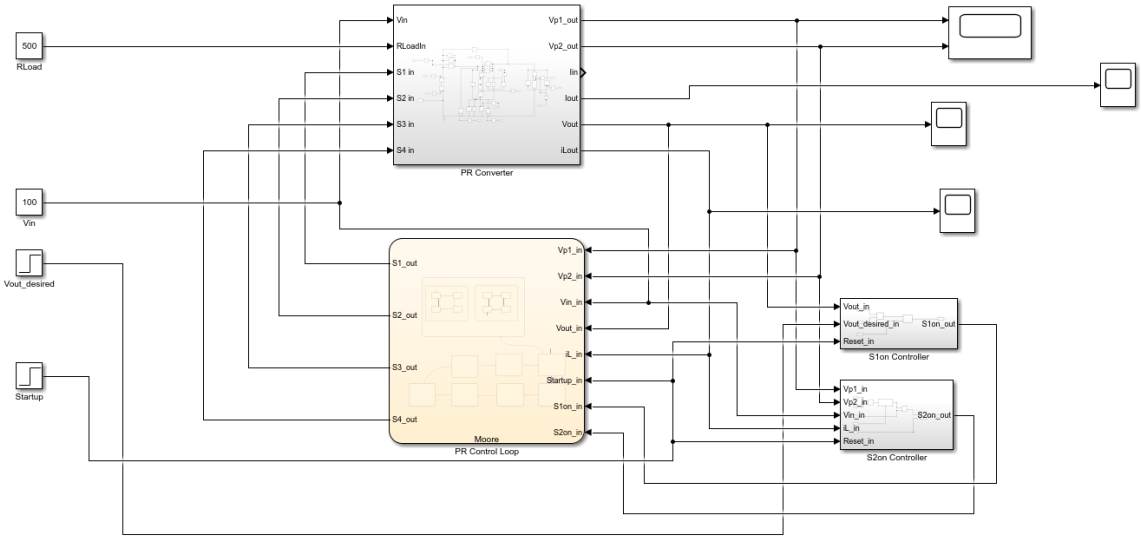


Figure E-1: Top Level Schematic. Integrates the circuit, switch controller FSM, and feedback loops.

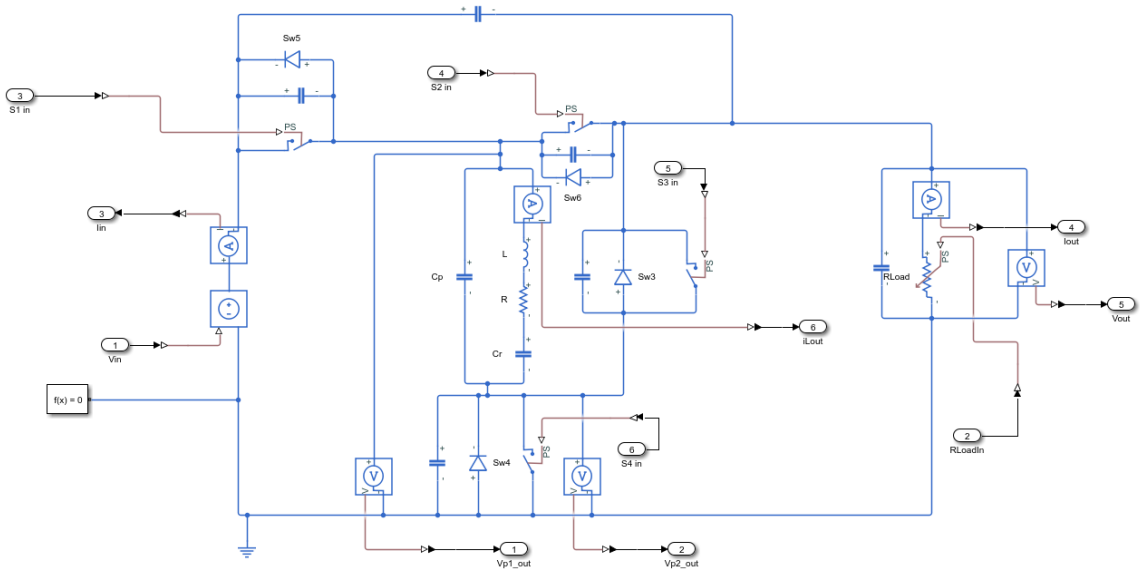


Figure E-2: Circuit Schematic. Implements the topology capable of realizing the$V_{in} - V_{out}$, $Zero$, V_{out} switching sequence.

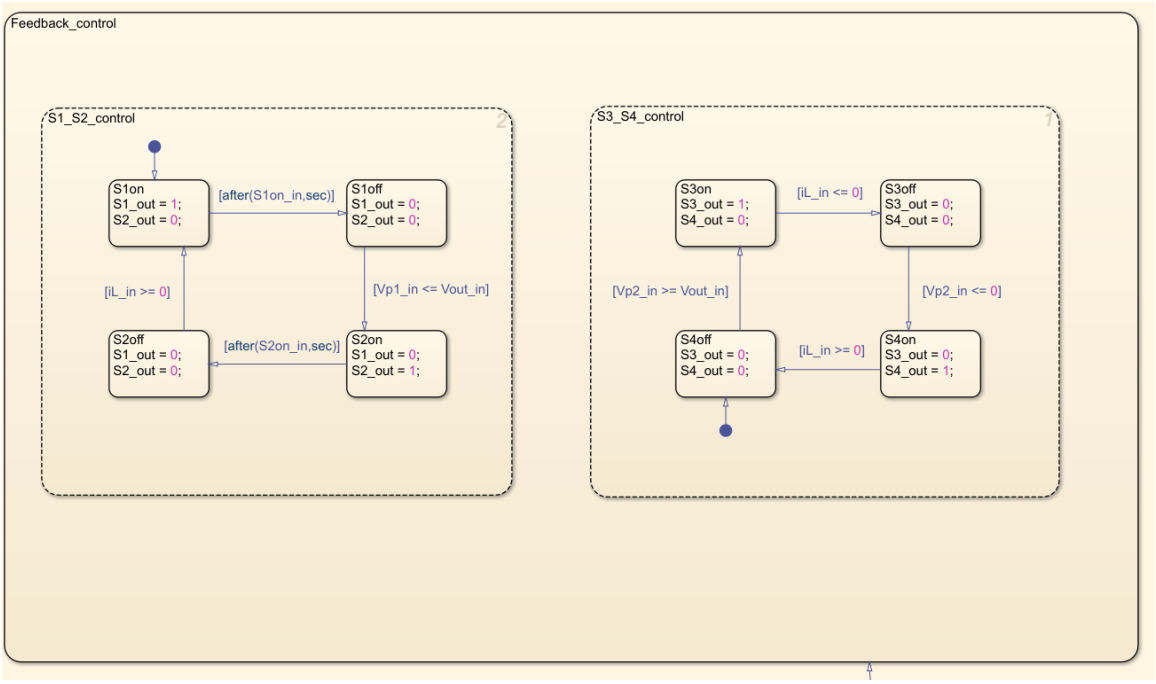


Figure E-3: Switch Control FSM Diagram. Implements the control conditions for sensed control described in Chapter 5.

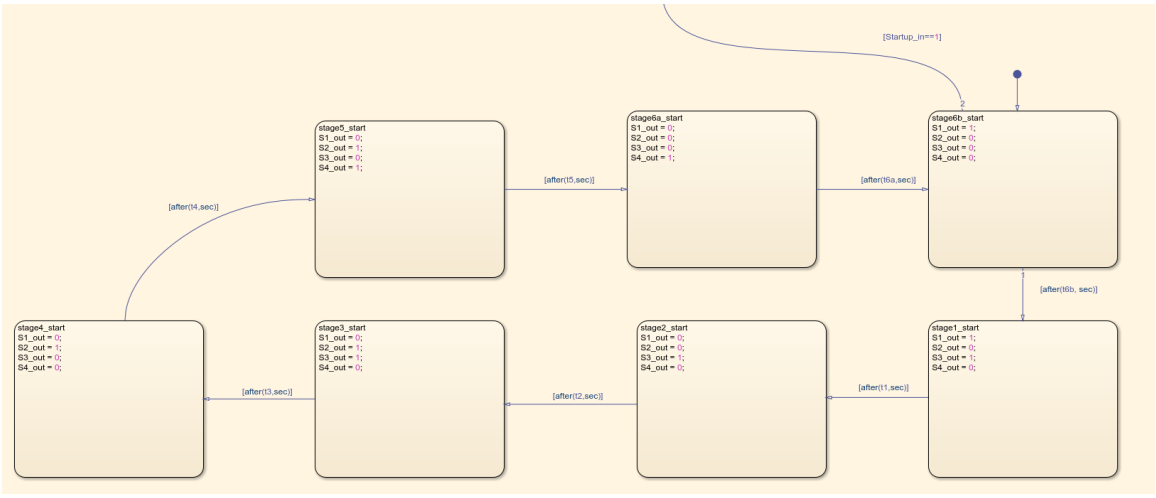


Figure E-4: Startup FSM Diagram. Implements open loop switching times defined as constants in the Simulink model explorer window.

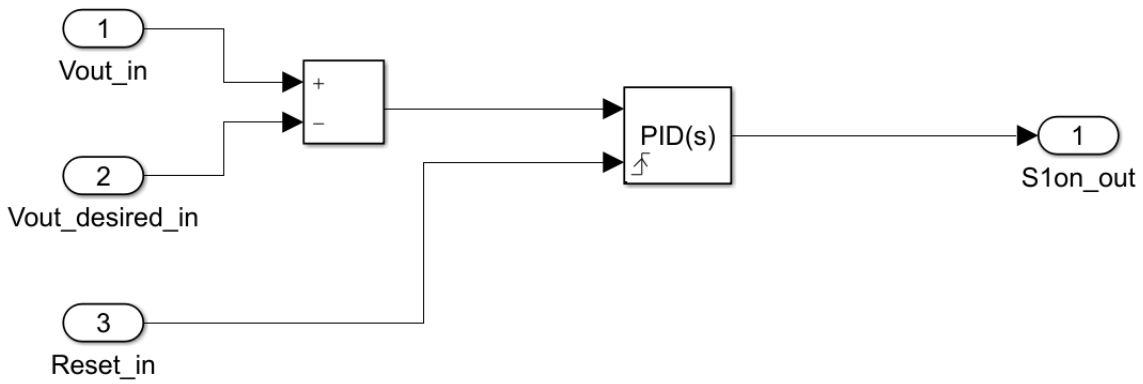


Figure E-5: $S1_{on}$ Feedback Schematic. Implements a PI loop driving the error in V_{out} to 0.

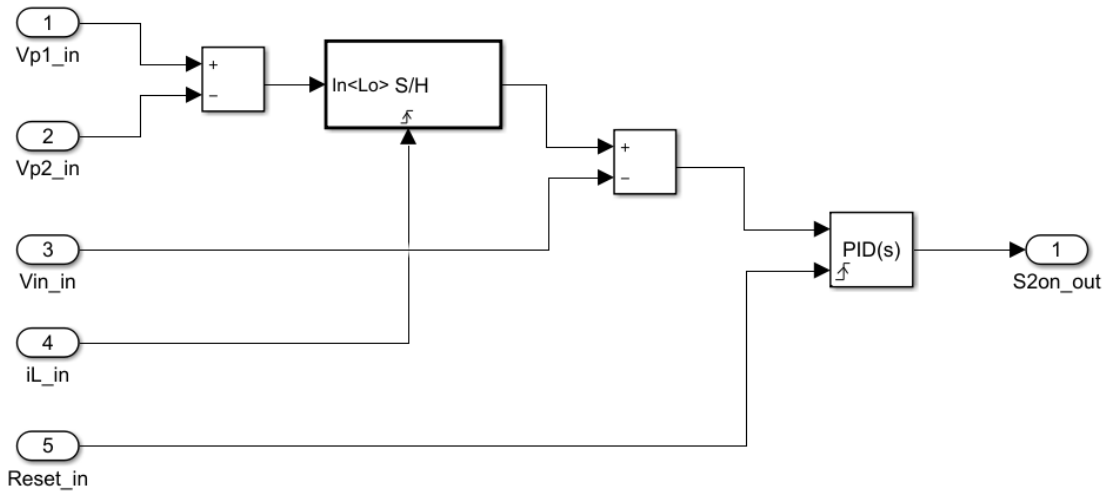


Figure E-6: $S2_{on}$ Feedback Schematic. Implements a PI loop ensuring ZVS is reached across S1. The sample and hold (S/H) block used used to sample v_p when S1 turns on.

E.2 Static Control Simulink Simulation

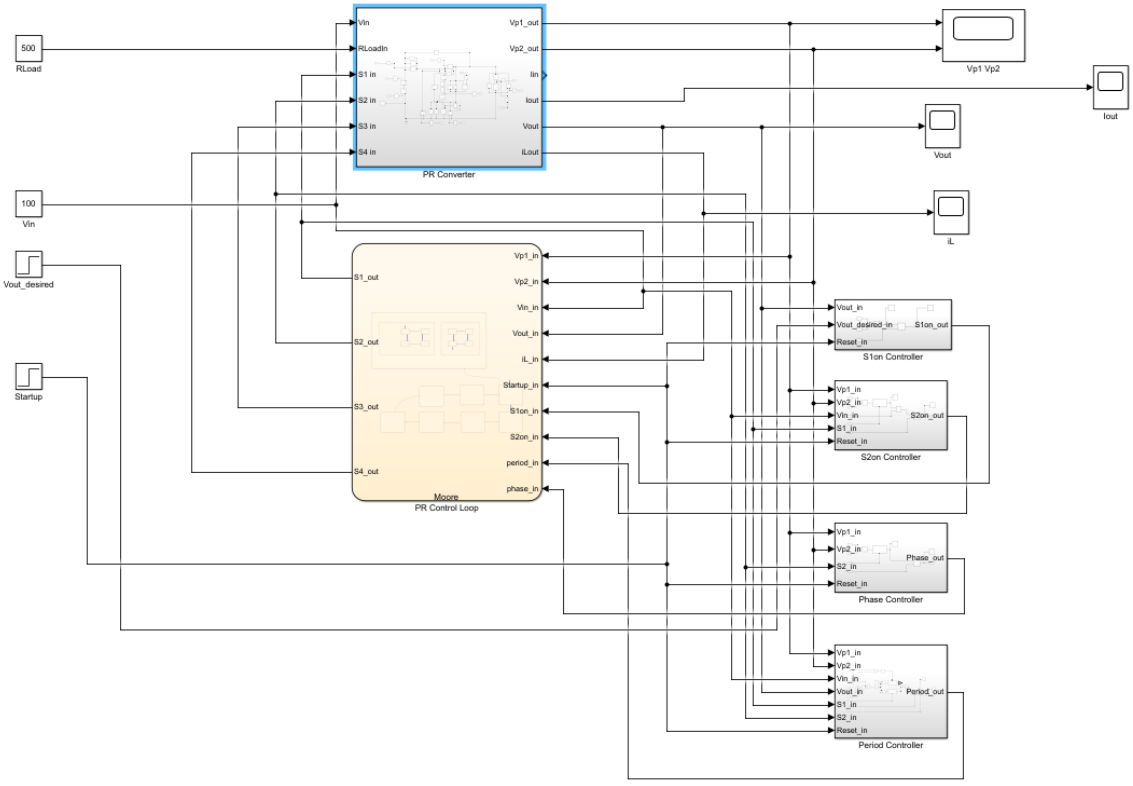


Figure E-7: Top Level Schematic. Integrates the circuit, switch controller FSM, and feedback loops.

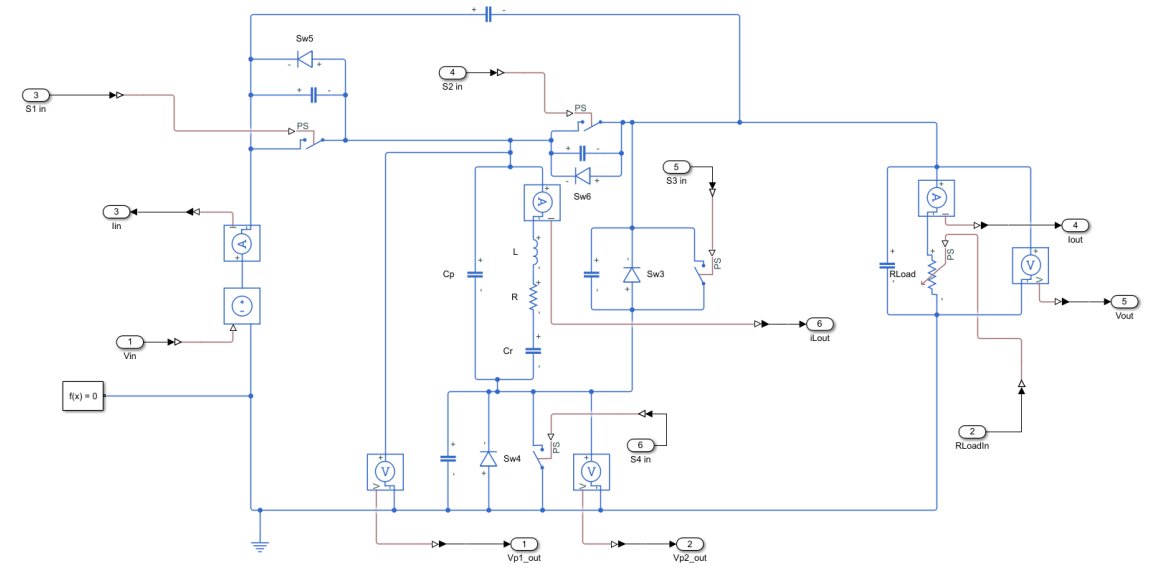


Figure E-8: Circuit Schematic. Implements the topology capable of realizing the$V_{in} - V_{out}$, $Zero$, V_{out} switching sequence.

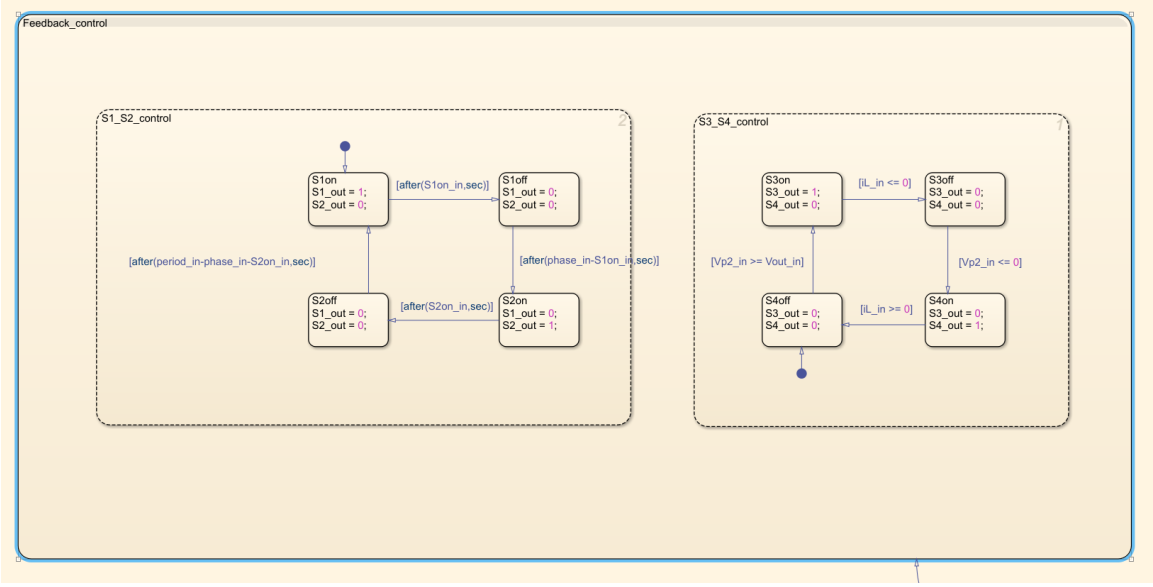


Figure E-9: Switch Control FSM Diagram. Implements the control conditions for static control described in Chapter 5.

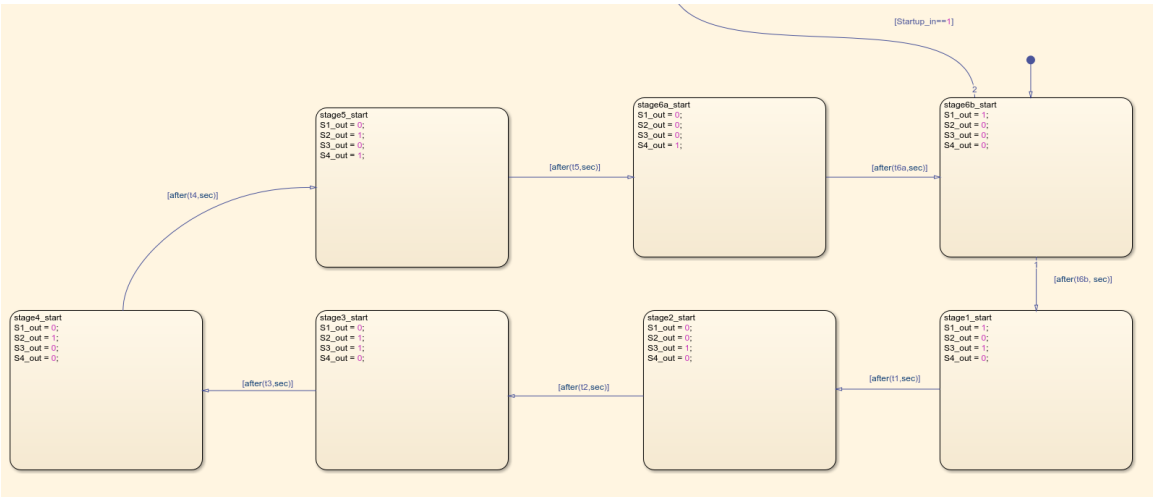


Figure E-10: Startup FSM Diagram. Implements open loop switching times defined as constants in the Simulink model explorer window.

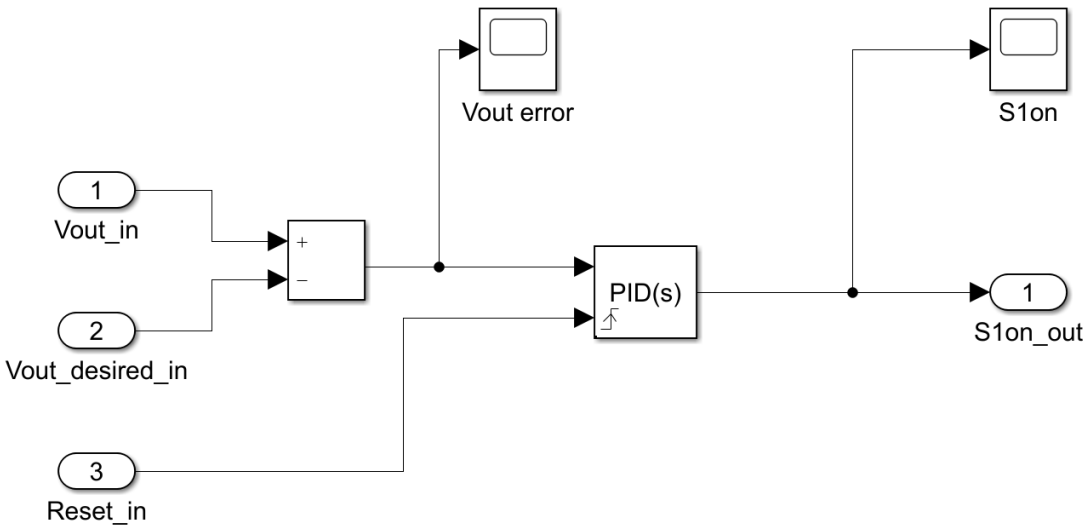


Figure E-11: $S1_{on}$ (RP_{on} Feedback Schematic. Implements a PI loop driving the error in V_{out} to 0.

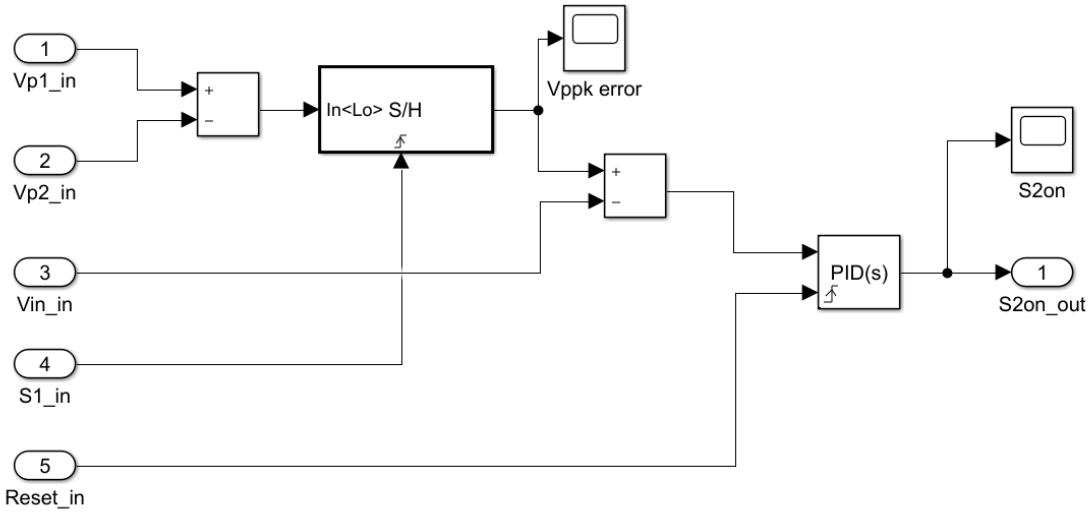


Figure E-12: $S2_{on}$ (RP_{dt}) Feedback Schematic. Implements a PI loop ensuring ZVS is reached across S1. The sample and hold (S/H) block used used to sample v_{p1} when S1 turns on. This is an outdated variable name and definition, and serves the function of implementing RP_{dt} control for ZVS of RP (S1).

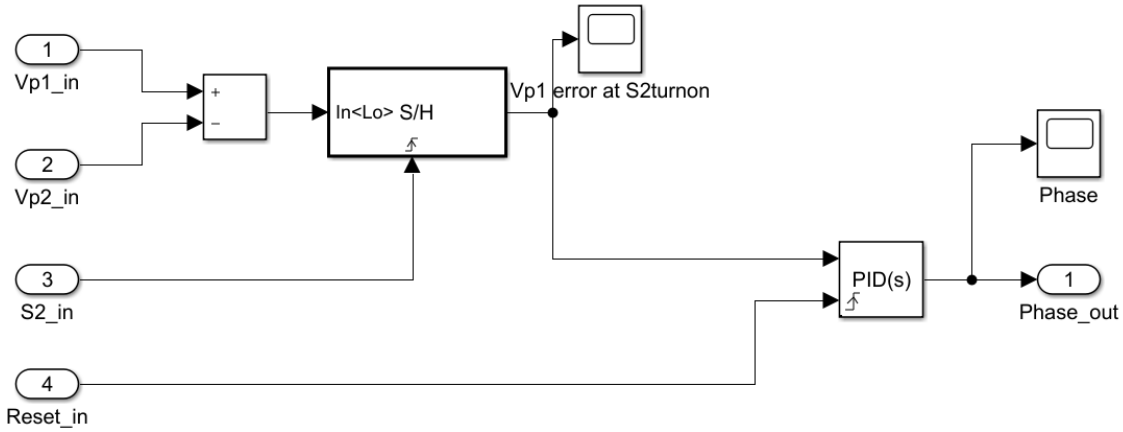


Figure E-13: Phase (RS_{dt}) Feedback Schematic. Implements a PI loop ensuring ZVS is reached across S2. The sample and hold (S/H) block used used to sample v_{p1} when S2 turns on. This is an outdated variable name and definition, and serves the function of implementing RS_{dt} control for ZVS of RS (S2).

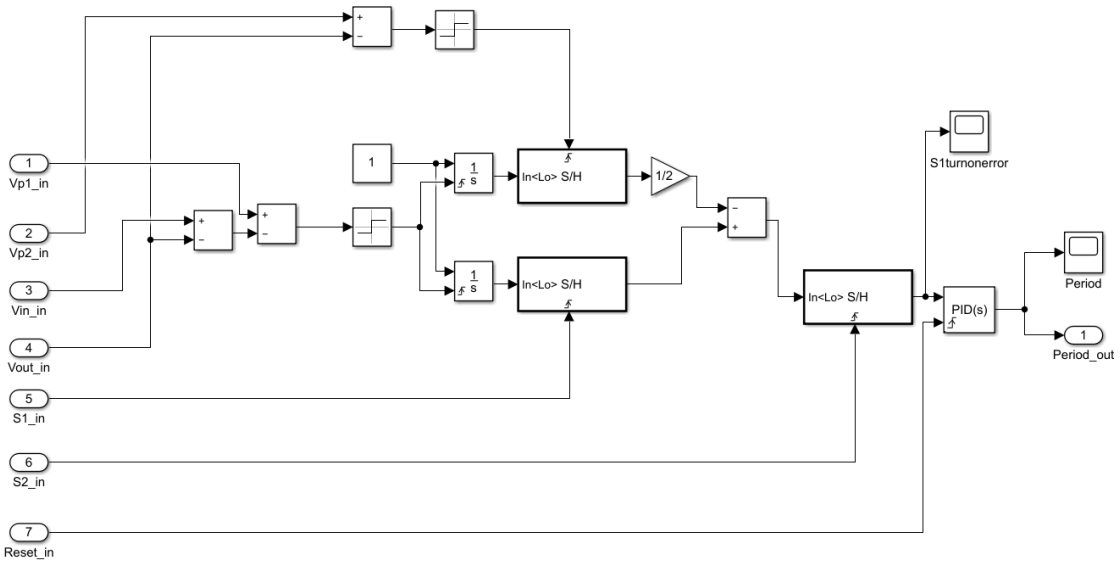


Figure E-14: T Feedback Schematic. Implements a version of the ZCD. Integrator modules that integrate 1 are used as timers, and S/H modules are used to capture t_α and t_β.

Appendix F

Piecewise Linear Dynamic Simulation Code

This appendix presents the MATLAB code used to compute the piecewise linear dynamic simulation of the PR converter operating with the $V_{in} - V_{out}$, $Zero$, V_{out} switching sequence with $V_{out} < \frac{1}{2}V_{in}$ under closed-loop control. The specifiable parameters include simulation step count, PR parameters, the output capacitance C_{out} and load resistance R_{load}, input voltage V_{in} and desired output voltage $V_{out,desired}$, and feedback coefficients $K_{p,S1_{on}}$ and $K_{i,S1_{on}}$. The script will repeatedly evaluate the CoC and CoE equations then compute the new $S1_{on}$ using feedback until the total number of steps is reached. After running, the script will output plots of the time domain PR waveforms and switch control values. The control uses $S1_{on}$ control only, and $S2_{on}$ is automatically solved for to ensure ZVS is reached at the start of stage 6B.

```
%Simulation of ideal converter cycle by cycle w/ feedback control
%Blows up to infinity, possibly because of rounding errors.
%see model 6 for better implementation

simulation_length = 300000;

%PR component values
%1553
Cp = 1.41e-9;
```

```
Cr = 510e-12;
L = 8.73e-3;
R = 2.30;

Ceff = Cp*Cr/(Cp+Cr);
Tr = sqrt(L*Cr)*2*pi;
Tar = sqrt(L*Cp*Cr/(Cp+Cr))*2*pi;

%Converter component values
Cout = 160e-6;
Rload = 600;%500;

%Desired control variables
Vin = 30;%100;
Vout_desired = 8.7;%40;
Vp_pk_desired = Vin;


%start
Vout_start = Vout_desired;
Vp_start = Vin; %Should be Vin in steady state
Vr_start = -210.58824; %Should be something (negative?) in SS
iL0 = 0; %Should be zero by current constraints

%Compute a switching cycle
S1on_int = 1.861764e-06 + 1.267914e-06;
S2on_int = 1.558095e-06 + 1.892829e-06 + 1.925979e-06;

K_p_S1on = 0;%-.25 * 1e-6;
K_i_S1on = -8.9728e-07;%-500 * 1e-6;

K_p_S2on = 0;%.02 * 1e-6;
K_i_S2on = 12  * 1e-6;


%States
```

```
%1 - Vp
%2 - Vr
%3 - Vout

states = zeros(3,simulation_length);
states(:,1) = [Vp_start; Vr_start; Vout_start];

%Data
%1 - S1on
%2 - Stage1time
%3 - S2on
%4 - Stage2time
%

data = zeros(6,simulation_length);


fprintf("Starting Simulation\n");

for i = 1:simulation_length

    %fprintf("%d",i);

    %Voltage command step simulation
    if i >= simulation_length/2
       %Vout_desired = Vout_desired + 1;
       Rload = 300;
    end

    %Setup previous states
    Vp0 = states(1,i);
    Vr0 = states(2,i);
    Vout = states(3,i);

    %Compute this cycle's switching times from previous states
    S1on = K_p_S1on*(states(3,i)-Vout_desired) + S1on_int;
```

```
82      S2on = K_p_S2on*(states(1,i)-Vp_pk_desired) + S2on_int;
83
84      %Start computing cycle
85
86      %Stage 6b
87      Vp1 = Vin-Vout;
88      Vr1 = Vr0 - Cp/Cr*(Vp1-Vp0);
89      iL1 = sqrt(1/L*(Cp*Vp0^2 + Cr*Vr0^2 + L*iL0^2 - Cp*Vp1^2 - Cr*
        Vr1^2));
90
91      theta6b = atan2(norm(cross([(Vp0-Vr0),iL0*sqrt(L/Ceff),0],[(Vp1-
        Vr1),iL1*sqrt(L/Ceff),0])),dot([(Vp0-Vr0),iL0*sqrt(L/Ceff),0],[(
        Vp1-Vr1),iL1*sqrt(L/Ceff),0]));
92      t6b = theta6b*sqrt(L*Ceff);
93
94
95      %stage 1
96      stage1time = S1on-t6b;
97
98      Vp2 = Vin-Vout;
99      Vr2 = Vp1 + iL1*sqrt(L/Cr)*sin(1/sqrt(L*Cr)*stage1time) + (Vr1-
        Vp1)*cos(1/sqrt(L*Cr)*stage1time);
100     iL2 = iL1*cos(1/sqrt(L*Cr)*stage1time) - (Vr1-Vp1)*sqrt(Cr/L)*
        sin(1/sqrt(L*Cr)*stage1time);
101
102     theta1 = atan2(norm(cross([(Vr1-Vp1),iL1*sqrt(L/Cr),0],[(Vr2-Vp2
        ),iL2*sqrt(L/Cr),0])),dot([(Vr1-Vp1),iL1*sqrt(L/Cr),0],[(Vr2-Vp2)
        ,iL2*sqrt(L/Cr),0]));
103     t1 = theta1*sqrt(L*Cr);
104
105
106     %stage 2
107     Vp3 = 0;
108     Vr3 = Vr2 - Cp/Cr*(Vp3-Vp2);
109     iL3 = sqrt(1/L*(Cp*Vp2^2 + Cr*Vr2^2 + L*iL2^2 - Cp*Vp3^2 - Cr*
        Vr3^2));
```

```
110
111     theta2 = atan2(norm(cross([(Vp2-Vr2),iL2*sqrt(L/Ceff),0],[(Vp3-
        Vr3),iL3*sqrt(L/Ceff),0])),dot([(Vp2-Vr2),iL2*sqrt(L/Ceff),0],[(
        Vp3-Vr3),iL3*sqrt(L/Ceff),0]));
112     t2 = theta2*sqrt(L*Ceff);
113
114
115     %Stage 3
116     Vp4 = 0;
117     iL4 = 0;
118     Vr4 = sqrt(1/Cr*(Cp*Vp3^2 + Cr*Vr3^2 + L*iL3^2 - Cp*Vp4^2 - L*
        iL4^2));
119
120     theta3 = atan2(norm(cross([(Vr3-Vp3),iL3*sqrt(L/Cr),0],[(Vr4-Vp4
        ),iL4*sqrt(L/Cr),0])),dot([(Vr3-Vp3),iL3*sqrt(L/Cr),0],[(Vr4-Vp4)
        ,iL4*sqrt(L/Cr),0]));
121     t3 = theta3*sqrt(L*Cr);
122
123     %Stage 4
124     Vp5 = Vout;
125     Vr5 = Vr4 - Cp/Cr*(Vp5-Vp4);
126     iL5 = -sqrt(1/L*(Cp*Vp4^2 + Cr*Vr4^2 + L*iL4^2 - Cp*Vp5^2 - Cr*
        Vr5^2)); %Negative on the square root since in the negative iL
        region
127
128     theta4 = atan2(norm(cross([(Vp4-Vr4),iL4*sqrt(L/Ceff),0],[(Vp5-
        Vr5),iL5*sqrt(L/Ceff),0])),dot([(Vp4-Vr4),iL4*sqrt(L/Ceff),0],[(
        Vp5-Vr5),iL5*sqrt(L/Ceff),0]));
129     t4 = theta4*sqrt(L*Ceff);
130
131     %Stage 5
132     stage5time = S2on - t3 - t4;
133
134     Vp6 = Vout;
135     Vr6 = Vout - ((Cr*Vout^2 - Cp*Vout^2 - Cp*Vin^2 + Cr*Vr5^2 + L*
        iL5^2 + 2*Cp*Vin*Vout - 2*Cr*Vout*Vr5)/Cr)^(1/2) + (Cp*Vin - Cp*
```

```
Vout)/Cr;
136  %iL6 = -((Cp*(Vin - Vout)*(Cr*Vin - Cp*Vin - 2*Cr*(Vout + ((Cr*
     Vout^2 - Cp*Vout^2 - Cp*Vin^2 + Cr*Vr5^2 + L*iL5^2 + 2*Cp*Vin*
     Vout - 2*Cr*Vout*Vr5)/Cr)^(1/2)) + Cp*Vout + Cr*Vout))/(Cr*L))
     ^(1/2);
137  iL6 = -sqrt(1/L*(Cr*(Vr5-Vp5)^2 + L*iL5^2 - Cr*(Vr6-Vp6)^2));
138
139  theta5 = atan2(norm(cross([(Vr5-Vp5),iL5*sqrt(L/Cr),0],[(Vr6-Vp6
     ),iL6*sqrt(L/Cr),0])),dot([(Vr5-Vp5),iL5*sqrt(L/Cr),0],[(Vr6-Vp6)
     ,iL6*sqrt(L/Cr),0]));
140  t5 = theta5*sqrt(L*Cr);
141
142  %Stage 6
143  iL_next = 0;
144  Vp_next = Vin;
145  Vr_next = Vout - ((Cr*Vout^2 - Cp*Vout^2 - Cp*Vin^2 + Cr*Vr5^2 +
     L*iL5^2 + 2*Cp*Vin*Vout - 2*Cr*Vout*Vr5)/Cr)^(1/2);
146
147  theta6a = atan2(norm(cross([(Vp6-Vr6),iL6*sqrt(L/Ceff),0],[(
     Vp_next-Vr_next),iL_next*sqrt(L/Ceff),0])),dot([(Vp6-Vr6),iL6*
     sqrt(L/Ceff),0],[(Vp_next-Vr_next),iL_next*sqrt(L/Ceff),0]));
148  t6a = theta6a*sqrt(L*Ceff);
149
150
151
152  Vp = [Vp0 Vp1 Vp2 Vp3 Vp4 Vp5 Vp6 Vp_next]';
153  Vr = [Vr0 Vr1 Vr2 Vr3 Vr4 Vr5 Vr6 Vr_next]';
154  iL = [iL0 iL1 iL2 iL3 iL4 iL5 iL6 iL_next]';
155
156  %Compute iout
157  q1 = Cr*(Vr2-Vr1);
158  q3 = Cr*(Vr4-Vr3);
159  q5 = -Cr*(Vr6-Vr5);
160
161  charge = [q1;q3;q5];
162
```

```
    % Solve for times
    times = [t6b;t1;t2;t3;t4;t5;t6a];
    T = sum(times);

    %Compute Vout change
    Vout_next = Vout + 1/Cout*(q1 + q5 - Vout/Rload*sum(times));

    S1on_int = S1on_int + K_i_S1on*(states(3,i)-Vout_desired)*T;
    S2on_int = S2on_int + K_i_S2on*(states(1,i)-Vp_pk_desired)*T;

    %K
    K = (q1+q5)/(q1+q3+q5);

    states(:,i+1) = double([Vp_next; Vr_next; Vout_next]);

    data(:,i+1) = [S1on;stage1time;t3+t4+t5;t5;T;K];

    if any(imag(states(:,i+1)) ~= 0)
        fprintf("Bad cycle, imaginary parts\n")
        break
    end


end

figure(1)
plot(cumsum(data(5,1:simulation_length)),states(:,1:
    simulation_length))
figure(2)
plot(cumsum(data(5,1:simulation_length)),data([1,3],1:
    simulation_length))
figure(3)
plot(cumsum(data(5,1:simulation_length)),data(5,1:simulation_length)
    )
figure(4)
```

```
195 plot(cumsum(data(5,1:simulation_length)),data(6,1:simulation_length)
    )
196
197 fprintf("\n")
```

Appendix G

State Space Dynamic Model Code

This appendix presents the MATLAB code used to compute the linearized state space dynamic simulation of the PR converter operating with the $V_{in} - V_{out}$, $Zero$, V_{out} switching sequence with $V_{out} < \frac{1}{2}V_{in}$ under closed-loop control. The specifiable parameters include PR parameters, the output capacitance C_{out} and load resistance R_{load}, input voltage V_{in} and desired output voltage $V_{out,desired}$, and feedback coefficients $K_{p,S1_{on}}$ and $K_{i,S1_{on}}$. The control uses $S1_{on}$ control only, and $S2_{on}$ is automatically solved for to ensure ZVS is reached at the start of stage 6B. The script defines the state equations symbolically and differentiates them to obtain the linearized equations. The linearized equations are then used to create a MATLAB state space model, which can be analyzed using MATLAB's suite of control theory functions and programs. The script will compute and plot the step response from a stem in desired output voltage, and the model is saved for further analysis.

```
1 syms Vout IL S1on Rload
2
3 %PR component values
4 %1553
5 Cp = 1.41e-9;
6 Cr = 510e-12;
7 L = 8.73e-3;
8 R = 2.3;
9
```

```
Tr = sqrt(L*Cr)*2*pi;
Tar = sqrt(L*Cp*Cr/(Cp+Cr))*2*pi;

%Converter component values
Cout = 16e-6;
Rload_bar = 600;

%Desired control variables
Vin = 30;
Vout_desired = 8.5;

T = 12.96e-6;%12.23e-6;
S1on_bar = 3.13e-6;

q1 = T*IL/(2*pi)*(1-cos(2*pi/T*S1on)) - Cp*Vout;
q3 = T*IL/pi - Cp*Vin - q1;
q5 = T*IL/pi - Cp*Vin;

d_Vout_d_t = 1/(T*Cout) * (q1 + q5 - T*Vout/Rload);
d_IL_d_t = 1/(T*L*IL) * ((Vin-Vout)*q1 - Vout*q5);

%Fix S1on_bar for equilibrium
%eqns = subs([d_Vout_d_t;d_IL_d_t],S1on,S1on_bar);
%out = solve(eqns);

%Vout_bar = vpa(out.Vout(1));
%iL_bar = vpa(out.IL(1));

%Fix Vout_desired for equilibrium
eqns = subs([d_Vout_d_t;d_IL_d_t],[Vout,Rload],[Vout_desired,
    Rload_bar]);
out = solve(eqns);

Vout_bar = Vout_desired;
S1on_bar = T-vpa(out.S1on(1));
iL_bar = vpa(out.IL(1));
```

```
45
46 %Fix iL_bar for equilibrium
47 %eqns = subs([d_Vout_d_t;d_IL_d_t],IL,18.7*1.01);
48 %out = solve(eqns);
49
50 %Vout_bar = vpa(out.Vout(1));
51 %S1on_bar = vpa(out.S1on(1));
52
53
54 A_converter_1 = [diff(d_Vout_d_t,Vout), diff(d_Vout_d_t,IL);
55                  diff(d_IL_d_t,Vout), diff(d_IL_d_t,IL)];
56
57
58 A_converter_2 = subs(A_converter_1,[Vout,IL,S1on,Rload],[Vout_bar,
    iL_bar, S1on_bar, Rload_bar]);
59
60 B_converter_1 = [diff(d_Vout_d_t,S1on), diff(d_Vout_d_t,Rload);
61                  diff(d_IL_d_t,S1on), diff(d_IL_d_t,Rload)];
62
63 B_converter_2 = subs(B_converter_1,[Vout,IL,S1on,Rload],[Vout_bar,
    iL_bar, S1on_bar, Rload_bar]);
64
65 C_converter_1 = [1,0];
66
67 C_converter_2 = subs(C_converter_1,[Vout,IL,S1on,Rload],[Vout_bar,
    iL_bar, S1on_bar, Rload_bar]);
68
69 D_converter_1 = [0, 0];
70
71 D_converter_2 = subs(D_converter_1,[Vout,IL,S1on,Rload],[Vout_bar,
    iL_bar, S1on_bar, Rload_bar]);
72
73
74 %State space model of PR Converter
75 %Inputs: S1on
76 %Outputs: Vout
```

```
77 %States: Vout, IL
78
79 converter_state_names = {'Vout','IL'};
80 converter_input_names = {'S1on', 'Rload'};
81 converter_output_names = {'Vout'};
82
83 A_converter = double(A_converter_2);
84
85 B_converter = double(B_converter_2);
86
87 C_converter = double(C_converter_2);
88
89 D_converter = double(D_converter_2);
90
91 converter_model = ss(A_converter, B_converter, C_converter,
       D_converter);
92 converter_model.StateName = converter_state_names;
93 converter_model.InputName = converter_input_names;
94 converter_model.OutputName = converter_output_names;
95
96 converter_model_vout = ss(A_converter, B_converter(:,1), C_converter
       , D_converter(:,1));
97 converter_model_vout.StateName = converter_state_names;
98 converter_model_vout.InputName = {'S1on'};
99 converter_model_vout.OutputName = converter_output_names;
100
101 % converter_model_rload = ss(A_converter, B_converter(:,2),
       C_converter, D_converter(:,2));
102 % converter_model_rload.StateName = converter_state_names;
103 % converter_model_rload.InputName = converter_input_names;
104 % converter_model_rload.OutputName = converter_output_names;
105
106 %Controller model
107 %Basic controller, only Vout feedback
108 controller_input_names = {'Vout error','Rload'};
109 controller_output_names = {'S1on','Rload'};
```

```
controller_state_names = {'Vout int'};

k_p_S1on = 0;%.25 * 1e-6;
k_i_S1on = 8.9728e-06;%500 * 1e-6;

k_p_S2on = -.02 * 1e-6;
k_i_S2on = -12  * 1e-6;

A_controller = 0;

B_controller = [1, 0];

C_controller = [k_i_S1on;
                0];

D_controller = [k_p_S1on, 0;
                0, 1];

controller_model = ss(A_controller, B_controller, C_controller,
   D_controller);
controller_model.InputName = controller_input_names;
controller_model.OutputName = controller_output_names;
controller_model.StateName = controller_state_names;

%Feedback loop
plant = series(controller_model,converter_model);
converter_cl = feedback(plant,1,[1],[1],1);

opt = stepDataOptions('StepAmplitude',-300);
step(converter_cl,opt)
```

Appendix H

Microcontroller Code

This appendix presents the C code used to program the TI TMDSCNCD28379D Control Card microcontroller. The code is designed to be used with the specific sensing circuitry and IO described in Chapter 7, however, it can be reconfigured. The code implements static control, but the code for initializing ePWMs in one-shot mode is present, allowing reconfiguring the code to implement sensed control if desired.

The easily configurable aspects of the code include the switch designations (maps RP, RS, and so on to specific ePWMs), the startup switching times, the feedback control coefficients, the desired output voltage, and any necessary correction terms. More information about all of these can be found in the code comments.

H.1 Main Code

```
1 //PR Controller - Sensing 3
2 //Joshua Piel
3 //11-17-21
4 //
5
6 //Set up for Vin-Vout,Zero,Vout or Vin,Vin-Vout,Vout where Vin >
     Vout > 1/2*Vin
7 //Static control
8 //Synchronous control
```

```
9
10 //Code marked with MODIFY is safe to edit to change operation.
11
12
13
14 //
   ##############################################################################

15 //
16 // FILE:   epwm_ex12_monoshot_mode.c
17 //
18 // TITLE:  Realization of Monoshot mode
19 //
20 //! \addtogroup driver_example_list
21 //! <h1>Realization of Monoshot mode</h1>
22 //!
23 //! This example showcases how to generate monoshot PWM output based
       on external
24 //! trigger i.e. generating just a single pulse output on receipt of
       an external
25 //! trigger. And the next pulse will be generated only when the next
       trigger
26 //! comes. The example utilizes external synchronization and T1
   action qualifier
27 //! event features to achieve the desired output.
28 //!
29 //! ePWM1 is used to generate the monoshot output and ePWM2 is used
   an external
30 //! trigger for that. No external connections are required as ePWM2A
       is fed
31 //! as the trigger using Input X-BAR automatically.
32 //!
33 //! ePWM1 is configured to generated a single pulse of 0.5us when
   received
34 //! an external trigger. This is achieved by enabling the phase
   synchronization
```

```
35 //! feature and configuring EPWMxSYNCI as EXTSYNCIN1. And this
       EPWMxSYNCI
36 //! is also configured as T1 event of action qualifier to set output
        HIGH while
37 //! "CTR = PRD" action is used to set output LOW.
38 //!
39 //! ePWM2 is configured to generate a 100 KHz signal with a duty of
       1% (to
40 //! simulate a rising edge trigger) which is routed to EXTSYNCIN1
       using Input XBAR.
41 //!
42 //! Observe GPIO6 (EPWM4A : Monoshot Output) and GPIO2(EPWM2 :
       External Trigger)
43 //! on oscilloscope.
44 //!
45 //!
46 //! \b NOTE : In the following example, the ePWM timer is still
       running in a
47 //!         continuous mode rather than a one-shot mode thus for more
        reliable
48 //!         implementation, refer to CLB based one shot PWM
       implementation
49 //!         demonstrated in "clb_ex17_one_shot_pwm" example
50 //
51 //
52 //
       ############################################################################


53 // $TI Release: F2837xD Support Library v3.12.00.00 $
54 // $Release Date: Fri Feb 12 19:03:23 IST 2021 $
55 // $Copyright:
56 // Copyright (C) 2013-2021 Texas Instruments Incorporated - http://
       www.ti.com/
57 //
58 // Redistribution and use in source and binary forms, with or
       without
```

```
59 // modification, are permitted provided that the following
      conditions
60 // are met:
61 //
62 //   Redistributions of source code must retain the above copyright
63 //   notice, this list of conditions and the following disclaimer.
64 //
65 //   Redistributions in binary form must reproduce the above
      copyright
66 //   notice, this list of conditions and the following disclaimer in
       the
67 //   documentation and/or other materials provided with the
68 //   distribution.
69 //
70 //   Neither the name of Texas Instruments Incorporated nor the
      names of
71 //   its contributors may be used to endorse or promote products
      derived
72 //   from this software without specific prior written permission.
73 //
74 // THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
      CONTRIBUTORS
75 // "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
76 // LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
       FOR
77 // A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
      COPYRIGHT
78 // OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
      INCIDENTAL,
79 // SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
80 // LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
      USE,
81 // DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
       ANY
82 // THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
      TORT
```

```
83 // (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
       USE
84 // OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
       DAMAGE.
85 // $
86 //
       ############################################################################
87
88 //
89 // Included Files
90 //
91 #include "driverlib.h"
92 #include "device.h"
93 #include "board.h"
94 #include "clb_config.h"
95 #include "clb.h"
96
97
98
99 //Defines which ePWM fulfills which conceptual switch
100 //MODIFY to fit switching sequence and setup
101 #define RP EPWM4_BASE
102 #define RS EPWM3_BASE
103 #define NP EPWM1_BASE
104 #define NS EPWM2_BASE
105
106
107 //FORWARD marks how the primary and secondary half bridges are
       oriented around the open stage zero crossing
108 //If FORWARD is true, then the nonregulating primary turns off and
       the regulating primary turns on at the zero crossing
109 //This is the same as "Mode 1"
110 //If FORWARD is false, then the regulating primary turns off and the
       nonregulating primary turns on at the zero crossing
111 //This is the same as "Mode 2"
```

```
112 #define FORWARD false
113
114
115
116 //
117 // Function Prototypes
118 //
119 void initEPWM_fixed(uint32_t);
120 void EPWM_set_timing(uint32_t, uint16_t, uint16_t, uint16_t,
       uint16_t);
121 void update_switches_reg(uint32_t, uint32_t, uint16_t, uint16_t,
       uint16_t, uint16_t);
122 void update_switches_nonreg(uint32_t, uint32_t, uint16_t, uint16_t,
       int16_t);
123
124 void configureADC(uint32_t);
125
126 void setupADCTriggered(uint32_t, uint32_t, ADC_SOCNumber,
       ADC_Trigger, ADC_IntNumber);
127
128 void initCLB_ZERO_CROSSING_TIMER(uint32_t);
129
130
131 __interrupt void feedback_control_ISR();
132
133
134 void initCMPSS(uint32_t, uint16_t, bool);
135
136 bool feedback_control_enabled = false;
137 bool trigger_safety_enabled = false;
138
139 //Startup Open Loop Switching Times
140 //MODIFY the following macros with the default switching values
141 //These can be computed using the matlab solver
142
143 //600 Ohm
```

```
144 //Standard
145 /*
146 #define DEF_PERIOD 1268//12.87us*100*.985
147 #define DEF_S1ON 337//3.42us*100*.985
148 #define DEF_S2ON 668//6.78us*100*.985
149 #define DEF_PHASE 418//4.24us*100*.985
150 */
151
152 /*
153 //600 Ohm
154 //1553 piezo
155 //Standard 30V->18V
156 #define DEF_PERIOD 1290//12.87us*100*.985
157 #define DEF_RPON 369-50//3.42us*100*.985
158 #define DEF_RPDT 43+20//6.78us*100*.985
159 #define DEF_RSDT 131+50//4.24us*100*.985
160 #define DEF_NPDT 106//4.3us*100*.985
161 */
162
163 //186c piezo
164 //Standard 30V->18V
165 #define DEF_PERIOD 203*2//12.87us*100*.985
166 #define DEF_RPON 54*2//3.42us*100*.985
167 #define DEF_RPDT 20*2//6.78us*100*.985
168 #define DEF_RSDT 25*2//4.24us*100*.985
169 #define DEF_NPDT 33*2//4.3us*100*.985
170
171
172 /*
173 //original (30V->12V and/or 700 ohm??)
174 uint16_t period = 1275;//12.75us
175 uint16_t sw1on = 360;//3.60us
176 uint16_t sw2on = 610;//6.10us
177 uint16_t phase = 114;//1.14us
178 */
179
```

```
180
181 //Default values of the integrators are the open loop switching
       times
182 int32_t period_integral = ((long)DEF_PERIOD)<<16;
183 int32_t rpon_integral = ((long)DEF_RPON)<<16;
184 int32_t rpdt_integral = ((long)DEF_RPDT)<<16;
185 int32_t rsdt_integral = ((long)DEF_RSDT)<<16;
186 int32_t npdt_integral = ((long)DEF_NPDT)<<16;
187
188
189 //Define measurement variables as globals so they can be accessed in
        the debug window
190 uint16_t current_vout_raw;
191 uint16_t current_vpr_before_rp_raw;
192 uint16_t current_vpr_before_rs_raw;
193 uint16_t current_vpn_before_np_raw;
194
195 uint32_t current_vout;
196 uint32_t current_vpr_before_rp;
197 uint32_t current_vpr_before_rs;
198 uint32_t current_vpn_before_np;
199
200 //Defines the ADC measurement digital filter
201 //MODIFY to change the coefficient of the filter. Need to enable
       more code in the feedback loop
202 uint32_t alpha = 0;//65126;//Cutoff frequency at 1/10 switching
       frequency
203 uint32_t one_minus_alpha = 65536;//65536-65126;
204
205 //Defines ZCD outputs
206 uint32_t t_alpha;
207 uint32_t t_beta;
208
209
210 //Defines error variables
211 int32_t error_vout;
```

```
212 int32_t error_vpr_before_rp;
213 int32_t error_vpr_before_rs;
214 int32_t error_vpn_before_np;
215 int32_t error_zero_crossing_offset;
216
217 //Defines current switching times, which are computed by the
      feedback loops every cycle
218 uint16_t current_rpon;
219 uint16_t current_rpdt;
220 uint16_t current_rsdt;
221 uint16_t current_npdt;
222 uint16_t current_period;
223
224
225 //Defines ADC correction terms.
226 //This arises because of time inaccuracies in the ADC measurement
227 //As well as inability to perfectly measure the switch nodes for ZVS
228 //Adds a linear offset to the ZVS feedback loops to manually tune
      ZVS control
229 //MODIFY based on observations to get the waveform to line up
230 //Reasonable values at 100kHZ are about plus/minus 100
231 //Extreme values were needed at 500kHz
232 int32_t VPR_BEFORE_RP_CORRECTION = 900;//850;
233 int32_t VPR_BEFORE_RS_CORRECTION = -900;//-800;
234 int32_t VPN_BEFORE_NP_CORRECTION = -50;//-100;
235
236
237 //Defines nonregulating half bridge duty cycle deviation from 50%
238 //Arises when the 50% duty cycle approximation breaks down
239 //MODIFY to line up the nonregulating HB with both iL 0 crossings
240 int16_t DUTY_CORRECTION = -3*2;
241
242 //Defines a correction to the ZCD feedback loop
243 //Arises because t_alpha ends up not equalling exactly 1/2 t_beta
244 //MODIFY to line up the RP turn on (mode 1) or turn off (mode 2)
245 //  with the zero crossing exactly, based on waveform observations
```

```
246 int32_t ZC_CORRECTION = 0;//-20;
247
248
249 //Defines the ADC levels that define 0V, Vin, and the desired output
        voltage
250 //MODIFY to match actual Vin and Zero, and the desired output
       voltage.
251 //ADC is configured to convert 0V-3V Full Scale Range into 12 bits
252 //See sensing circuit for output ranges.
253 int32_t VIN = 2930;
254 int32_t ZERO = 650;
255 int32_t DESIRED_VOUT = 2250; // Desired ADC Measurement. 2000 = 17.5
       V
256
257 //
258 // FEEDBACK COEFFICIENTS
259 // MODIFY as desired to tune the dynamic response
260
261
262 //
263 // FAST RESPONSE RLOAD STEP
264 //
265 /*
266 int32_t K_P_S1ON = -30000;//-5000;//-10000;//-1000
267 int32_t K_INT_S1ON = -40;//-150;//-29; //units of us/2^16, should be
        negative
268
269 int32_t K_P_S2ON = 0;//7000;//100;
270 int32_t K_INT_S2ON = 45;//3; //units of xxxx, should be positive
271
272 int32_t K_P_PHASE = 0;
273 int32_t K_INT_PHASE = 10;//should be positive
274
275 int32_t K_P_PERIOD = -30000;//-30000
276 int32_t K_INT_PERIOD = -500;//-60;//should be negative
277
```

```
int32_t K_P_S3ON = 0;
int32_t K_INT_S3ON = 0;//should be positive
*/

//
// STABLE(ISH?) WITH SYNCHRONOUS at 1553 for vout>1/2vin
//

/*
int32_t K_P_VOUT = 15000;//+
int32_t K_INT_VOUT = 20;

int32_t K_P_ZVS = 0;//+
int32_t K_INT_ZVS = 10;

int32_t K_P_ZC = -5000;//-
int32_t K_INT_ZC = -20;
*/


//
// STABLE(ISH?) WITH SYNCHRONOUS at 186 for vout>1/2vin
//
/*
int32_t K_P_VOUT = 5000;//+
int32_t K_INT_VOUT = 4;

int32_t K_P_ZVS = 0;//+
int32_t K_INT_ZVS = 4;

int32_t K_P_ZC = 0;//-
int32_t K_INT_ZC = -4;
*/

//
// Working Coeffs for SYNCHRONOUS at 186 for vout>1/2vin
```

```
314 //
315
316 int32_t K_P_VOUT = 1000;//10000;//+
317 int32_t K_INT_VOUT = 1;//10;
318
319 int32_t K_P_ZVS = 0;//+
320 int32_t K_INT_ZVS = 1;//4;
321
322 int32_t K_P_ZC = 0;//-
323 int32_t K_INT_ZC = -2;//-40;
324
325 //
326 // SLOW RESPONSE
327 //
328 /*
329 int32_t K_P_S1ON = -0;//-5000;//-10000;//-1000
330 int32_t K_INT_S1ON = -0;//-150;//-29; //units of us/2^16, should be
       negative
331
332 int32_t K_P_S2ON = 0;//7000;//100;
333 int32_t K_INT_S2ON = 1;//3; //units of xxxx, should be positive
334
335 int32_t K_P_PHASE = 0;
336 int32_t K_INT_PHASE = 1;//should be positive
337
338 int32_t K_P_PERIOD = 0;//-30000
339 int32_t K_INT_PERIOD = -1;//-60;//should be negative
340
341 int32_t K_P_S3ON = 0;
342 int32_t K_INT_S3ON = 1;//should be positive
343 */
344
345
346 //FEEDBACK BOUNDS
347 //Sets the minimum and maximum times deat times can be
348 //MODIFY to match a reasonable fraction of the current operating
```

```
   period
uint16_t DEAD_TIME_MIN = 10*2;//50;
uint16_t DEAD_TIME_MAX = 44*2;//220;

//ADC measurement offset from switch turn on
//MODIFY ONLY IF NECESSARY to tell the ADC to start measuring so
   that its
//  sample and hold window finishes just before the switch turns on
uint16_t ADC_MEASUREMENT_DELAY = 12*2;


//Interrupt routine trigger counter
//The interrupt will execute the feedback loop after being called
   COUNT_MAX
//  times, then it resets the count to 0
uint16_t interrupt_count = 0;
//MODIFY if necessary to change the feedback loop frequency
uint16_t COUNT_MAX = 5;



#define EX_ADC_RESOLUTION       12


void main(void)
{
    //
    // Initialize device clock and peripherals
    //
    Device_init();

    //
    // Disable pin locks and enable internal pull-ups.
    //
    Device_initGPIO();

```

```
382     //
383     // Initialize PIE and clear PIE registers. Disables CPU
    interrupts.
384     //
385     Interrupt_initModule();
386
387     //
388     // Initialize the PIE vector table with pointers to the shell
    Interrupt
389     // Service Routines (ISR).
390     //
391     Interrupt_initVectorTable();
392
393     //
394     // Configure ePWM1, ePWM2 GPIOs and XBAR configuration
395     //
396     Board_init();
397
398     //
399     // Disable sync(Freeze clock to PWM as well)
400     //
401     SysCtl_disablePeripheral(SYSCTL_PERIPH_CLK_TBCLKSYNC);
402
403     SysCtl_setEPWMClockDivider(SYSCTL_EPWMCLK_DIV_1);
404
405     //
406     // Initialize ePWM1 and ePWM2
407     //
408
409     initEPWM_fixed(RP);//S1
410     initEPWM_fixed(RS);//S2
411     initEPWM_fixed(NP);//S3
412     initEPWM_fixed(NS);//S4
413
414     EPWM_setSyncOutPulseMode(EPWM1_BASE,
    EPWM_SYNC_OUT_PULSE_ON_COUNTER_ZERO);
```

```
415     SysCtl_setSyncInputConfig(SYSCTL_SYNC_IN_EPWM4,
    SYSCTL_SYNC_IN_SRC_EPWM1SYNCOUT);
416
417     update_switches_reg(RP, RS, DEF_PERIOD, DEF_RPON, DEF_RPDT,
    DEF_RSDT);
418     update_switches_nonreg(NP, NS, DEF_PERIOD, DEF_NPDT,
    DUTY_CORRECTION);
419
420
421     //
422     // Enable CLB1
423     // Configured to time the comparator measurements used to
    compute the S1 trigger offset from the zero crossing
424     //
425     initCLB_ZERO_CROSSING_TIMER(CLB1_BASE);
426
427     //
428     // Enable sync and clock to PWM
429     //
430     SysCtl_enablePeripheral(SYSCTL_PERIPH_CLK_TBCLKSYNC);
431
432
433     //
434     // Configure the comparators
435     //
436     initCMPSS(CMPSS3_BASE, 1100, false);//DESIRED_VOUT, false);//
    Comparator watching Vp1
437     initCMPSS(CMPSS1_BASE, VIN-DESIRED_VOUT+ZERO-100, false);//
    Comparator watching Vp2
438
439     //
440     // Configure the ADC and power it up
441     //
442     configureADC(ADCA_BASE); //Vp2
443     configureADC(ADCB_BASE); //Vp1
444     configureADC(ADCD_BASE); //Vout
```

```
445
446     //Set up which ePWMs trigger which ADCs
447     //MODIFY as necessary to match the switching sequence
448     setupADCTriggered(ADCA_BASE, 2, ADC_SOC_NUMBER0,
    ADC_TRIGGER_EPWM4_SOCA, ADC_INT_NUMBER3); //Measure Vp2 just
    before S4 (RP)
449     setupADCTriggered(ADCA_BASE, 2, ADC_SOC_NUMBER1,
    ADC_TRIGGER_EPWM3_SOCA, ADC_INT_NUMBER4); //Measure Vp2 just
    before S3 (RS)
450     setupADCTriggered(ADCB_BASE, 2, ADC_SOC_NUMBER0,
    ADC_TRIGGER_EPWM1_SOCA, ADC_INT_NUMBER3); //Measure Vp1 just
    before S1 (NS)
451     setupADCTriggered(ADCD_BASE, 0, ADC_SOC_NUMBER0,
    ADC_TRIGGER_EPWM1_SOCA, ADC_INT_NUMBER3); //Measure Vout (just
    before S1, but the specific time doesn't really matter, just
    consistency)
452
453     //Configures which interrupt (in this case ADC conversion)
454     //  finish triggers the feedback control ISR
455     //MODIFY if necessary to change what portion of the cycle
456     //  executes the feedback loop
457     Interrupt_register(INT_ADCA4, &feedback_control_ISR);
458     Interrupt_enable(INT_ADCA4);//B4 originally
459
460
461     //
462     // Enable Global Interrupt (INTM) and real time interrupt (DBGM)
463     //
464     EINT;
465     ERTM;
466
467     GPIO_writePin(myGPIO0, 0);
468
469
470
471     //Wait for the converter to absorb energy, and pause, allowing
```

```
   the user to control transition to automatic
472     DEVICE_DELAY_US(2000006);
473
474     //GPIO_writePin(myGPIO0, 1);
475
476
477     //DEVICE_DELAY_US(100000);
478
479     //GPIO_writePin(myGPIO0, 1);
480
481     //Enables the feedback control after startup
482     feedback_control_enabled = true;
483
484     DEVICE_DELAY_US(2000006>>2);
485
486     //Use this GPIO pin to make a debug signal (scopeable point in
    time)
487     GPIO_writePin(myGPIO0, 1);
488
489
490
491     //
492     // IDLE loop. Just sit and loop forever (optional):
493     //
494
495 /*
496     for(DESIRED_VOUT = 1200; DESIRED_VOUT < 1800; DESIRED_VOUT +=
    50)
497     {
498         DEVICE_DELAY_US(500000);
499     }
500 */
501
502     for(;;)
503     {
504         //Use this code to create repeated voltage steps
```

```
505         /*
506
507         for(DESIRED_VOUT = 1950; DESIRED_VOUT < 2350; DESIRED_VOUT
    += 50)
508         {
509             DEVICE_DELAY_US(500000<<2);
510         }
511         */
512
513         /*
514         DEVICE_DELAY_US(500000);
515         DESIRED_VOUT = 1200;
516         GPIO_writePin(myGPIO0, 1);
517         //GPIO_writePin(myGPIO0, 0);
518
519         DEVICE_DELAY_US(500000);
520         DESIRED_VOUT = 1600;
521         //GPIO_writePin(myGPIO0, 1);
522         GPIO_writePin(myGPIO0, 0);
523         */
524
525         //ADC_clearInterruptStatus(ADCA_BASE, ADC_INT_NUMBER4);
526         //Interrupt_clearACKGroup(INTERRUPT_ACK_GROUP10);
527         DEVICE_DELAY_US(500000);
528
529     }
530 }
531
532
533 //Configures the ePWM modules as described in thesis ch7 for static
    control
534 void initEPWM_fixed(uint32_t epwm_base)
535 {
536     //
537     // Setting counter as 0
538     //
```

```
539     EPWM_setTimeBaseCounter(epwm_base, 0U);
540
541     //
542     // Configuring the counter in up mode
543     //
544     EPWM_setTimeBaseCounterMode(epwm_base, EPWM_COUNTER_MODE_UP);
545
546     //
547     // Set ePWM clock pre-scaler
548     //
549     EPWM_setClockPrescaler(epwm_base, EPWM_CLOCK_DIVIDER_1,
    EPWM_HSCLOCK_DIVIDER_1);
550
551     //
552     // Set counting direction UP after synchronization
553     //
554     EPWM_setCountModeAfterSync(epwm_base,
    EPWM_COUNT_MODE_UP_AFTER_SYNC);
555
556     //
557     // Set actions
558     //
559     EPWM_setActionQualifierAction(epwm_base, EPWM_AQ_OUTPUT_A,
    EPWM_AQ_OUTPUT_HIGH, EPWM_AQ_OUTPUT_ON_TIMEBASE_ZERO);
560     EPWM_setActionQualifierAction(epwm_base, EPWM_AQ_OUTPUT_A,
    EPWM_AQ_OUTPUT_LOW, EPWM_AQ_OUTPUT_ON_TIMEBASE_UP_CMPA);
561
562     //
563     // Enable ADC start of conversion triggering
564     //
565     EPWM_enableADCTrigger(epwm_base, EPWM_SOC_A);
566     EPWM_setADCTriggerSource(epwm_base, EPWM_SOC_A,
    EPWM_SOC_TBCTR_U_CMPC);
567     EPWM_setADCTriggerEventPrescale(epwm_base, EPWM_SOC_A, 1);
568     EPWM_clearADCTriggerFlag(epwm_base, EPWM_SOC_A);
569
```

```
570     //
571     // Set up shadowing
572     //
573     EPWM_selectPeriodLoadEvent(epwm_base,
        EPWM_SHADOW_LOAD_MODE_COUNTER_ZERO);
574     EPWM_setCounterCompareShadowLoadMode(epwm_base,
        EPWM_COUNTER_COMPARE_A, EPWM_COMP_LOAD_ON_SYNC_CNTR_ZERO);
575     EPWM_setCounterCompareShadowLoadMode(epwm_base,
        EPWM_COUNTER_COMPARE_C, EPWM_COMP_LOAD_ON_SYNC_CNTR_ZERO);
576     EPWM_enablePhaseShiftLoad(epwm_base);
577
578 }
579
580
581 //Configures the ePWMs to accept an external sync and to
582 // act in one shot mode
583 //Necessary for sensed control, not used for static
584 //included for completeness
585
586 void initEPWM_oneshot(uint32_t epwm_base,
        EPWM_DigitalCompareTripInput trigger_in,
        EPWM_DigitalCompareTripInput trip_in, uint16_t on_time)
587 {
588     //
589     // Clear the effects from fixed PWM settings
590     //
591     EPWM_setActionQualifierAction(epwm_base,
592                                   EPWM_AQ_OUTPUT_A,
593                                   EPWM_AQ_OUTPUT_NO_CHANGE,
594
        EPWM_AQ_OUTPUT_ON_TIMEBASE_UP_CMPA);
595
596     EPWM_setActionQualifierAction(epwm_base,
597                                   EPWM_AQ_OUTPUT_A,
598                                   EPWM_AQ_OUTPUT_NO_CHANGE,
599
```

```
    EPWM_AQ_OUTPUT_ON_TIMEBASE_ZERO);
600
601     //
602     // Seting up Period value to produce a pulse of 0.5us
603     //
604     EPWM_setTimeBasePeriod(epwm_base, 0xFFFF);
605
606     //
607     // Configuring the counter in up mode
608     //
609     EPWM_setTimeBaseCounterMode(epwm_base, EPWM_COUNTER_MODE_UP);
610
611     //
612     // Set ePWM clock pre-scaler
613     //
614     EPWM_setClockPrescaler(epwm_base,
615                            EPWM_CLOCK_DIVIDER_1,
616                            EPWM_HSCLOCK_DIVIDER_1);
617
618     //
619     // Configuring synchronization source as Digital Compare
620     // This sets the PWM to do a one shot output on DC
    synchronization
621     //
622
623     EPWM_selectDigitalCompareTripInput(epwm_base, trigger_in,
    EPWM_DC_TYPE_DCAL);
624     EPWM_setTripZoneDigitalCompareEventCondition(epwm_base,
    EPWM_TZ_DC_OUTPUT_A1, EPWM_TZ_EVENT_DCXL_HIGH);
625     EPWM_enableDigitalCompareEdgeFilter(epwm_base);
626     EPWM_setDigitalCompareEdgeFilterMode(epwm_base,
    EPWM_DC_EDGEFILT_MODE_RISING);
627     EPWM_setDigitalCompareEdgeFilterEdgeCount(epwm_base,
    EPWM_DC_EDGEFILT_EDGECNT_1);
628     EPWM_setDigitalCompareFilterInput(epwm_base,
    EPWM_DC_WINDOW_SOURCE_DCAEVT1);
```

```
629     EPWM_disableDigitalCompareBlankingWindow(epwm_base);
630     EPWM_setDigitalCompareEventSource(epwm_base, EPWM_DC_MODULE_A,
    EPWM_DC_EVENT_1, EPWM_DC_EVENT_SOURCE_FILT_SIGNAL);
631     EPWM_enableDigitalCompareSyncEvent(epwm_base, EPWM_DC_MODULE_A);
632     EPWM_setDigitalCompareEventSyncMode(epwm_base, EPWM_DC_MODULE_A,
        EPWM_DC_EVENT_1, EPWM_DC_EVENT_INPUT_SYNCED);
633
634     //
635     // Configuring trip source Digital Compare Event
636     // This will turn off the EPWM in that cycle for safety reasons
637     //
638
639     EPWM_setTripZoneAction(epwm_base, EPWM_TZ_ACTION_EVENT_DCAEVT2,
    EPWM_TZ_ACTION_LOW);
640     EPWM_enableTripZoneSignals(epwm_base, EPWM_TZ_SIGNAL_DCAEVT2);
641     EPWM_selectDigitalCompareTripInput(epwm_base, trip_in,
    EPWM_DC_TYPE_DCAH);
642     EPWM_setTripZoneDigitalCompareEventCondition(epwm_base,
    EPWM_TZ_DC_OUTPUT_A2, EPWM_TZ_EVENT_DCXH_HIGH);
643     EPWM_setDigitalCompareEventSource(epwm_base, EPWM_DC_MODULE_A,
    EPWM_DC_EVENT_2, EPWM_DC_EVENT_SOURCE_ORIG_SIGNAL);
644
645     //
646     // Setting phase offset as 0 after synchronization
647     //
648     EPWM_setPhaseShift(epwm_base, 0U);
649
650     //
651     // Set counting direction UP after synchronization
652     //
653     EPWM_setCountModeAfterSync(epwm_base,
    EPWM_COUNT_MODE_UP_AFTER_SYNC);
654
655     //
656     // Setting counter as 0
657     //
```

```
658     EPWM_setTimeBaseCounter(epwm_base, 0U);
659
660     //
661     // Set up shadowing
662     //
663     //EPWM_setCounterCompareShadowLoadMode(epwm_base,
    EPWM_COUNTER_COMPARE_A, EPWM_COMP_LOAD_ON_CNTR_ZERO);
664     EPWM_selectPeriodLoadEvent(epwm_base,
    EPWM_SHADOW_LOAD_MODE_COUNTER_ZERO);
665     //EPWM_setActionQualifierShadowLoadMode(epwm_base,
    EPWM_ACTION_QUALIFIER_A, EPWM_AQ_LOAD_ON_SYNC_CNTR_ZERO);
666     EPWM_setCounterCompareShadowLoadMode(epwm_base,
    EPWM_COUNTER_COMPARE_A, EPWM_COMP_LOAD_ON_SYNC_CNTR_ZERO);
667
668     //
669     // Set PWM output as LOW on CTR = PRD
670     //
671     EPWM_setActionQualifierAction(epwm_base, EPWM_AQ_OUTPUT_A,
    EPWM_AQ_OUTPUT_LOW, EPWM_AQ_OUTPUT_ON_TIMEBASE_PERIOD);
672
673     //
674     // Set PWM output as LOW on CTR = CMPA
675     //
676     EPWM_setActionQualifierAction(epwm_base, EPWM_AQ_OUTPUT_A,
    EPWM_AQ_OUTPUT_LOW, EPWM_AQ_OUTPUT_ON_TIMEBASE_UP_CMPA);
677
678     //
679     // Set PWM output as HIGH on T1 event
680     //
681     EPWM_setActionQualifierAction(epwm_base, EPWM_AQ_OUTPUT_A,
    EPWM_AQ_OUTPUT_HIGH, EPWM_AQ_OUTPUT_ON_T1_COUNT_UP);
682
683     //
684     // Set up counter compare with the on time
685     //
686     EPWM_setCounterCompareValue(epwm_base, EPWM_COUNTER_COMPARE_A,
```

```
    on_time);
687
688     //
689     // Configure T1 trigger source as PWM SYNC signal
690     //
691     //EPWM_setActionQualifierT1TriggerSource(epwm_base,
    EPWM_AQ_TRIGGER_EVENT_TRIG_EPWM_SYNCIN);
692     EPWM_setActionQualifierT1TriggerSource(epwm_base,
    EPWM_AQ_TRIGGER_EVENT_TRIG_DCA_1);
693
694     //
695     // Enabling phase load on synchronization
696     //
697     EPWM_enablePhaseShiftLoad(epwm_base);
698
699     //
700     // Enable ADC start of conversion triggering
701     //
702     EPWM_enableADCTrigger(epwm_base, EPWM_SOC_A);
703     EPWM_enableDigitalCompareADCTrigger(epwm_base, EPWM_DC_MODULE_A)
    ;
704     EPWM_setADCTriggerSource(epwm_base, EPWM_SOC_A, EPWM_SOC_DCxEVT1
    );
705     EPWM_setADCTriggerEventPrescale(epwm_base, EPWM_SOC_A, 1);
706     EPWM_clearADCTriggerFlag(epwm_base, EPWM_SOC_A);
707
708     //EPWM_enableInterrupt(epwm_base);
709     EPWM_enableTripZoneInterrupt(epwm_base,
    EPWM_TZ_INTERRUPT_DCAEVT1);
710     //EPWM_setInterruptEventCount(epwm_base, 1);
711
712 }
713
714 //Used to update the switch time parameters.
715 //See update_switches_reg and update_switches_nonreg
716 //  to get switch times in this format
```

```
717 void EPWM_set_timing(uint32_t epwm_base, uint16_t period, uint16_t
    duty, uint16_t phase, uint16_t adc_delay)
718 {
719     //
720     // Configuring time period of output signal as 10us
721     //
722     EPWM_setTimeBasePeriod(epwm_base, period);
723
724     //
725     // Set switch on time
726     //
727     EPWM_setCounterCompareValue(epwm_base, EPWM_COUNTER_COMPARE_A,
    duty);
728
729
730     //
731     // Setting phase offset after synchronization
732     //
733     EPWM_setPhaseShift(epwm_base, phase);
734
735     //
736     // Set up ADC measurement delay before turn on
737     //
738     EPWM_setCounterCompareValue(epwm_base, EPWM_COUNTER_COMPARE_C,
    adc_delay);
739 }
740
741
742 //Configures the regulating half bridge based on feedback loop
    parameters
743 void update_switches_reg(uint32_t primary_base, uint32_t
    secondary_base, uint16_t period, uint16_t primary_on, uint16_t
    primary_dt, uint16_t secondary_dt)
744 {
745     //
746     // Boundaries
```

```
747     //
748     //uint16_t dead_time_max = time_to_S3_on - time_to_S1_on;
749     uint16_t dead_time_max = DEAD_TIME_MAX;
750 
751     if (primary_on < DEAD_TIME_MIN)
752     {
753         primary_on = DEAD_TIME_MIN;
754         rpon_integral = ((long)primary_on)<<16;
755     }
756     else if(primary_on > ((period>>1) - DEAD_TIME_MIN))
757     {
758         primary_on = (period>>1) - DEAD_TIME_MIN;
759         rpon_integral = ((long)primary_on)<<16;
760     }
761 
762     if(primary_dt < DEAD_TIME_MIN)
763     {
764         primary_dt = DEAD_TIME_MIN;
765         rpdt_integral = ((long)primary_dt)<<16;
766     }
767     else if(primary_dt > dead_time_max)
768     {
769         primary_dt = dead_time_max;
770         rpdt_integral = ((long)primary_dt)<<16;
771     }
772 
773     if(secondary_dt < DEAD_TIME_MIN)
774     {
775         secondary_dt = DEAD_TIME_MIN;
776         rsdt_integral = ((long)secondary_dt)<<16;
777     }
778     else if(secondary_dt > dead_time_max)
779     {
780         secondary_dt = dead_time_max;
781         rsdt_integral = ((long)secondary_dt)<<16;
782     }
```

```
783
784     //
785     // Standard/Direct Control
786     //
787     //EPWM_set_timing(s1_base, period, s1on, 0, period-
    ADC_MEASUREMENT_DELAY);
788     //EPWM_set_timing(s2_base, period, s2on, period-phase, period-
    ADC_MEASUREMENT_DELAY);
789
790     //
791     // S1-sensitive control
792     // Holds S2on-S1off and S2off 'constant' for a given S1on step
793     //
794
795     if(FORWARD)
796     {
797         EPWM_set_timing(primary_base, period, primary_on, 0, period
    - ADC_MEASUREMENT_DELAY);
798         EPWM_set_timing(secondary_base, period, period-primary_on-
    primary_dt-secondary_dt, period-primary_on-secondary_dt, period-
    ADC_MEASUREMENT_DELAY);
799     }
800     else
801     {
802         EPWM_set_timing(primary_base, period, primary_on, primary_on
    , period - ADC_MEASUREMENT_DELAY);
803         EPWM_set_timing(secondary_base, period, period-primary_on-
    primary_dt-secondary_dt, period-secondary_dt, period-
    ADC_MEASUREMENT_DELAY);
804     }
805 }
806
807
808
809 //Configures the nonregulating half bridge based on feedback loop
    parameters
```

```
810 void update_switches_nonreg(uint32_t primary_base, uint32_t
    secondary_base, uint16_t period, uint16_t primary_dt, int16_t
    duty_correction)
811 {
812     //
813     // Half Period Symmetric control
814     // Assumes S3on=S4on and t4=t6b
815     // Ties S4 to turn off when S1 turns on
816     //
817     //EPWM_set_timing(s1_base, period, s3on, period>>2 + s3on,
    period - ADC_MEASUREMENT_DELAY);
818
819
820     if(primary_dt < DEAD_TIME_MIN)
821     {
822         primary_dt = DEAD_TIME_MIN;
823         npdt_integral = ((long)primary_dt)<<16;
824     }
825     else if(primary_dt > DEAD_TIME_MAX)
826     {
827         primary_dt = DEAD_TIME_MAX;
828         npdt_integral = ((long)primary_dt)<<16;
829     }
830
831
832     if(FORWARD)
833     {
834         EPWM_set_timing(primary_base, period, (period>>1)+
    duty_correction-primary_dt, (period>>1)+duty_correction-
    primary_dt, period - ADC_MEASUREMENT_DELAY);
835         EPWM_set_timing(secondary_base, period, (period>>1)-
    duty_correction-primary_dt, period-primary_dt, period-
    ADC_MEASUREMENT_DELAY);
836     }
837     else
838     {
```

```
839         EPWM_set_timing(primary_base, period, (period>>1)+
    duty_correction-primary_dt, 0, period - ADC_MEASUREMENT_DELAY);
840         EPWM_set_timing(secondary_base, period, (period>>1)-
    duty_correction-primary_dt, (period>>1)-duty_correction, period-
    ADC_MEASUREMENT_DELAY);
841     }
842 }
843
844
845 //
846 // configureADC - Write ADC configurations and power up the ADC for
    both
847 //                 ADC A and ADC B
848 //
849 //Configures ADCs in single ended mode and enables them
850 void configureADC(uint32_t adcBase)
851 {
852     //
853     // Set ADCDLK divider to /4
854     //
855     ADC_setPrescaler(adcBase, ADC_CLK_DIV_4_0);
856
857     //
858     // Set resolution and signal mode (see #defines above) and load
859     // corresponding trims.
860     //
861     ADC_setMode(adcBase, ADC_RESOLUTION_12BIT, ADC_MODE_SINGLE_ENDED
    );
862
863
864     //
865     // Set pulse positions to late
866     //
867     ADC_setInterruptPulseMode(adcBase, ADC_PULSE_END_OF_CONV);
868
869     //
```

```
870     // Power up the ADCs and then delay for 1 ms
871     //
872     ADC_enableConverter(adcBase);
873
874     //
875     // Delay for 1ms to allow ADC time to power up
876     //
877     DEVICE_DELAY_US(1000);
878 }
879
880
881 //Configures the ADCs to trigger a start of conversion (SOC)
882 //  on a given trigger signal
883 void setupADCTriggered(uint32_t adcBase, uint32_t channel,
        ADC_SOCNumber soc, ADC_Trigger trigger, ADC_IntNumber adc_int)
884 {
885     uint16_t acqps = 10;//30;
886
887     ADC_setupSOC(adcBase, soc, trigger, (ADC_Channel)channel, acqps)
        ;
888     ADC_setInterruptSource(adcBase, adc_int, soc);
889     ADC_enableInterrupt(adcBase, adc_int);
890     ADC_clearInterruptStatus(adcBase, adc_int);
891 }
892
893
894 //Runs the main feedback code
895 //Is an interrupt service routine because it is called
896 //  by hardware at certain points in the switching sequence.
897 __interrupt void feedback_control_ISR()
898 {
899
900     //Check if we are at the appropriate count to run
901     if(interrupt_count < COUNT_MAX)
902     {
903         interrupt_count++;
```

```
904     }
905     else
906     {
907         interrupt_count = 0;
908 
909         //Get current measurement values from hardware
910         //MODIFY ONLY IF NECESSARY the adc result registers to get
    proper measurements
911         current_vpr_before_rp = (uint32_t)ADC_readResult(
    ADCARESULT_BASE, ADC_SOC_NUMBER0);
912         current_vpr_before_rs = (uint32_t)ADC_readResult(
    ADCARESULT_BASE, ADC_SOC_NUMBER1);
913         current_vpn_before_np = (uint32_t)ADC_readResult(
    ADCBRESULT_BASE, ADC_SOC_NUMBER0);
914         current_vout = (uint32_t)ADC_readResult(ADCDRESULT_BASE,
    ADC_SOC_NUMBER0);
915 
916         /* FIRST ORDER DISCRETE FILTER - uncomment to enable
917         current_vpr_before_rp_raw = ADC_readResult(ADCBRESULT_BASE,
    ADC_SOC_NUMBER0);
918         current_vpr_before_rs_raw = ADC_readResult(ADCBRESULT_BASE,
    ADC_SOC_NUMBER1);
919         current_vpn_before_np_raw = ADC_readResult(ADCARESULT_BASE,
    ADC_SOC_NUMBER0);
920         current_vout_raw = ADC_readResult(ADCDRESULT_BASE,
    ADC_SOC_NUMBER0);
921 
922         current_vp1_before_s1 = (current_vp1_before_s1)*alpha +
    current_vp1_before_s1_raw*(one_minus_alpha);
923         current_vp1_before_s2 = (current_vp1_before_s2)*alpha +
    current_vp1_before_s2_raw*(one_minus_alpha);
924         current_vp2_before_s3 = (current_vp2_before_s3)*alpha +
    current_vp2_before_s3_raw*(one_minus_alpha);
925         current_vout = current_vout*alpha + current_vout_raw*(
    one_minus_alpha);
926 
```

```
927         current_vp1_before_s1 = current_vp1_before_s1 >> 16;
928         current_vp1_before_s2 = current_vp1_before_s2 >> 16;
929         current_vp2_before_s3 = current_vp2_before_s3 >> 16;
930         current_vout = current_vout >> 16;
931         */
932
933         //Get Zero crossing detector measurements
934         uint32_t t_beta_raw = CLB_getRegister(CLB1_BASE,
    CLB_REG_HLC_R0);
935         uint32_t t_alpha_raw = CLB_getRegister(CLB1_BASE,
    CLB_REG_HLC_R1);
936         bool zc_valid = false;
937
938         //Ignore measurements if one or both times are 0
939         if (t_alpha_raw > 0 && t_beta_raw > 0)// && t_beta_raw < 2*(
    current_npdt + current_rsdt))
940         {
941             t_alpha = t_alpha_raw;
942             t_beta = t_beta_raw;
943
944             zc_valid = true;
945         }
946
947         //Update comparator DAC for comparison value of Vp2
948         //MODIFY if the comparator should instead listen for a
    different value
949         //  Than Vin-Vout. +Zero is added because that term will
    disappear
950         //  after the difference
951         // the -250 is an empirically determined offset to make the
    DAC
952         //  output the proper values. MODIFY IF NECESSARY to make
    the
953         //comparator edges occur when they are supposed to on the
    scope
954         CMPSS_setDACValueHigh(CMPSS1_BASE, VIN-current_vout+ZERO
```

```
-250);
955
956
957         //Update comparator DAC for comparison value of Vp1
958         //MODIFY if the comparator should instead listen for a
    different value
959         //  Than Vout. +Zero is needed with a difference because
    that term will disappear
960         //  after the difference
961         // the -250 is an empirically determined offset to make the
    DAC
962         //  output the proper values. MODIFY IF NECESSARY to make
    the
963         //comparator edges occur when they are supposed to on the
    scope
964         if(current_vout > 500)
965         {
966             CMPSS_setDACValueHigh(CMPSS3_BASE, current_vout-250);
967         }
968
969
970         //Only update switching times if the feedback control is
    enabled
971         if(feedback_control_enabled)
972         {
973             //GPIO_writePin(myGPIO0, 1);
974             //GPIO_writePin(myGPIO0, 0);
975
976             //error computations
977             /*
978             error_vout = current_vout - DESIRED_VOUT;
979             error_vpr_before_rp = (current_vpr_before_rp - (ZERO +
    VPR_BEFORE_RP_CORRECTION));
980             error_vpr_before_rs = -(current_vpr_before_rs - (
    current_vout + VPR_BEFORE_RS_CORRECTION));
981             error_vpn_before_np = -(current_vpn_before_np - (VIN +
```

```
VPN_BEFORE_NP_CORRECTION));
982             error_zero_crossing_offset = t_alpha - (t_beta/2 +
ZC_CORRECTION);
983             */
984
985             //Compute feedback loop errors from measurements
986             //MODIFY to match the switching sequence
987             error_vout = current_vout - DESIRED_VOUT;
988             error_vpr_before_rp = (current_vpr_before_rp - (ZERO +
VPR_BEFORE_RP_CORRECTION));
989             error_vpr_before_rs = -(current_vpr_before_rs - (
current_vout + VPR_BEFORE_RS_CORRECTION));
990             error_vpn_before_np = -(current_vpn_before_np - (VIN +
VPN_BEFORE_NP_CORRECTION));
991             error_zero_crossing_offset = t_alpha - (t_beta/2 +
ZC_CORRECTION);
992
993
994
995             //
996             // RHB Primary on time feedback
997             // Based on Vout error
998             //
999             current_rpon = (uint16_t)((error_vout*K_P_VOUT +
rpon_integral)>>16);
1000            rpon_integral += error_vout*K_INT_VOUT;
1001
1002            //
1003            // RHB Primary dead time feedback
1004            // Based on Vpr error just before it turns on
1005            //
1006            current_rpdt = (uint16_t)((error_vpr_before_rp*K_P_ZVS +
 rpdt_integral)>>16);
1007            rpdt_integral += error_vpr_before_rp*K_INT_ZVS;
1008
1009            //
```

```
1010            // RHB Secondary dead time feedback
1011            // Based on Vpr error just before it turns on
1012            //
1013            current_rsdt = (uint16_t)((error_vpr_before_rs*K_P_ZVS +
     rsdt_integral)>>16);
1014            rsdt_integral += error_vpr_before_rs*K_INT_ZVS;
1015
1016            //
1017            // Period feedback
1018            // Based on zero crossing offset time
1019            //
1020
1021            //Attempted period control "save" if the output voltage
    drops too low
1022            //Converter will get stuck outside the desired PR
    frequency range for
1023            //  the load resistance. This attempts to lower the
    frequency to fix that
1024            //Used with the Vout > 1/2 Vin mode.
1025            //MODIFY (or remove) depending on switching sequence.
    For example, this
1026            // should not be enabled for Vout < 1/2 Vin, because
    that is the desired range.
1027            if (current_vout < (ZERO+VIN)/2)
1028            {
1029                period_integral += 1<<12;
1030                current_period = (uint16_t)((
    error_zero_crossing_offset*K_P_ZC + period_integral)>>16);
1031            }
1032            else if(zc_valid)
1033            {
1034                current_period = (uint16_t)((
    error_zero_crossing_offset*K_P_ZC + period_integral)>>16);
1035                period_integral += error_zero_crossing_offset*
    K_INT_ZC;
1036            }
```

```
1037
1038
1039             //
1040             // NRHB Primary dead time feedback
1041             // Based on Vpn error just before it turns on
1042             //
1043             current_npdt = (uint16_t)((error_vpn_before_np*K_P_ZVS +
     npdt_integral)>>16);
1044             npdt_integral += error_vpn_before_np*K_INT_ZVS;
1045
1046             //
1047             // Update switching times for the next cycle
1048             //
1049             update_switches_reg(RP, RS, current_period, current_rpon
    , current_rpdt, current_rsdt);
1050             update_switches_nonreg(NP, NS, current_period,
    current_npdt, DUTY_CORRECTION);
1051
1052             //GPIO_writePin(myGPIO0, 1);
1053             //GPIO_writePin(myGPIO0, 0);
1054         }
1055
1056     }
1057
1058
1059     //Clear the interrupt status so the interrupt can run again
1060     //ENSURE that this matches the interrupt that triggers the
1061     // Feedback loop code.
1062
1063
1064     //ADC_clearInterruptStatus(ADCB_BASE, ADC_INT_NUMBER4);
1065     //ADC_clearInterruptStatus(ADCD_BASE, ADC_INT_NUMBER3);
1066     ADC_clearInterruptStatus(ADCA_BASE, ADC_INT_NUMBER4);
1067     Interrupt_clearACKGroup(INTERRUPT_ACK_GROUP10);
1068
1069 }
```

```
1070
1071
1072 //
1073 // initCMPSS - Function to configure the high comparator of CMPSS1
1074 //
1075 void initCMPSS(uint32_t cmpssBase, uint16_t threshold, bool invert)
1076 {
1077     //
1078     // Enable CMPSS and configure the negative input signal to come
     from
1079     // the DAC
1080     //
1081     CMPSS_enableModule(cmpssBase);
1082     if(invert)
1083     {
1084         CMPSS_configHighComparator(cmpssBase, CMPSS_INSRC_DAC |
     CMPSS_INV_INVERTED);
1085     }
1086     else
1087     {
1088         CMPSS_configHighComparator(cmpssBase, CMPSS_INSRC_DAC);
1089     }
1090
1091     //
1092     // Use VDDA as the reference for the DAC and set DAC value to
     midpoint for
1093     // arbitrary reference.
1094     //
1095     CMPSS_configDAC(cmpssBase, CMPSS_DACREF_VDDA |
     CMPSS_DACVAL_SYSCLK | CMPSS_DACSRC_SHDW);
1096     CMPSS_setDACValueHigh(cmpssBase, threshold);
1097
1098     //
1099     // Set up hysteresis
1100     //
1101     CMPSS_setHysteresis(cmpssBase, 1);
```

```
1102
1103     CMPSS_configFilterHigh(cmpssBase, 0, 3, 2);
1104
1105     CMPSS_initFilterHigh(cmpssBase);
1106
1107     //
1108     // Configure the output signals. Both CTRIPH and CTRIPOUTH will
    be fed by
1109     // the asynchronous comparator output.
1110     //
1111     CMPSS_configOutputsHigh(cmpssBase, CMPSS_TRIP_FILTER |
    CMPSS_TRIPOUT_FILTER);
1112 }
1113
1114 void initCLB_ZERO_CROSSING_TIMER(uint32_t clb_base)
1115 {
1116     //
1117     // Enable
1118     //
1119     CLB_enableCLB(clb_base);
1120
1121
1122     //
1123     // Select Global input instead of local input for all CLB IN
1124     //
1125     CLB_configLocalInputMux(clb_base, CLB_IN0,
    CLB_LOCAL_IN_MUX_GLOBAL_IN);
1126     CLB_configLocalInputMux(clb_base, CLB_IN1,
    CLB_LOCAL_IN_MUX_GLOBAL_IN);
1127     CLB_configLocalInputMux(clb_base, CLB_IN2,
    CLB_LOCAL_IN_MUX_GLOBAL_IN);
1128     CLB_configLocalInputMux(clb_base, CLB_IN3,
    CLB_LOCAL_IN_MUX_GLOBAL_IN);
1129     CLB_configLocalInputMux(clb_base, CLB_IN4,
    CLB_LOCAL_IN_MUX_GLOBAL_IN);
1130
```

```
1131
1132
1133     //
1134     // Configure inputs for switch turn offs and comparator signals
1135     //
1136     //MODIFY so that the start, alpha, and beta pulses are the
        correct signals
1137     //  for the ZCD for the configured switching sequence
1138     CLB_configGlobalInputMux(clb_base, CLB_IN0,
        CLB_GLOBAL_IN_MUX_EPWM2_CTR_CMPA); //t0
1139     CLB_configGlobalInputMux(clb_base, CLB_IN1,
        CLB_GLOBAL_IN_MUX_CLB_AUXSIG1);    //t2
1140     CLB_configGlobalInputMux(clb_base, CLB_IN2,
        CLB_GLOBAL_IN_MUX_EPWM1_CTR_ZERO); //t1
1141     CLB_configGlobalInputMux(clb_base, CLB_IN3,
        CLB_GLOBAL_IN_MUX_EPWM4_CTR_ZERO); //Reset counters
1142     CLB_configGlobalInputMux(clb_base, CLB_IN4,
        CLB_GLOBAL_IN_MUX_EPWM3_CTR_ZERO); //Latch output (Out1 = t1-t0,
        out2 = t2-t0)
1143
1144
1145
1146     //
1147     // Configure inputs 1-4 as external, the rest are unused so tie
        to the general purpose registers
1148     //
1149     CLB_configGPInputMux(clb_base, CLB_IN0, CLB_GP_IN_MUX_EXTERNAL);
1150     CLB_configGPInputMux(clb_base, CLB_IN1, CLB_GP_IN_MUX_EXTERNAL);
1151     CLB_configGPInputMux(clb_base, CLB_IN2, CLB_GP_IN_MUX_EXTERNAL);
1152     CLB_configGPInputMux(clb_base, CLB_IN3, CLB_GP_IN_MUX_EXTERNAL);
1153     CLB_configGPInputMux(clb_base, CLB_IN4, CLB_GP_IN_MUX_EXTERNAL);
1154     CLB_configGPInputMux(clb_base, CLB_IN5, CLB_GP_IN_MUX_GP_REG);
1155     CLB_configGPInputMux(clb_base, CLB_IN6, CLB_GP_IN_MUX_GP_REG);
1156     CLB_configGPInputMux(clb_base, CLB_IN7, CLB_GP_IN_MUX_GP_REG);
1157
1158
```

```
1159
1160     //
1161     // Configure CLB-XBAR AUXSIG0 as CMPSS3.CTRIPH (Vp1 rises above
         Vin-Vout)
1162     //
1163     XBAR_setCLBMuxConfig(XBAR_AUXSIG0, XBAR_CLB_MUX04_CMPSS3_CTRIPH)
         ;
1164     XBAR_enableCLBMux(XBAR_AUXSIG0, XBAR_MUX04);
1165
1166     //
1167     // Configure CLB-XBAR AUXSIG1 as CMPSS1.CTRIPH (Vp2 rises above
         Vout)
1168     //
1169     XBAR_setCLBMuxConfig(XBAR_AUXSIG1, XBAR_CLB_MUX00_CMPSS1_CTRIPH)
         ;
1170     XBAR_enableCLBMux(XBAR_AUXSIG1, XBAR_MUX00);
1171
1172
1173     //
1174     // Load generated logic configuration
1175     //
1176     initTILE_ZERO_CROSSING(clb_base);
1177
1178 }
1179 //
1180 // End of file
1181 //
```

H.2 System Configuration File

This is a ".sysconfig" file is used by the TI SysCfg tool in the CCS IDE to autogenerate code that configures many of the hardware components, including some aspects of the ePWMs and the CLB. It is necessary for the code in the previous section to run properly.

```
1 /**
```

```
 * These arguments were used when this file was generated. They will
    be automatically applied on subsequent loads
 * via the GUI or CLI. Run CLI with '--help' for additional
   information on how to override these arguments.
 * @cliArgs --device "F2837xD" --package "F2837xD_176PTP" --part "
   F2837xD_176PTP" --product "C2000WARE@3.00.00.00"
 * @versions {"data":"2021010520","timestamp":"2021010520","tool
   ":"1.7.0+1746","templates":null}
 */

/**
 * Import the modules used in this configuration.
 */
const epwm        = scripting.addModule("/driverlib/epwm.js", {},
   false);
const epwm1       = epwm.addInstance();
const epwm2       = epwm.addInstance();
const epwm3       = epwm.addInstance();
const epwm4       = epwm.addInstance();
const epwm5       = epwm.addInstance();
const epwm6       = epwm.addInstance();
const epwmxbar    = scripting.addModule("/driverlib/epwmxbar.js",
   {}, false);
const epwmxbar1   = epwmxbar.addInstance();
const epwmxbar2   = epwmxbar.addInstance();
const gpio        = scripting.addModule("/driverlib/gpio.js", {},
   false);
const gpio1       = gpio.addInstance();
const inputxbar   = scripting.addModule("/driverlib/inputxbar.js",
   {}, false);
const inputxbar1  = inputxbar.addInstance();
const outputxbar  = scripting.addModule("/driverlib/outputxbar.js",
   {}, false);
const outputxbar1 = outputxbar.addInstance();
const outputxbar2 = outputxbar.addInstance();
const outputxbar3 = outputxbar.addInstance();
```

```
29 const TILE        = scripting.addModule("/utilities/clb_tool/
      clb_syscfg/source/TILE", {}, false);
30 const TILE1       = TILE.addInstance();
31 const TILE2       = TILE.addInstance();
32
33 /**
34  * Write custom configuration values to the imported modules.
35  */
36 epwm1.useCase                = "CUSTOM";
37 epwm1.useInterfacePins       = ["EPWM#A"];
38 epwm1.$name                  = "myEPWM2";
39 epwm1.epwm.$assign           = "EPWM2";
40 epwm1.epwm.epwmaPin.$assign = "162";
41
42 epwm2.$name                  = "myEPWM1";
43 epwm2.useCase                = "CUSTOM";
44 epwm2.useInterfacePins       = ["EPWM#A"];
45 epwm2.epwm.$assign           = "EPWM1";
46 epwm2.epwm.epwmaPin.$assign = "160";
47
48 epwm3.$name                  = "myEPWM7";
49 epwm3.useCase                = "CUSTOM";
50 epwm3.useInterfacePins       = ["EPWM#A"];
51 epwm3.epwm.$assign           = "EPWM7";
52 epwm3.epwm.epwmaPin.$assign = "4";
53
54 epwm4.$name                  = "myEPWM8";
55 epwm4.useCase                = "CUSTOM";
56 epwm4.useInterfacePins       = ["EPWM#A"];
57 epwm4.epwm.$assign           = "EPWM8";
58 epwm4.epwm.epwmaPin.$assign = "6";
59
60 epwm5.$name                  = "myEPWM3";
61 epwm5.epwm.$assign           = "EPWM3";
62 epwm5.epwm.epwmaPin.$assign = "164";
63 epwm5.epwm.epwmbPin.$assign = "165";
```

```
64
65 epwm6.$name                  = "myEPWM4";
66 epwm6.epwm.$assign           = "EPWM4";
67 epwm6.epwm.epwmaPin.$assign = "166";
68 epwm6.epwm.epwmbPin.$assign = "167";
69
70 epwmxbar1.mux1Config = "XBAR_EPWM_MUX01_INPUTXBAR1";
71 epwmxbar1.$name      = "myEPWMXBAR0_highside_trigger";
72 epwmxbar1.muxesUsed  = ["XBAR_MUX05"];
73 epwmxbar1.mux5Config = "XBAR_EPWM_MUX05_INPUTXBAR3";
74
75 epwmxbar2.tripInput  = "XBAR_TRIP5";
76 epwmxbar2.mux8Config = "XBAR_EPWM_MUX08_ADCBEVT1";
77 epwmxbar2.mux3Config = "XBAR_EPWM_MUX03_INPUTXBAR2";
78 epwmxbar2.$name      = "myEPWMXBAR1_lowside_trigger";
79 epwmxbar2.muxesUsed  = ["XBAR_MUX07"];
80 epwmxbar2.mux7Config = "XBAR_EPWM_MUX07_INPUTXBAR4";
81
82 gpio1.$name           = "myGPIO0";
83 gpio1.direction       = "GPIO_DIR_MODE_OUT";
84 gpio1.gpioPin.$assign = "27";
85
86 inputxbar1.$name          = "myINPUTXBAR0";
87 inputxbar1.inputsUsed     = ["inputxbar1Gpio","inputxbar2Gpio","
      inputxbar3Gpio","inputxbar4Gpio"];
88 inputxbar1.inputxbar1Lock = true;
89 inputxbar1.inputxbar2Lock = true;
90 inputxbar1.inputxbar3Lock = true;
91 inputxbar1.inputxbar4Lock = true;
92 inputxbar1.inputxbar2Gpio = "GPIO6";
93 inputxbar1.inputxbar3Gpio = "GPIO12";
94 inputxbar1.inputxbar4Gpio = "GPIO14";
95
96 outputxbar1.$name                              = "myOUTPUTXBAR0";
97 outputxbar1.muxesUsed                          = ["XBAR_MUX00"];
98 outputxbar1.outputxbar.$assign                 = "OUTPUTXBAR1";
```

```
outputxbar1.outputxbar.outputxbarPin.$assign = "24";

outputxbar2.$name                            = "myOUTPUTXBAR1";
outputxbar2.mux8Config                       = "
   XBAR_OUT_MUX08_ADCBEVT1";
outputxbar2.muxesUsed                        = ["XBAR_MUX04"];
outputxbar2.outputxbar.$assign               = "OUTPUTXBAR2";
outputxbar2.outputxbar.outputxbarPin.$assign = "25";

outputxbar3.$name = "myOUTPUTXBAR2";

TILE1.$name               = "TILE_SW_TRIGGER";
TILE1.BOUNDARY.$name      = "BOUNDARY0";
TILE1.LUT_0.$name         = "LUT_0";
TILE1.LUT_1.$name         = "LUT_1";
TILE1.LUT_2.$name         = "LUT_2";
TILE1.FSM_0.$name         = "FSM_0";
TILE1.FSM_0.e0            = "BOUNDARY.in1";
TILE1.FSM_0.e1            = "BOUNDARY.in2";
TILE1.FSM_0.eqn_out       = "0";
TILE1.FSM_0.eqn_s0        = "(!e1&!e0&s0) | (e1 & !e0)";
TILE1.FSM_0.eqn_s1        = "e1&!e0&!s1&!s0";
TILE1.FSM_1.$name         = "FSM_1";
TILE1.FSM_1.eqn_s0        = "(!e1&!e0&s0) | (e1 & !e0)";
TILE1.FSM_1.eqn_out       = "0";
TILE1.FSM_1.e0            = "BOUNDARY.in0";
TILE1.FSM_1.e1            = "BOUNDARY.in3";
TILE1.FSM_1.eqn_s1        = "e1&!e0&!s1&!s0";
TILE1.FSM_2.$name         = "FSM_2";
TILE1.COUNTER_0.$name     = "COUNTER_0";
TILE1.COUNTER_1.$name     = "COUNTER_1";
TILE1.COUNTER_2.$name     = "COUNTER_2";
TILE1.OUTLUT_0.$name      = "OUTLUT_0";
TILE1.OUTLUT_1.$name      = "OUTLUT_1";
TILE1.OUTLUT_2.$name      = "OUTLUT_2";
TILE1.OUTLUT_3.$name      = "OUTLUT_3";
```

```
TILE1.OUTLUT_4.$name      = "OUTLUT_4";
TILE1.OUTLUT_4.i0         = "FSM_0.S1";
TILE1.OUTLUT_4.i1         = "BOUNDARY.in4";
TILE1.OUTLUT_4.eqn        = "i0&i1";
TILE1.OUTLUT_5.$name      = "OUTLUT_5";
TILE1.OUTLUT_5.i0         = "FSM_1.S1";
TILE1.OUTLUT_5.i1         = "BOUNDARY.in4";
TILE1.OUTLUT_5.eqn        = "i0&i1";
TILE1.OUTLUT_6.$name      = "OUTLUT_6";
TILE1.OUTLUT_7.$name      = "OUTLUT_7";
TILE1.HLC.$name           = "HLC_0";
TILE1.HLC.program0.$name  = "HLCP_0";
TILE1.HLC.program1.$name  = "HLCP_1";
TILE1.HLC.program2.$name  = "HLCP_2";
TILE1.HLC.program3.$name  = "HLCP_3";

TILE2.$name                   = "TILE_ZERO_CROSSING";
TILE2.BOUNDARY.$name          = "BOUNDARY1";
TILE2.LUT_0.$name             = "LUT_3";
TILE2.LUT_0.eqn               = "i0|i1";
TILE2.LUT_0.i0                = "BOUNDARY.in1";
TILE2.LUT_0.i1                = "BOUNDARY.in3";
TILE2.LUT_1.$name             = "LUT_4";
TILE2.LUT_1.eqn               = "i0|i1";
TILE2.LUT_1.i0                = "BOUNDARY.in2";
TILE2.LUT_1.i1                = "BOUNDARY.in3";
TILE2.LUT_2.$name             = "LUT_5";
TILE2.FSM_0.$name             = "FSM_3";
TILE2.FSM_0.e0                = "BOUNDARY.in0";
TILE2.FSM_0.eqn_s1            = "0";
TILE2.FSM_0.eqn_out           = "e0&~s0";
TILE2.FSM_0.eqn_s0            = "e0";
TILE2.FSM_1.$name             = "FSM_4";
TILE2.FSM_1.eqn_out           = "0";
TILE2.FSM_1.eqn_s0            = "(s0|e0)&(~e1)";
TILE2.FSM_1.eqn_s1            = "0";
```

```
TILE2.FSM_1.e1                = "LUT_0.OUT";
TILE2.FSM_1.e0                = "FSM_0.OUT";
TILE2.FSM_2.$name             = "FSM_5";
TILE2.FSM_2.eqn_out           = "0";
TILE2.FSM_2.eqn_s0            = "(s0|e0)&(~e1)";
TILE2.FSM_2.eqn_s1            = "0";
TILE2.FSM_2.e0                = "FSM_0.OUT";
TILE2.FSM_2.e1                = "LUT_1.OUT";
TILE2.COUNTER_0.$name         = "COUNTER_3";
TILE2.COUNTER_0.mode1         = "1";
TILE2.COUNTER_0.reset         = "BOUNDARY.in3";
TILE2.COUNTER_0.mode0         = "FSM_1.S0";
TILE2.COUNTER_1.$name         = "COUNTER_4";
TILE2.COUNTER_1.reset         = "BOUNDARY.in3";
TILE2.COUNTER_1.mode1         = "1";
TILE2.COUNTER_1.mode0         = "FSM_2.S0";
TILE2.COUNTER_2.$name         = "COUNTER_5";
TILE2.OUTLUT_0.$name          = "OUTLUT_8";
TILE2.OUTLUT_1.$name          = "OUTLUT_9";
TILE2.OUTLUT_2.$name          = "OUTLUT_10";
TILE2.OUTLUT_3.$name          = "OUTLUT_11";
TILE2.OUTLUT_4.$name          = "OUTLUT_12";
TILE2.OUTLUT_5.$name          = "OUTLUT_13";
TILE2.OUTLUT_6.$name          = "OUTLUT_14";
TILE2.OUTLUT_7.$name          = "OUTLUT_15";
TILE2.HLC.$name               = "HLC_1";
TILE2.HLC.e0                  = "BOUNDARY.in4";
TILE2.HLC.R0_init             = "0xDEADCODE";
TILE2.HLC.R1_init             = "0xFEEDBEEF";
TILE2.HLC.program0.$name      = "HLCP_4";
TILE2.HLC.program0.instruct1  = "MOV C1,R1";
TILE2.HLC.program0.instruct0  = "MOV C0,R0";
TILE2.HLC.program1.$name      = "HLCP_5";
TILE2.HLC.program2.$name      = "HLCP_6";
TILE2.HLC.program3.$name      = "HLCP_7";

```

```
206 /**
207  * Pinmux solution for unlocked pins/peripherals. This ensures that
        minor changes to the automatic solver in a future
208  * version of the tool will not impact the pinmux you originally saw
        .  These lines can be completely deleted in order to
209  * re-solve from scratch.
210  */
211 outputxbar3.outputxbar.$suggestSolution                = "OUTPUTXBAR3
        ";
212 outputxbar3.outputxbar.outputxbarPin.$suggestSolution = "90";
```

Bibliography

[1] J. M. Alonso, C. Ordiz, and M. A. Dalla Costa. A novel control method for piezoelectric-transformer based power supplies assuring zero-voltage-switching operation. *IEEE Transactions on Industrial Electronics*, 55(3):1085–1089, 2008.

[2] Jessica D Boles, Joseph E Bonavia, Pedro L Acosta, Y K Ramadass, J H Lang, and David J Perreault. Evaluating piezoelectric materials and vibration modes for power conversion. *IEEE Transactions on Power Electronics*, 2022.

[3] Jessica D Boles, Elaine Ng, Jeffrey H Lang, and David J Perreault. Dc-dc converter implementations based on piezoelectric transformers. *Journal of Emerging and Selected Topics in Power Electronics*, 2022.

[4] Jessica D Boles, Joshua J Piel, and David J Perreault. Enumeration and analysis of dc-dc converter implementations based on piezoelectric resonators. *IEEE Transactions on Power Electronics*, 36(1):129–145, 2021.

[5] Joseph E. Bonavia, Jessica D. Boles, Jeffrey H. Lang, and David J. Perreault. Augmented piezoelectric resonators for power conversion. In *2021 IEEE 22nd Workshop on Control and Modelling of Power Electronics (COMPEL)*, pages 1–8, 2021.

[6] Weston D. Braun, Eric A. Stolt, Lei Gu, Jeronimo Segovia-Fernandez, Sombuddha Chakraborty, Ruochen Lu, and Juan M. Rivas-Davila. Optimized resonators for piezoelectric power conversion. *IEEE Open Journal of Power Electronics*, 2:212–224, 2021.

[7] S. Chen and C. Chen. ZVS considerations for a phase-lock control dc/dc converter with piezoelectric transformer. In *IECON 2006 - 32nd Annual Conference on IEEE Industrial Electronics*, pages 2244–2248, 2006.

[8] M. Ekhtiari, Z. Zhang, and M. A. E. Andersen. Analysis of bidirectional piezoelectric-based converters for zero-voltage switching operation. *IEEE Transactions on Power Electronics*, 32(1):866–877, Jan 2017.

[9] David M Giuliano, Matthew E D'Asaro, Jacob Zwart, and David J Perreault. Miniaturized low-voltage power converters with fast dynamic response. *IEEE Journal of Emerging and Selected Topics in Power Electronics*, 2(3):395–405, 2014.

[10] Alex J. Hanson and David J. Perreault. A high-frequency power factor correction stage with low output voltage. *IEEE Journal of Emerging and Selected Topics in Power Electronics*, 8(3):2143–2155, 2020.

[11] E. L. Horsley, A. V. Carazo, N. Nguyen-Quang, M. P. Foster, and D. A. Stone. Analysis of inductorless zero-voltage-switching piezoelectric transformer-based converters. *IEEE Transactions on Power Electronics*, 27(5):2471–2483, 2012.

[12] Phyo Aung Kyaw, Aaron LF Stein, and Charles R Sullivan. Fundamental examination of multiple potential passive component technologies for future power electronics. *IEEE Transactions on Power Electronics*, 33(12):10,708–10,722, 2018.

[13] Yutian Lei and Robert Carl Nikolai Pilawa-Podgurski. A general method for analyzing resonant and soft-charging operation of switched-capacitor converters. *IEEE Transactions on Power Electronics*, 30(10):5650–5664, 2015.

[14] Yongjun Li, Jikang Chen, Mervin John, Ricky Liou, and Seth R Sanders. Resonant switched capacitor stacked topology enabling high dc-dc voltage conversion ratios and efficient wide range regulation. In *Proc. IEEE Energy Conversion Congress and Exposition*, pages 1–7, Milwaukee, WI, USA, September 2016.

[15] Seungbum Lim, John Ranson, David M. Otten, and David J. Perreault. Two-stage power conversion architecture suitable for wide range input voltage. *IEEE Transactions on Power Electronics*, 30(2):805–816, 2015.

[16] B. Pollet, G. Despesse, and F. Costa. A new non-isolated low-power inductorless piezoelectric dc–dc converter. *IEEE Transactions on Power Electronics*, 34(11):11002–11013, 2019.

[17] M. S. Rßdgaard, T. Andersen, and M. A. E. Andersen. Empiric analysis of zero voltage switching in piezoelectric transformer based resonant converters. In *6th IET International Conference on Power Electronics, Machines and Drives (PEMD 2012)*, 2012.

[18] Christopher Schaef and Jason T Stauth. A highly integrated series–parallel switched-capacitor converter with 12 V input and quasi-resonant voltage-mode regulation. *IEEE Journal of Emerging and Selected Topics in Power Electronics*, 6(2):456–464, 2018.

[19] G Seo, J Shin, and B Cho. A magnetic component-less series resonant converter using a piezoelectric transducer for low profile application. In *The 2010 International Power Electronics Conference - ECCE ASIA*, pages 2810–2814, 2010.

[20] Charles R Sullivan, Bradley A Reese, Aaron LF Stein, and Phyo Aung Kyaw. On size and magnetics: Why small efficient power inductors are rare. In *Proc. IEEE International Symposium on 3D Power Electronics Integration and Manufacturing*, pages 1–23, Raleigh, NC, USA, June 2016.

[21] M. Touhami, G. Despesse, and F. Costa. A new topology of dc-dc converter based on piezoelectric resonator. In *2020 IEEE 21st Workshop on Control and Modeling for Power Electronics*, pages 1–7, 2020.

[22] M. Touhami, G. Despesse, F. Costa, and B. Pollet. Implementation of control strategy for step-down dc-dc converter based on piezoelectric resonator. In *2020 22nd European Conference on Power Electronics and Applications (EPE'20 ECCE Europe)*, pages 1–9, 2020.

[23] Karl S Van Dyke. The piezo-electric resonator and its equivalent network. *Proceedings of the Institute of Radio Engineers*, 16(6):742–764, 1928.

[24] Z. Yang, J. Forrester, J. N. Davidson, M. P. Foster, and D. A. Stone. Resonant current estimation and phase-locked loop feedback design for piezoelectric transformer-based power supplies. *IEEE Transactions on Power Electronics*, 35(10):10466–10476, 2020.