Alternative Topologies for the Low-Voltage Buck Converter

by

Jonas Alan Whitney

S.B., Massachusetts Institute of Technology (2017)

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 2018

Author .....
Department of Electrical Engineering and Computer Science
January 29, 2018

Certified by .....
Serhii Zhak
Sr. Design Engineer
VI-A Company Thesis Supervisor

Certified by .....
David J. Perreault
Associate Professor
M.I.T. Thesis Supervisor

Accepted by .....
Christopher J. Terman
Chairman, Department Committee on Graduate Theses

Alternative Topologies for the Low-Voltage Buck Converter

by

Jonas Alan Whitney

Submitted to the Department of Electrical Engineering and Computer Science
on January 29, 2018, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In this thesis, investigative work on and development of alternative topologies for the buck converter for low voltage dc dc conversion was performed. The three level buck, Resonant Switch Capacitor (ResSC), and Ćuk-Buck2 were selected to be studied further based on the fact that they contain few components and were discovered in this work to have the possibility of operating at fixed frequency while smoothly regulating output voltage over the entire conversion ratio of 0 to 1. All three use a capacitive storage element in addition to a small inductance/s, so it was believed this may allow for efficiency or density improvements due to the excellent energy storage capability of MLCCs. New control methods were developed in order to operate the ResSC and Ćuk-Buck2 at fixed frequency over the entire output range. New work was done to in order to achieve flying capacitor balancing in the ResSC and Ćuk-Buck2, practical for future implementation in a monolithic converter. Simulated efficiency and other characteristics of the three converters are compared. Prototypes were built and used to confirm functionality of the new control schemes and balancing methods.

VI-A Company Thesis Supervisor: Serhii Zhak
Title: Sr. Design Engineer

M.I.T. Thesis Supervisor: David J. Perreault
Title: Associate Professor

Acknowledgments

I owe many thanks to my company supervisors Serhii Zhak and Tom Sheehan, who guided me through this project. Serhii spent a great deal of time teaching me about circuit design, which I am extremely grateful for, and hope to make good use of in my career. I also thank Sam Nork for allowing me the opportunity to work at the design center and I will miss all the friends I made there.

I thank Professor David Perreault for teaching me the fundamentals of power electronics and being my thesis adviser.

I wouldn't have make it through the past couple of years without the love and support of my fiancée Bonnie Wang. I wish that I will support you the same way you have me in the future and that we can grow together for the rest of our lives. My family has given me so much throughout my whole life. I will greatly miss my mom, dad, brother, and sister as I continue on in my life and move elsewhere.

I am forever thankful for God, who gave me life and a purpose for living.

Contents

List of Figures

Chapter 1

Introduction

With the continued increasing presence of digital processors in our world, a challenge of providing power to all of these chips exists. These processors push to be smaller, faster, and more efficient every year, in a cycle that has been reoccurring for a long time. A great invention to power these devices was the buck converter. This circuit is able to efficiency produce lower DC voltages from higher DC voltages and regulate its output via feedback. This allows energy to be transported at a higher voltage over distance and then stepped-down by a buck converter to the voltage needed by processors. This is much the same structure that power grids for towns, states, and countries have but on a significantly smaller scale and with DC instead of AC power. The buck converter still remains the best choice for many applications and is widely used.

Over the years this converter has improved as materials have improved and more fabrication techniques and knowledge have been discovered, but it is possible this converter may be reaching its limits. What then could the next step be if the buck converter has almost been perfected given its constraints?

One solution could be an alternative topology (a different circuit). Compared to the buck converter, other converters might be able to trade increased complexity for some ability that the buck converter does not have. This thesis focuses on alternative

converters, attempting to see whether they could be used commercially and could provide any benefits.

1.1 Low-Voltage DC DC Conversion

1.1.1 Applications

One main application for low-voltage DC-DC conversion is the powering of digital processors. As process technology improves, we are able to make smaller and smaller transistors which can use lower voltages to achieve higher energy efficiency or speed. Modern processors can draw currents in excess of 100A around 1V. Two examples utilizing Linear Technology conversion parts for a Xilinx FPGA[3] and an Altera FPGA[1] use 52A at 1V and 109A at 0.9V respectively. 100W may not be a ridiculous amount of power, but this power is usually transmitted at a higher voltage, so less current is carried over a longer distance. Common transmitted power voltages are 5V, 12V, and 48V. This is done for the same reason as high voltage power lines, as carrying power at higher voltage over distance leads to less loss.

1.1.2 Wanted Characteristics

Some wanted characteristics of these converters are fixed frequency switching and continuous conversion ratio. Having a fixed frequency means that electrical and magnetic noise created by the converter will mostly be kept within certain frequency ranges, which allows designers who use these converter to check for interference at the operational frequency. If the frequency were to change due to temperature, voltage, or current changes, the frequency may be such that the noise produced interferes with the circuit being powered. If the frequency can change due to many things, it would be hard to test if this would ever be a problem, so it is advisable to avoid the possible problem in the first place by always switching at an determined frequency independent of any conditions. A continuous conversion ratio is important to be able to maintain the output voltage well. Even if a single output voltage is required, the

input voltage is likely to change a bit with many factors, which means that a continuous ratio would be required to keep the output voltage correct. Switched capacitor converters lack this ability as they have an inherent conversion ratio/s that is tied to their topology. Their output from fixed frequency operation can be regulated with a linear regulator, however that reduces efficiency.

A full output range is also good, as it allows for a general use converter. If a topology is limited to only a 5:1 to 5:0 conversion ratio, it would not be able to convert from 5V to 2.5V or 3.3V. While this isn't necessary, this work is looking for a replacement for the buck converter which can do any conversion ration less than one.

1.2 The Buck Converter

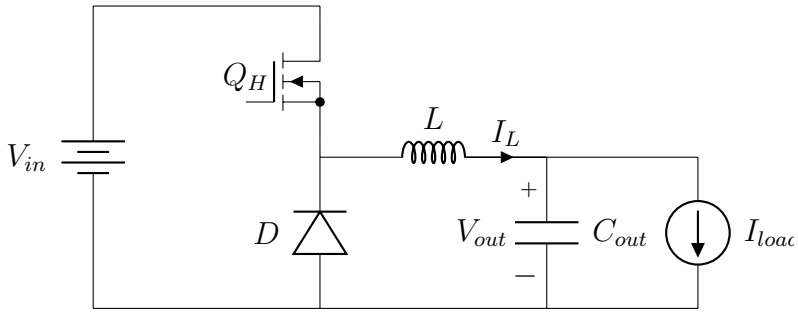


Figure 1-1: A buck converter with a constant current load.

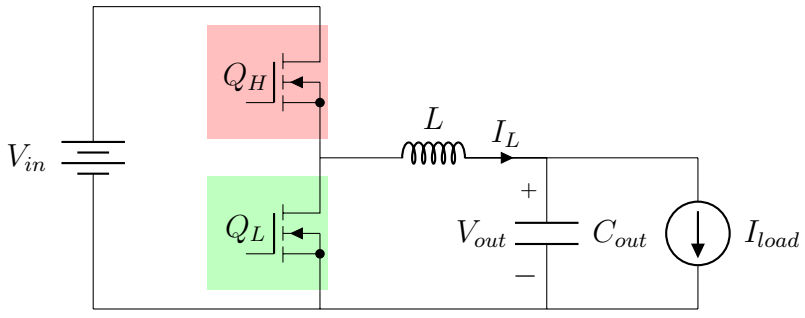


Figure 1-2: A synchronous buck converter with a constant current load.

1.2.1 Function

A standard low voltage buck converter is made up of two switches that create a square wave of voltage, which is then passed through an inductor and an output capacitor to create a DC voltage. It can be operated in two modes. One is where the inductor always has a non-zero voltage applied across it called continuous conduction mode (CCM), meaning the inductor current is always changing in time. The second is where the inductor current is always positive and allowed to remain at zero for some period of time, called discontinuous conduction mode (DCM) due to the inductor current remaining at zero at some point in the cycle.

Buck, CCM

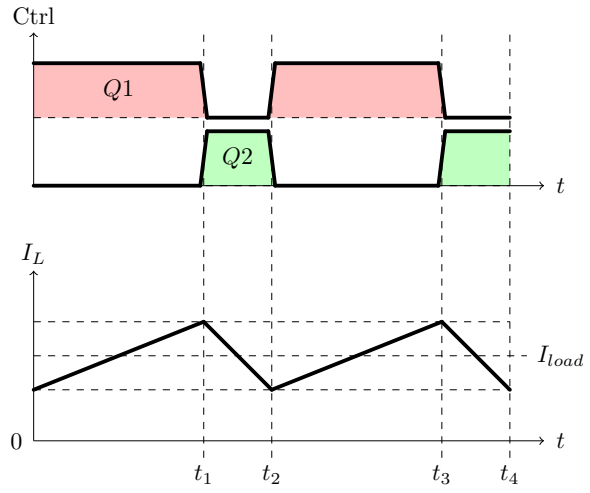


Figure 1-3: Buck Converter in CCM.

Buck, DCM

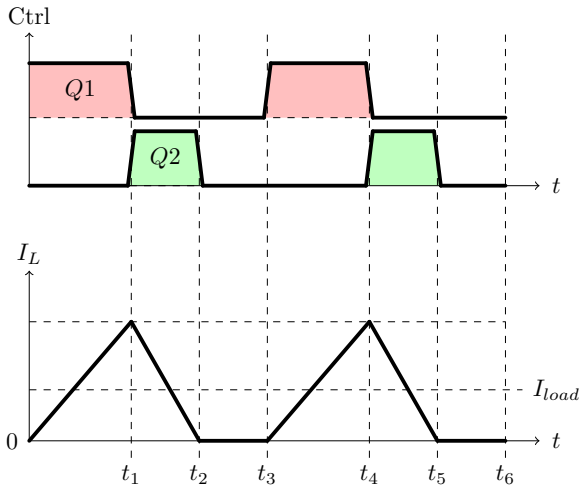


Figure 1-4: Buck Converter in DCM.

Either the top switch is on or the bottom switch is on in CCM, the top switch is on D of the time and the bottom switch is on (1-D) of the time, where D is a duty cycle between 0 and 1. The output voltage then ends up being D times the input voltage. In DCM the relationship between Vin and Vout at steady state is more complicated and depends on the output current.

1.2.2 Control

Feedback is added in order to maintain the output at a specific voltage. A number of different feedback types can be used, which will not be discussed in detail here, but are voltage mode feeedback and current mode feedback. The inductor current is sensed in current mode feedback and is regulated in order to maintain the output voltage, instead of just regulating D.

1.3 What Could Be Improved

Things that can be improved for these converters are better transient response, smaller size, higher efficiency, and lower cost.

1.3.1 Transient Response

A better transient response means the converter is better able to maintain the output voltage when its load changes. If it is powering a processor, a better response could mean less overhead is needed to make sure the processor always has enough voltage, which would increase efficiency and reliability. A better response could allow for smaller output capacitors, which would reduce size and cost.

1.3.2 Size

A smaller size allows the conversion circuit to take up a smaller part of a PCB or fit in a constrained space such at the boxes in a server room or in a mobile phone.

1.3.3 Efficiency

Higher efficiency means that less heat will be dissipated, so the circuit can run at a lower temperature which will increase its lifespan, or the size can be reduced more for a given power dissipation if thermal limits are the constraining factor for maximum power delivery.

1.3.4 Cost

Lower cost is fairly simple, it is better for either the seller (higher profit for a given selling price) or the buyer (cheaper to buy).

1.4 Focus Of This Work

This work will limit itself to fixed frequency, full output range, continuous conversion ratio, and high efficiency low voltage DC-DC converters. After preliminary research, the Three Level Buck, Ćuk-Buck2 , and Resonant Switched Capacitor (ResSC) were selected as converters of interest. It was found that the capabilities of the Three Level buck are relatively well known and it functions very similarly to a normal buck converter. The only new challenge is balancing the flying capacitor voltage. This has been studied in many papers (one example here[13] and a practical implementation will be discussed later in this document.

The Ćuk-Buck2 was not found to be in literature much,the author's only information coming from an online article[4] and a patent[9], but it was later found to operate in a very similar manner to the ResSC converter. It uses two inductors and a flying capacitor. One inductor is resonant and helps to balance the flying capacitor and the other inductor is larger and works similar to a Three Level Buck. Ordinarily this converter is limited to a 2:1 to 2:0 conversion ratio, but it will be shown later in this document that the full conversion range can be achieved with the addition of one more switch while retaining the normal functioning of the circuit and not affecting the converter in the 2:1 to 2:0 range.

Particular emphasis was placed on using printed circuit board (PCB) air core inductors, as they save on cost and possible increase efficiency due to lack of core losses. Papers were reviewed[7][10][6] and a resistance of 2mOhm/nH will be assumed for PCB trace inductors for the rest of this work. Some monolithic converters

are being packaged with small PCBs as modules (such as the LTM4650[2]), which could make use of this type of inductor. The three mentioned converters are able to use air core inductors at single digit MHz due to their topologies, whereas a normal buck converter may suffer unacceptable efficiency losses if air core inductors were to be used.

1.5 Previous Work

When looking at previous work, converters were looked for that had continuous conversion ratios, possibility of fixed frequency operation, and simplicity (avoiding unneeded complexity that adds size). Four topologies were found and three were evaluated for this work. Those three are the 3-Level Buck[13], Ćuk-Buck2 [4][9], and Resonant Switched Capacitor[8][5][12][11]. The one not evaluated was the Series Capacitor Buck, due to the limited output range.

Chapter 2

Topology Overviews

The diagrams shown in this section are idealized, which most importantly means that are no losses, the output capacitance is very large (so that output ripple is negligible compared to the DC output voltage), and the input impedance is zero.

2.1 3-Level Buck

2.1.1 Brief Description

The Three Level Buck converter[13] is very similar to the standard buck converter. For comparison, the standard buck would be called a two level converter using the same naming convention. The number of levels refers to the number of voltages that can be produced at the node connected to the side of the inductor opposite the output. The normal buck can produce V_{in} or 0V at this node, hence it is two level.

The three level buck utilizes an additional flying capacitor (C_{fly}) in order to create V_{in}, $V_{in}/2$, or 0V at this node. The flying capacitor nominally has a voltage of $V_{in}/2$, so $V_{in}/2$ on the switch node is achieved as just the flying capacitor voltage or V_{in} minus the flying capacitor voltage.

If the output voltage is desired to be less than $V_{in}/2$, then the switch node is con-

trolled to be either $V_{in}/2$ or 0V. If the output voltage is desired to be greater than $V_{in}/2$, then the switch node is controlled to be either V_{in} or $V_{in}/2$.

As mentioned in Chapter 1 with the standard buck converter, the Three Level Buck can operate in CCM or DCM in much the same way. The mode depends on the input and output voltages, L, the switching frequency, and possibly other factors.

2.1.2 Circuit Diagram

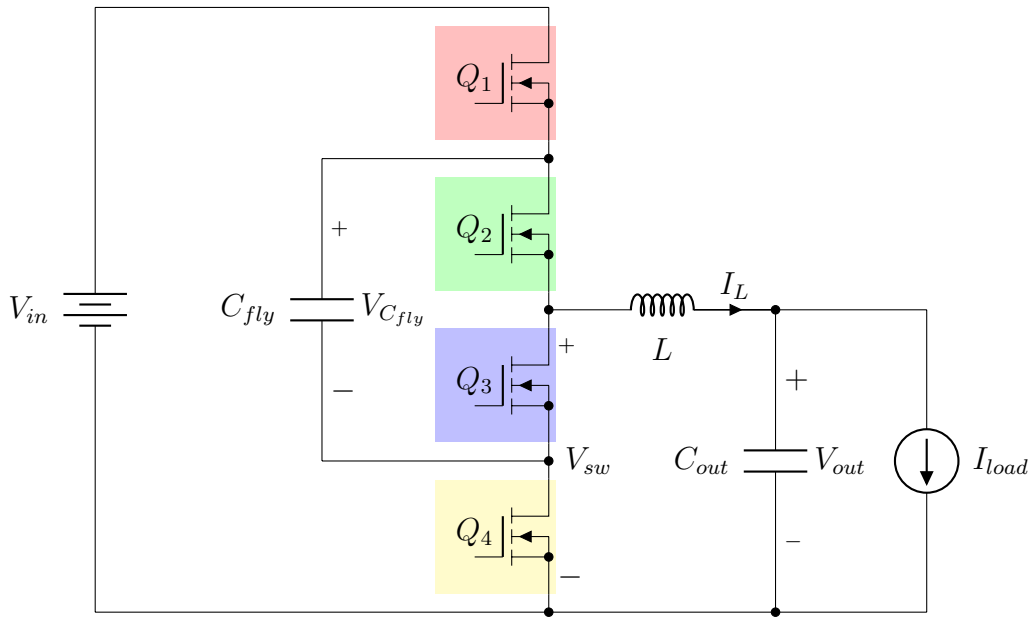


Figure 2-1: A three level buck converter.

2.1.3 Timing Diagram

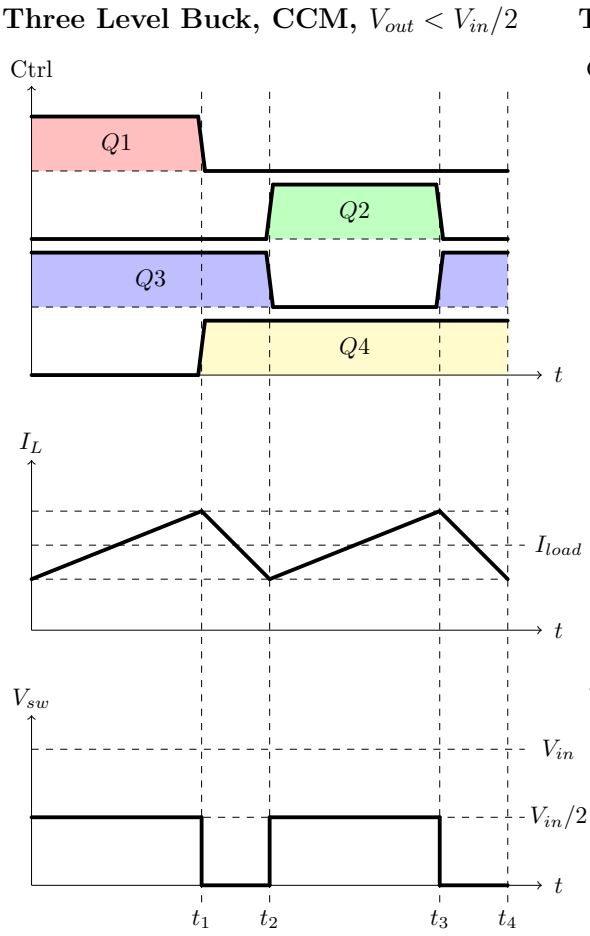


Figure 2-2: Timing diagram for Three Level Buck in CCM at low output voltage.

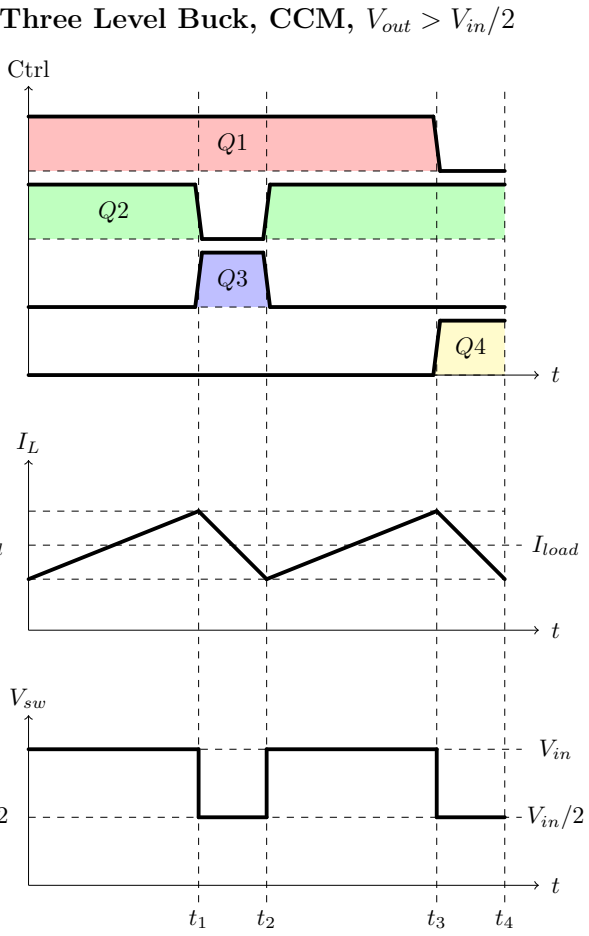


Figure 2-3: Timing diagram for Three Level Buck in CCM at high output voltage.

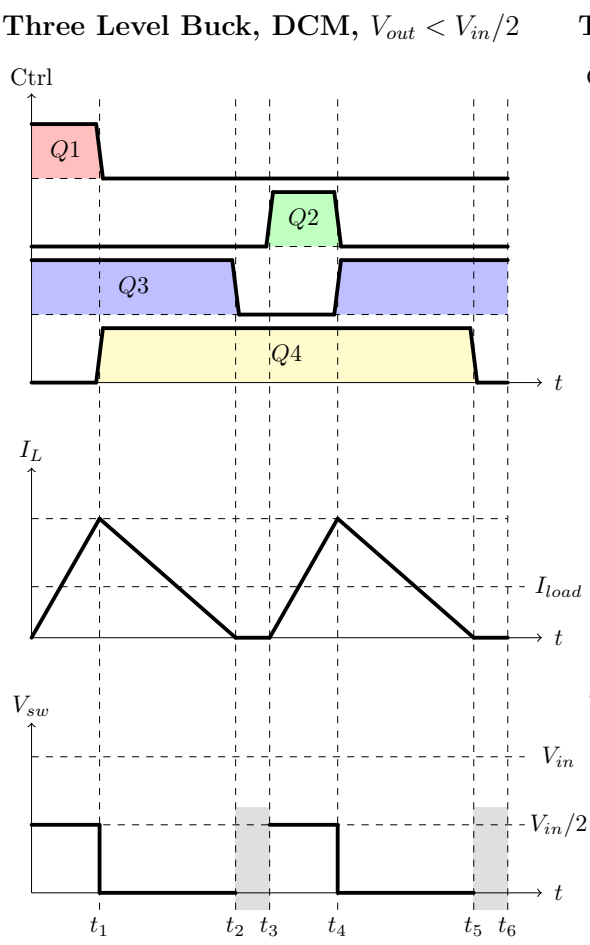


Figure 2-4: Timing diagram for Three Level Buck in DCM at low output voltage.

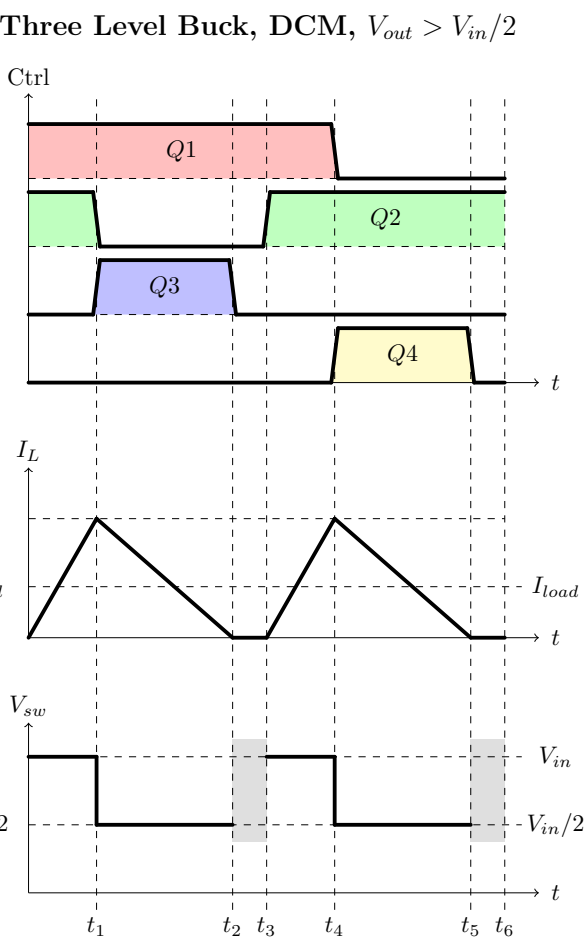


Figure 2-5: Timing diagram for Three Level Buck in DCM at high output voltage.

If the output voltage is exactly equal to half the input voltage with the idealities assumed to plot these graphs, the inductor current would remain exactly the same always, since there would never be a voltage across it. This is not the case in reality as the flying capacitance is not infinite, so the flying capacitor voltage does not remain constant throughout a switching period.

Three Level Buck, $V_{out} = V_{in}/2$

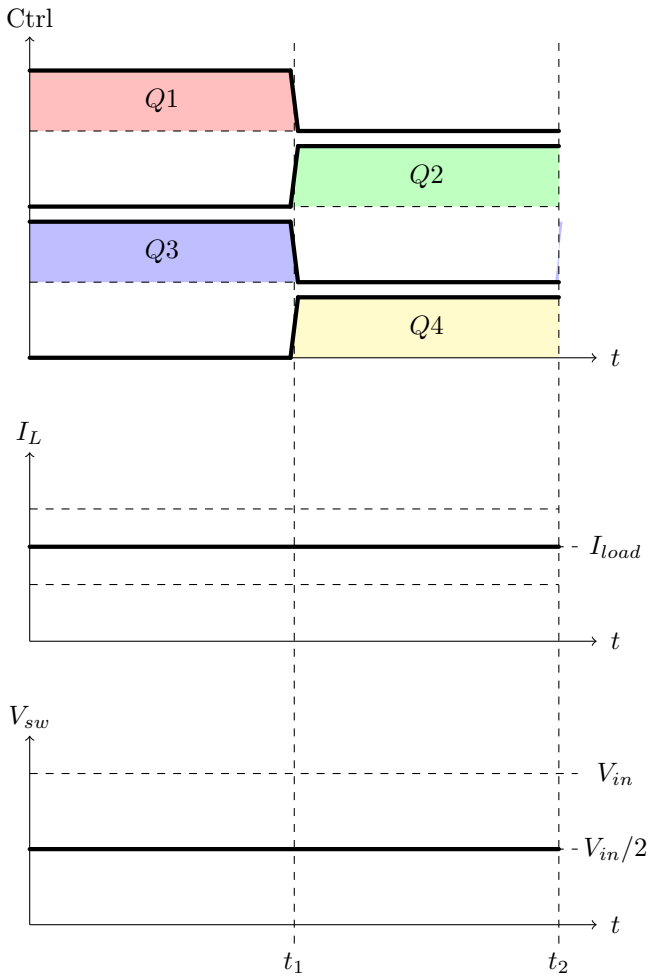


Figure 2-6: Timing diagram for Three Level Buck at middle output voltage.

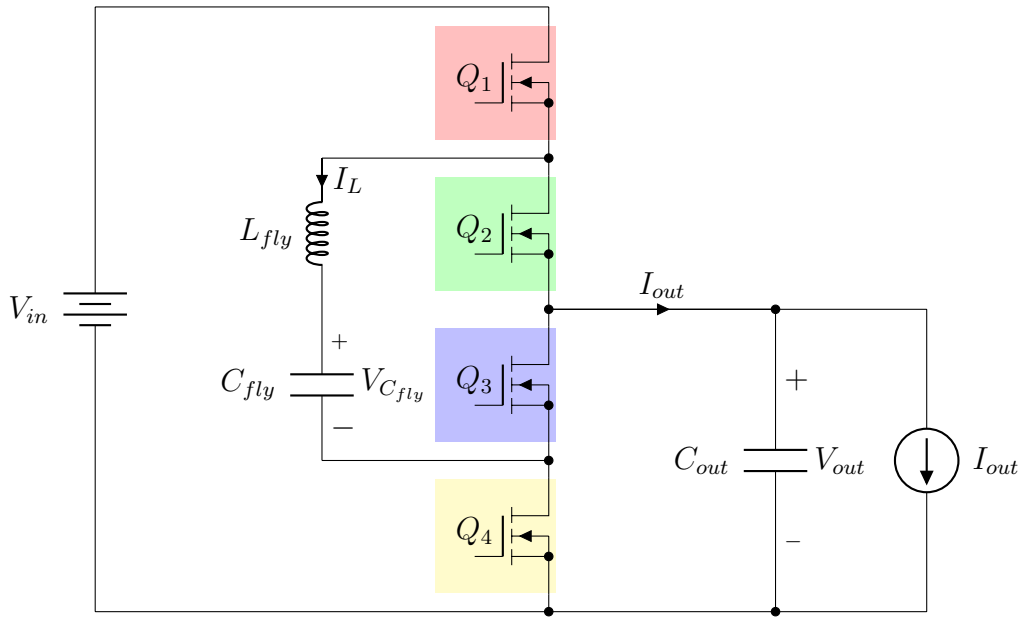
2.2 Resonant Switched Capacitor (ResSC)

2.2.1 Brief Description

The Resonant Switched Capacitor converter[8][5][12][11] is a 2 to 1 switched capacitor circuit with an inductor inserted to produce resonant switching operation. This converter can still be operated as a 2 to 1 converter, but the inductor allows for two additional states that can be utilized to smoothly regulate the output voltage.

State notation will be borrowed from [12] which labeled the four possible states A, B, C, and D. If the output voltage is desired to be less than $V_{in}/2$, then only states B, C, and D will be present. If the output voltage is desired to be greater than $V_{in}/2$, then only states A, B, and D will be present. If the output voltage is desired to be equal to $V_{in}/2$, then an ideal converter would only have states B and D. The control methods seen in these diagrams are taken from [12].

2.2.2 Circuit Diagram



2.2.3 Timing Diagram

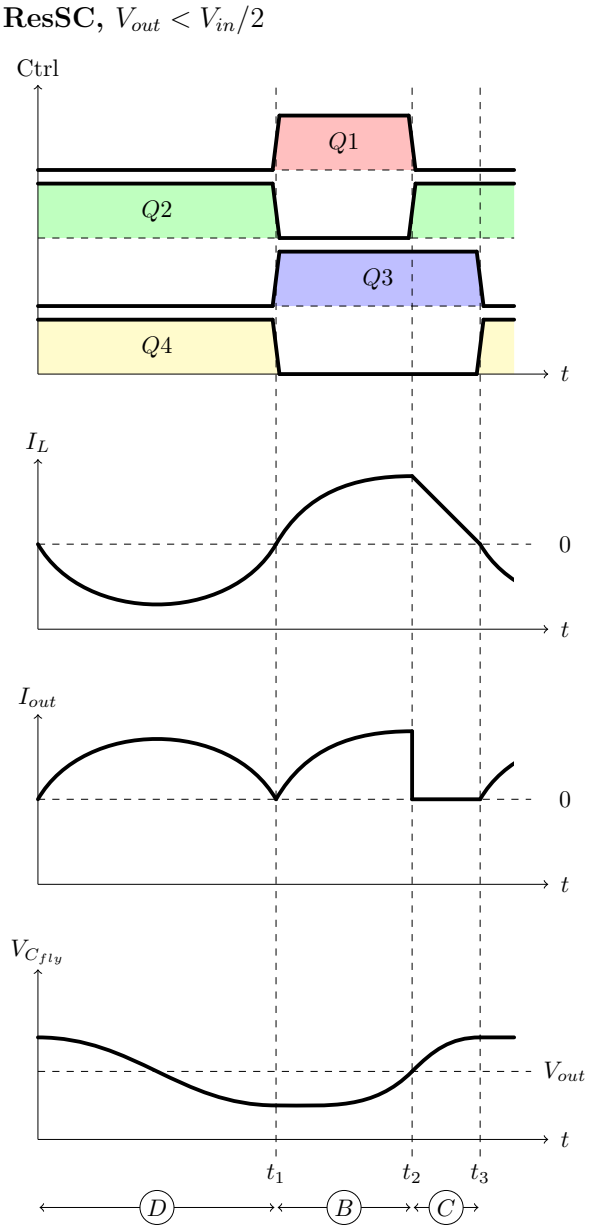


Figure 2-7: Timing diagram for Resonant Switched Capacitor at low output voltage.

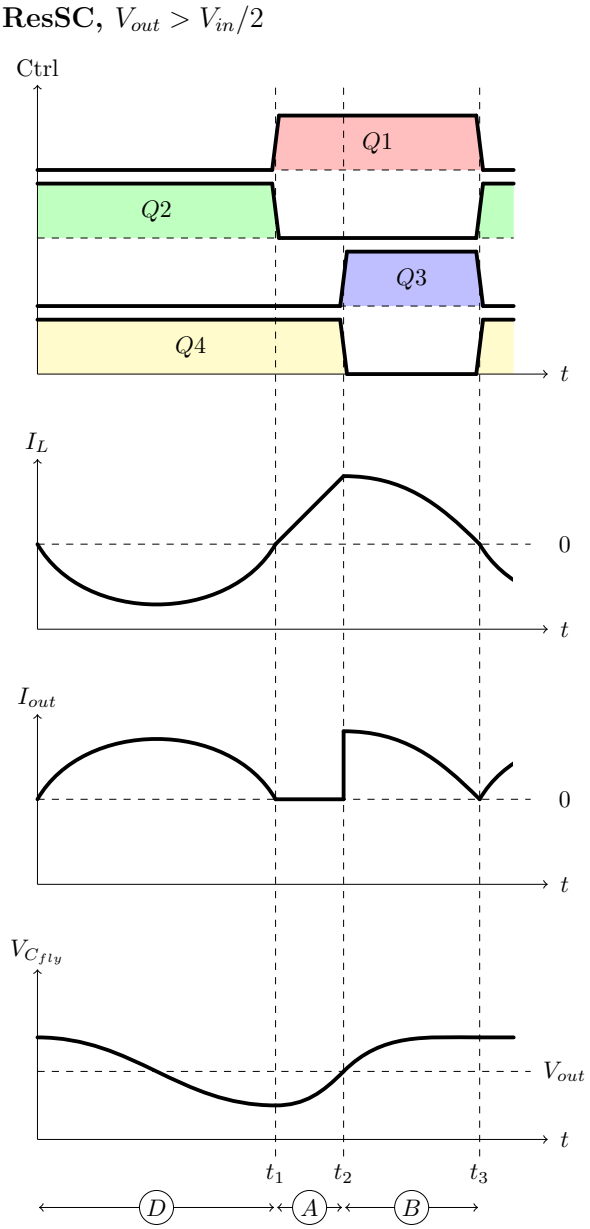


Figure 2-8: Timing diagram for Resonant Switched Capacitor at high output voltage.

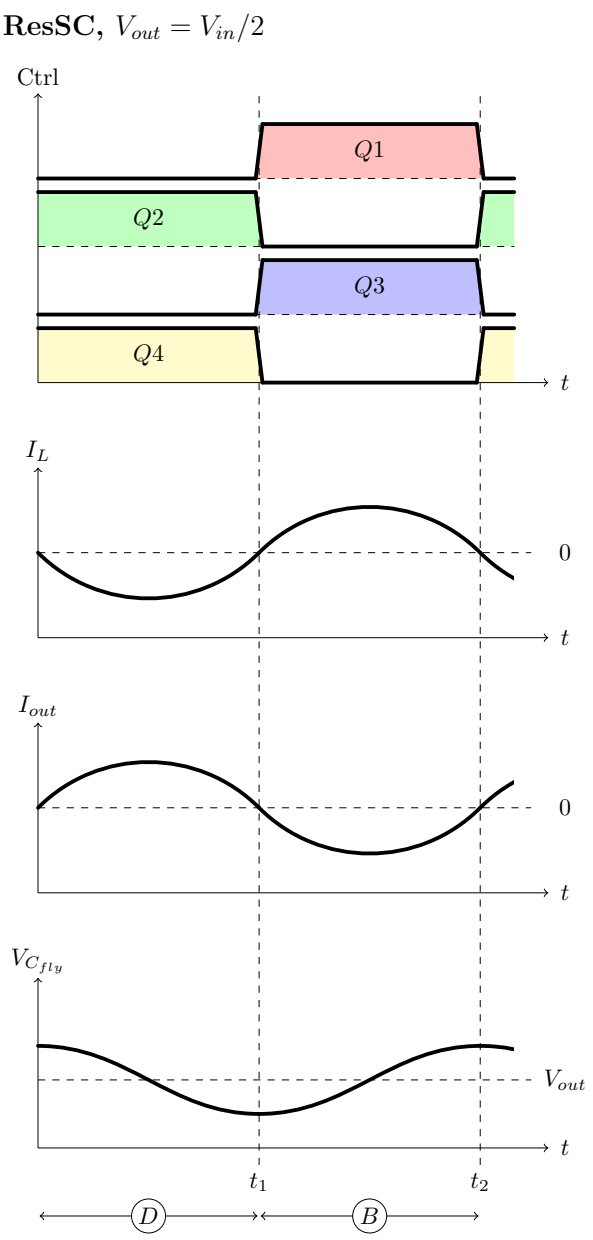


Figure 2-9: Timing diagram for Resonant Switched Capacitor at middle output voltage.

2.3 Ćuk-Buck2

2.3.1 Brief Description

The Ćuk-Buck2 [4][9] converter operates similarly to the Three Level Buck. In this thesis, will be extended in a later chapter for full output range with the addition of a fifth switch. It uses an additional "flying" capacitor to produce $V_{in} - V_{out}$, or 0V on the node connected to the side of the inductor opposite to the output. This converter also has a resonant inductor (L_r) which is used to reset the flying capacitor voltage.

2.3.2 Circuit Diagram

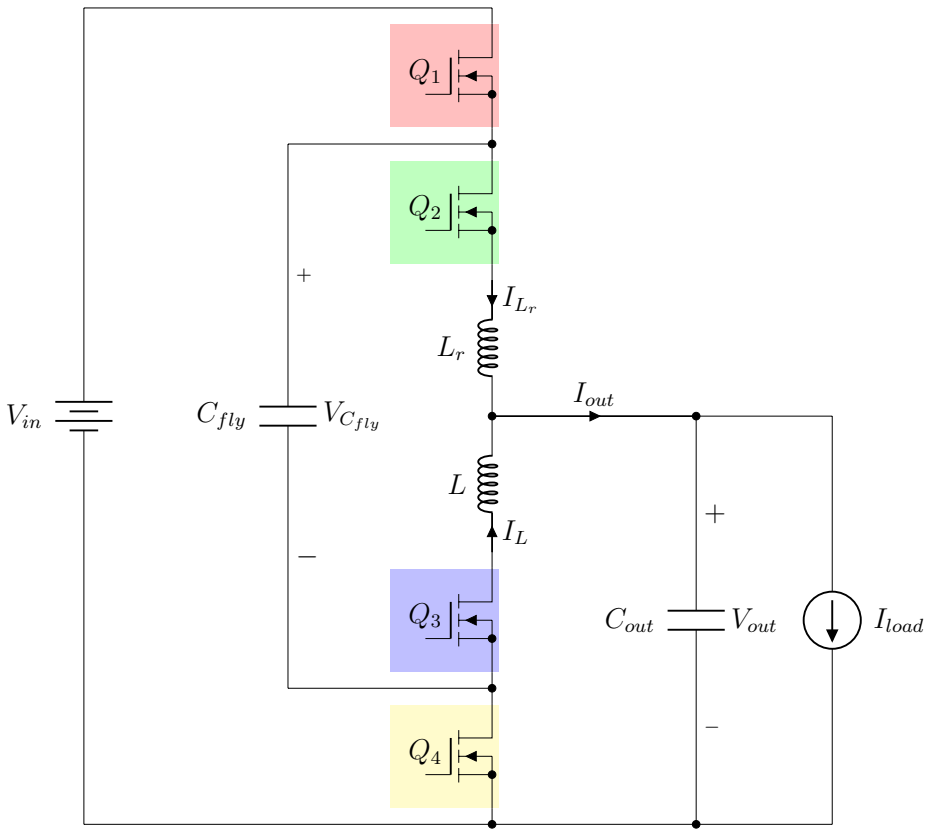


Figure 2-10: Ćuk-Buck2 with second resonant inductance L_r.

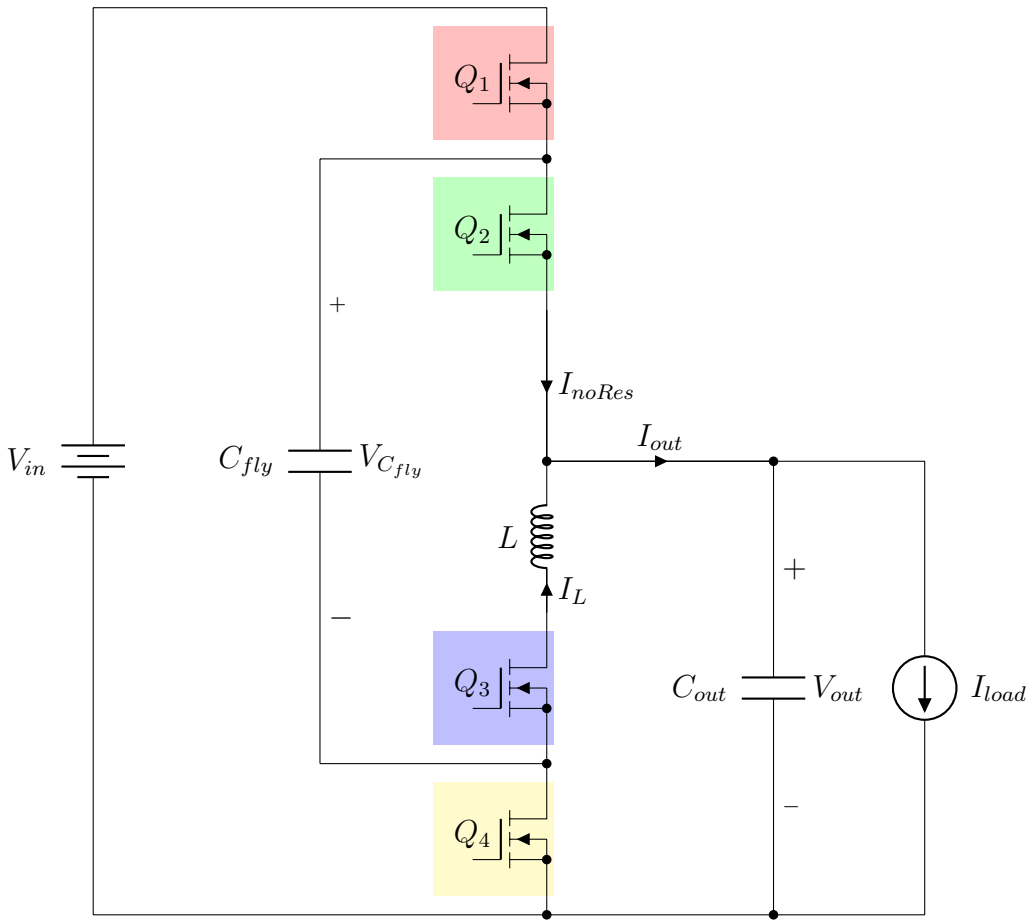


Figure 2-11: Ćuk-Buck2 without second resonant inductance L_r.

2.3.3 Timing Diagram

CukBuck2 with Lres, $V_{out} < V_{in}/2$

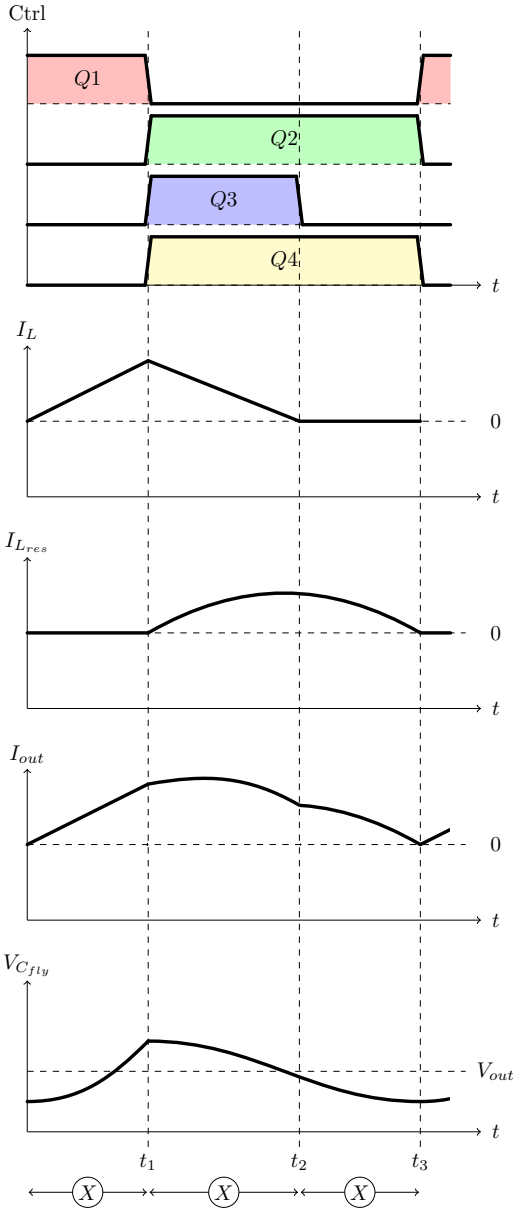


Figure 2-12: Timing diagram for Ćuk-Buck2 with second resonant inductance L_r at low output voltage.

CukBuck2 without Lres, $V_{out} < V_{in}/2$

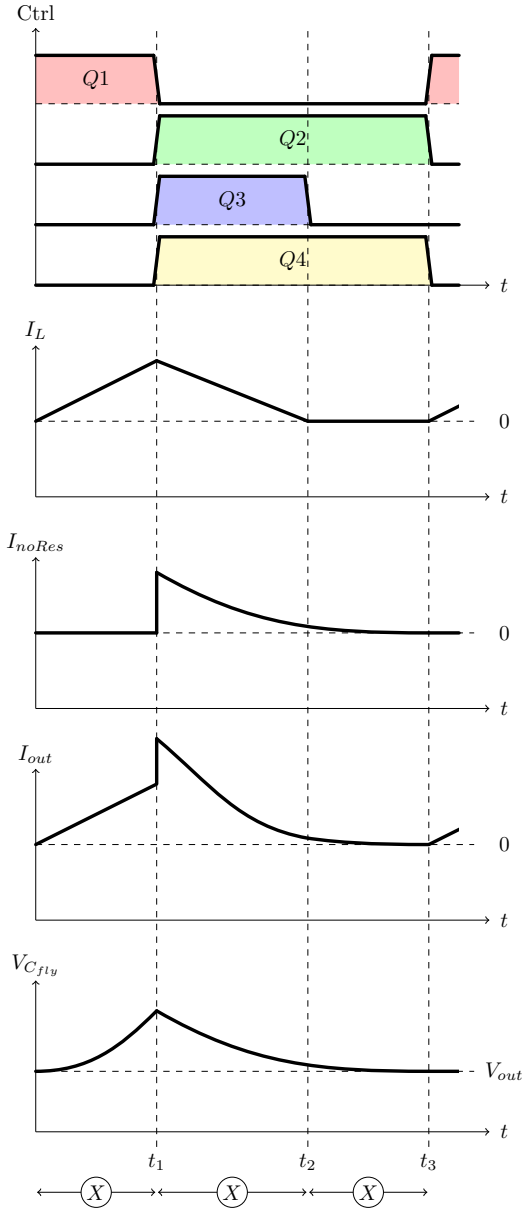


Figure 2-13: Timing diagram for Ćuk-Buck2 without second resonant inductance L_r at low output voltage.

Chapter 3

Topology Work Completed

3.1 Three Level Buck

3.1.1 Design Constraints

The flying capacitor for the Three Level Buck[13] should be large enough such that its voltage does not swing too much with the maximum output current. Some designs may allow for large flying voltage changes, but those were not considered for this work, as a standard Three Level Buck maintains a relatively constant flying capacitor voltage. The inductor must be significantly larger than the output capacitor's parasitic inductance in order to have reasonable ripple, which is important when dealing with very small PCB trace inductors, as larger values take up much more space.

3.1.2 Capacitor Balancing

The flying capacitor in the Three Level Buck does not maintain the correct voltage by itself. If the converter was perfectly controlled and lossless, then the average flying voltage would remain the same. In reality the control will not be perfect, which can lead to different duty cycles period to period along with other perturbations that could force the flying capacitor voltage higher or lower than nominal.

It was found in simulations that a lossy converter provides some restoring force for

this voltage, but not significant enough that the voltage does not need to be actively maintained. A mechanism inherent to the topology provides an increased restoring force in DCM, which could be large enough to maintain the correct voltage in some cases, however active balancing will ensure the right voltage is present.

In non-phase correct PWM control, varying ctrl1 and ctrl2 (see Figure 3-4) by opposite amounts will either increase or decrease the flying capacitor voltage depending on the direction they are varied. This allows a second slower control loop to vary ctrl1 and ctrl2 to control the flying capacitor voltage. If the sum of ctrl1 and ctrl2 stays the same, the average output voltage is not affected, although some additional ripple will be present. This ripple remains small unless the flying cap voltage gets significantly away from $V_{in}/2$ and only small shifts in ctrl1 and ctrl2 are normally needed to maintain this.

3.1.3 Mode Transition

It was found that the transition around an output of $V_{in}/2$ produced instabilities in simulation, most likely due to the fact that the converter can enter resonant states when the output is very close to $V_{in}/2$.

A solution to this was to not use a perfect phase shift of 180deg, but instead 162deg, which is 0.45 of a period, instead of 0.5 of a period like normal. This appeared to allow the converter to avoid entering the resonant states around $V_{out} = V_{in}/2$ and provide smooth output voltage regulation over the entire output range. Active flying capacitor balancing must be used, otherwise this shifted phase will cause the flying capacitor voltage to drift.

Three Level Buck, CCM, Capacitor Balancing, $V_{out} < V_{in}/2$

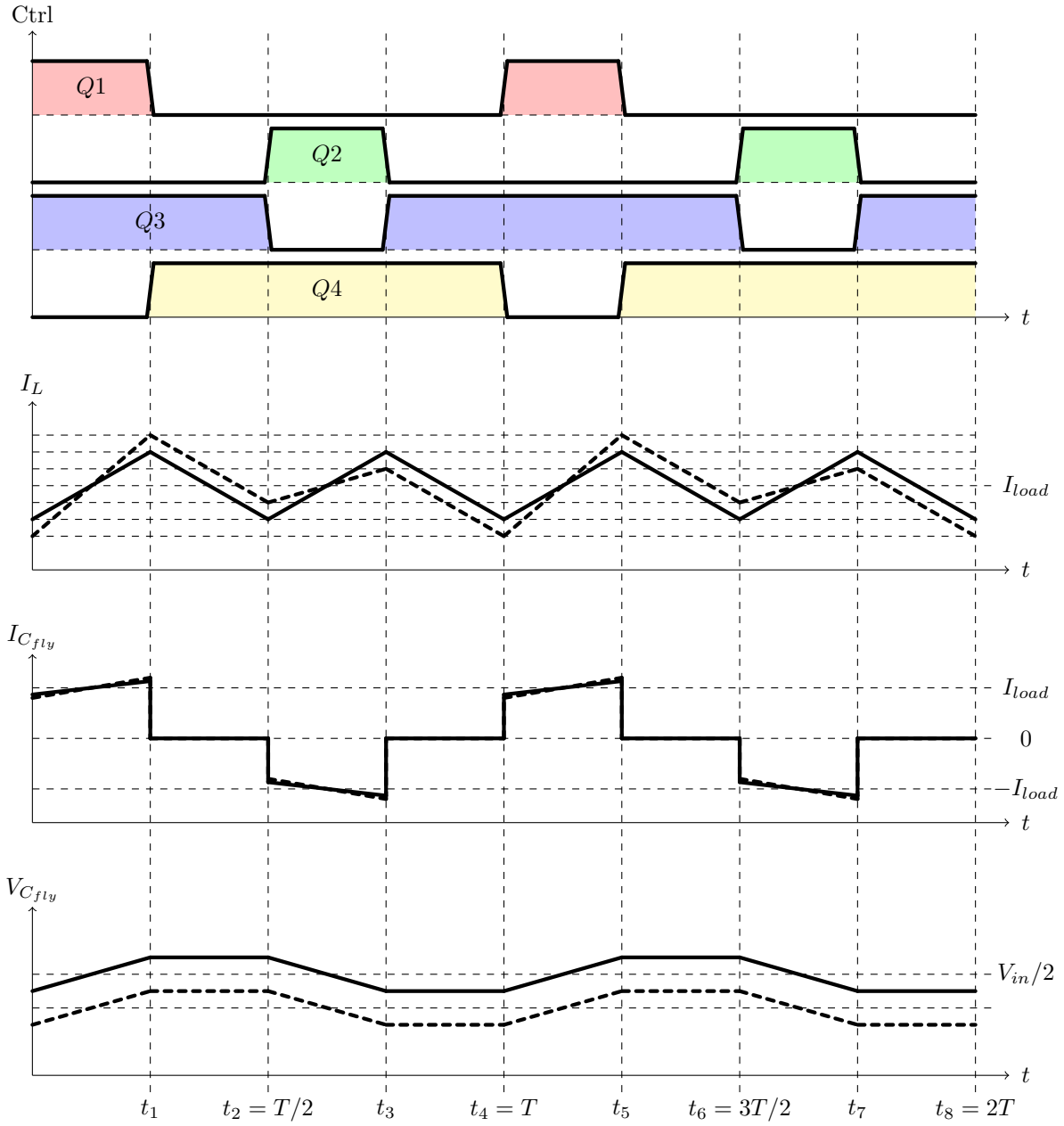


Figure 3-1: A Three Level Buck operating in CCM with $V_{out} < V_{in}/2$. Dashed lines are plotted for I_L, $I_{C_{fly}}$, and V_{Cfly} which show the result if $V_{C_{fly}}$ was a bit lower than its nominal value of $V_{in}/2$. No self-correction of the flying capacitor voltage is seen.

Three Level Buck, CCM, Capacitor Balancing, $V_{out} > V_{in}/2$

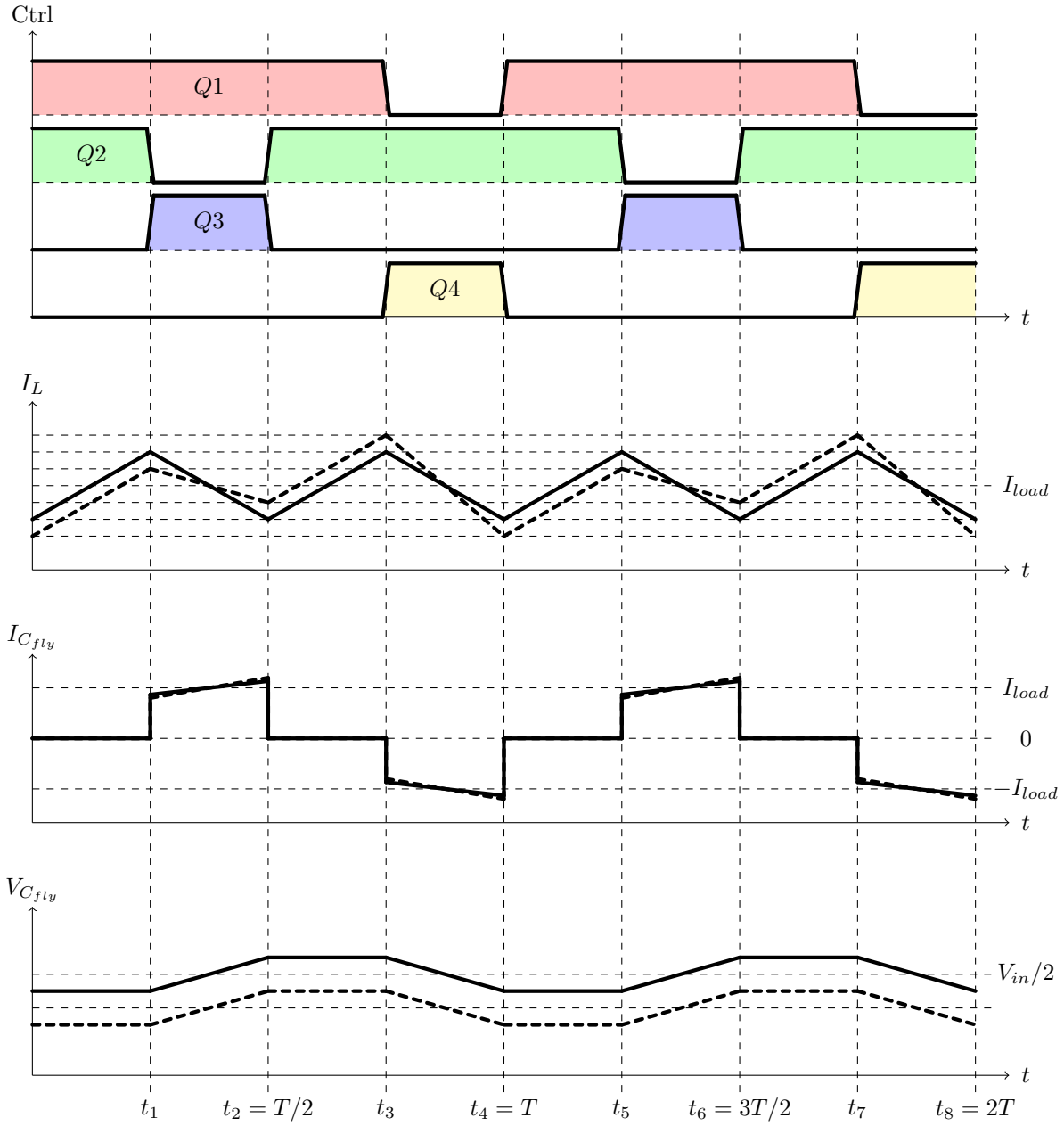


Figure 3-2: A Three Level Buck operating in CCM with $V_{out} > V_{in}/2$. Dashed lines are plotted for I_L, $I_{C_{fly}}$, and V_{Cfly} which show the result if $V_{C_{fly}}$ was a bit lower than its nominal value of $V_{in}/2$. No self-correction of the flying capacitor voltage is seen.

Three Level Buck, Capacitor Balancing, $V_{out} = V_{in}/2$

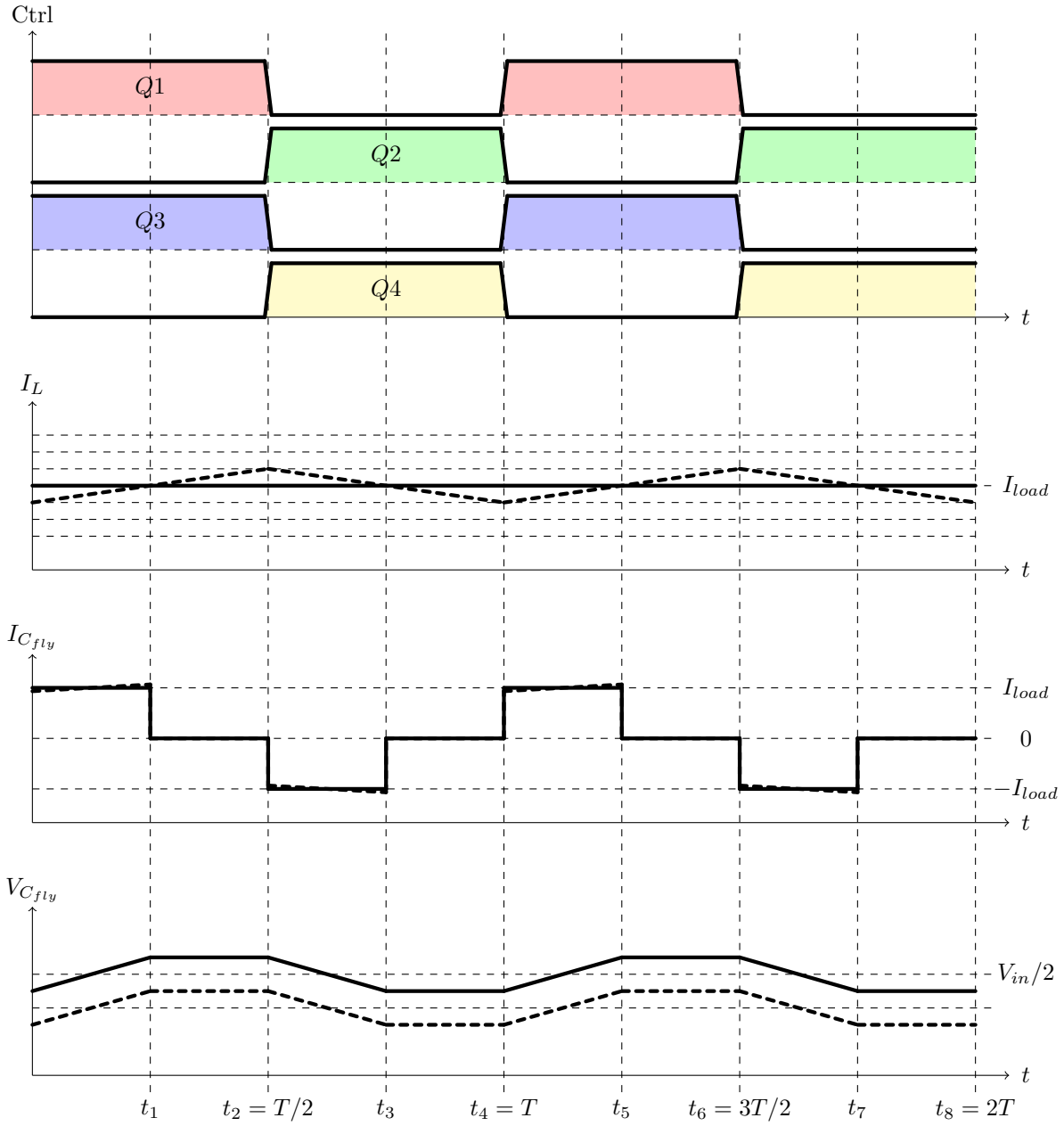


Figure 3-3: A Three Level Buck operating with $V_{out} \approx V_{in}/2$. Dashed lines are plotted for I_L, $I_{C_{fly}}$, and V_{Cfly} which show the result if $V_{C_{fly}}$ was a bit lower than its nominal value of $V_{in}/2$. No self-correction of the flying capacitor voltage is seen.

Three Level Buck, Fixed Frequency and Smooth Control, $V_{out} < V_{in}/2$

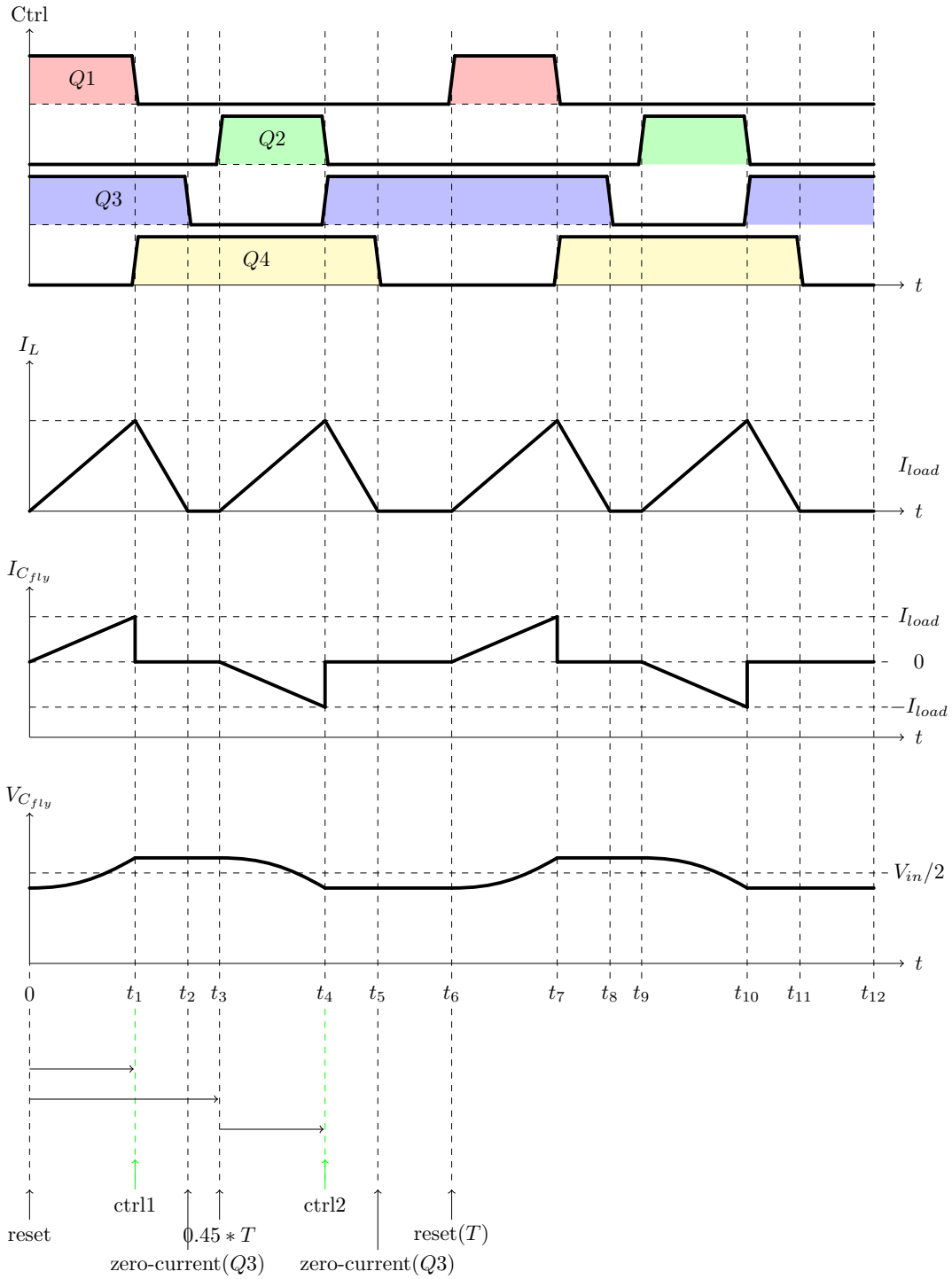


Figure 3-4: A Three Level Buck operating with $V_{out} < V_{in}/2$ at a fixed frequency with period T. A *ctrl1* timing is set by feedback and is measured from *reset*. Another *ctrl2* timing is offset from *0.45*T* by a fixed amount.

3.1.4 DCM operation

Overall there is nothing particularly special about DCM operation as compared with CCM, except that the duty cycle to output voltage transfer function is no longer linear. It is better approximated by being proportional to the square of the duty cycle, as an increase in duty cycle increases both the peak and width of the triangular current waveform, leading to a square relationship with the current. In order to counteract this and maintain a similar transient response over the range of output voltages and currents, the duty cycle from a CCM control circuit can be passed through a square root transform to control a converter in DCM operation.

3.1.5 Low Voltage Switch Possibility

If started up and controlled while keeping the maximum voltage across the switches minimized, it would be possible to use lower voltage rated switches than the input power rail. This would lead to an increase in efficiency, as lower voltage switches have a lower $R_{on} * Q_g$ factor, so the combined switching and conduction losses for the MOSFETs could be reduced.

This would add additional control complexity, because the lower voltages would have to be maintained across the switches at all times, including startup, where the flying capacitor could have zero voltage.

It was chosen to use switches rated for the full input voltage in this work for increased robustness against fault states and to reduce complexity.

Three Level Buck, DCM, Capacitor Balancing, $V_{out} < V_{in}/2$

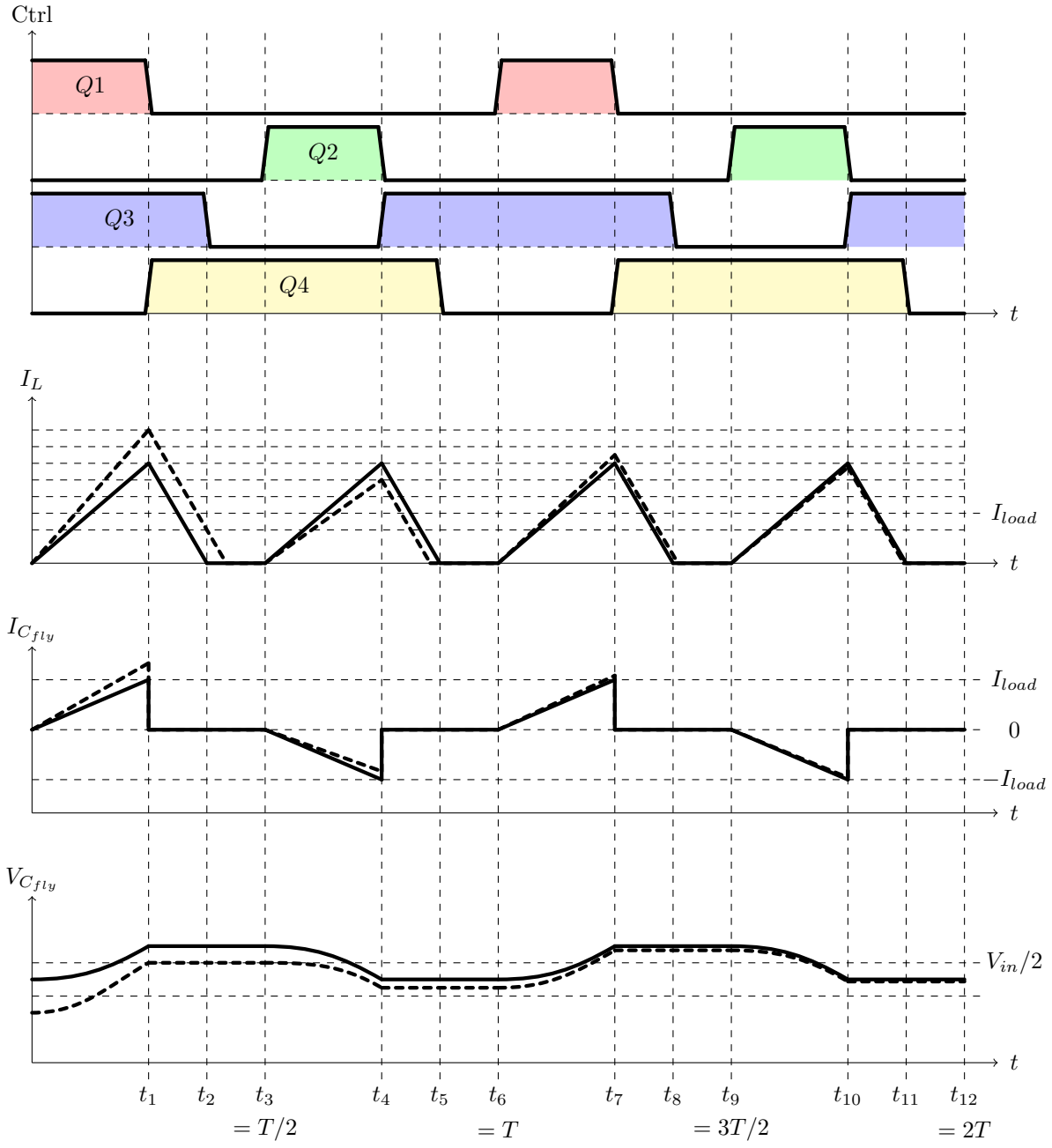


Figure 3-5: A Three Level Buck operating in DCM with $V_{out} < V_{in}/2$. Dashed lines are plotted for I_L, $I_{C_{fly}}$, and V_{Cfly} which show the result if $V_{C_{fly}}$ was a bit lower than its nominal value of $V_{in}/2$. $Q1$, $Q2$, $Q3$, and $Q4$ logic are only plotted for the nominal flying capacitor voltage, timing for the lower voltage would be very slightly different. Self-correction of the flying capacitor voltage can seen.

Three Level Boost, DCM, Capacitor Balancing, $V_{out} > V_{in}/2$

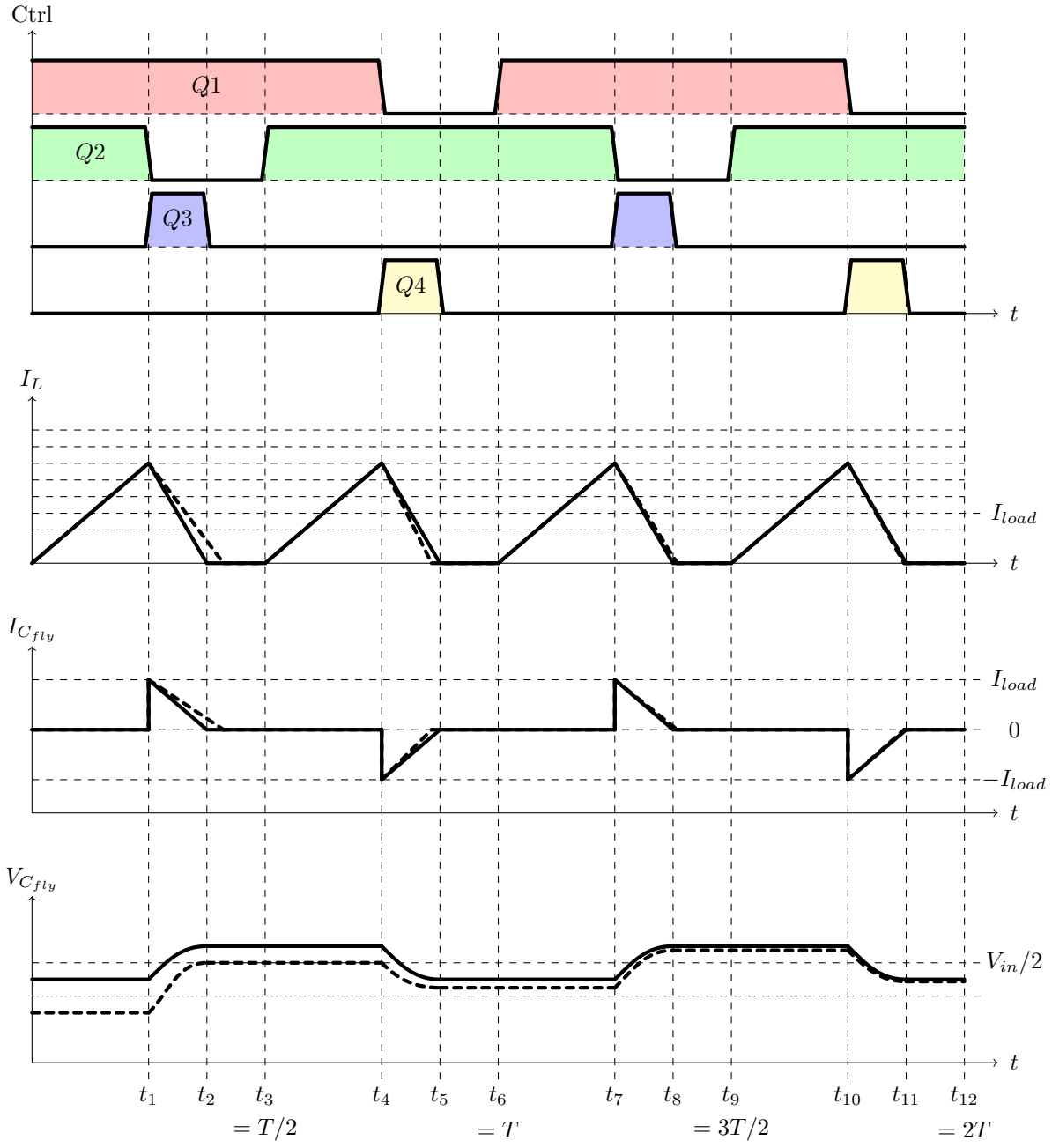


Figure 3-6: A Three Level Buck operating in DCM with $V_{out} > V_{in}/2$. Dashed lines are plotted for I_L, $I_{C_{fly}}$, and V_{Cfly} which show the result if $V_{C_{fly}}$ was a bit lower than its nominal value of $V_{in}/2$. $Q1$, $Q2$, $Q3$, and $Q4$ logic are only plotted for the nominal flying capacitor voltage, timing for the lower voltage would be very slightly different. Self-correction of the flying capacitor voltage can seen.

3.2 ResSC

3.2.1 Fixed Frequency Operation

It is possible to have the ResSC[8][5][12][11] run at a fixed switching frequency by inserting a blank time every period. During this blank time no circuit elements will change state except for the output capacitor being discharged by the load.

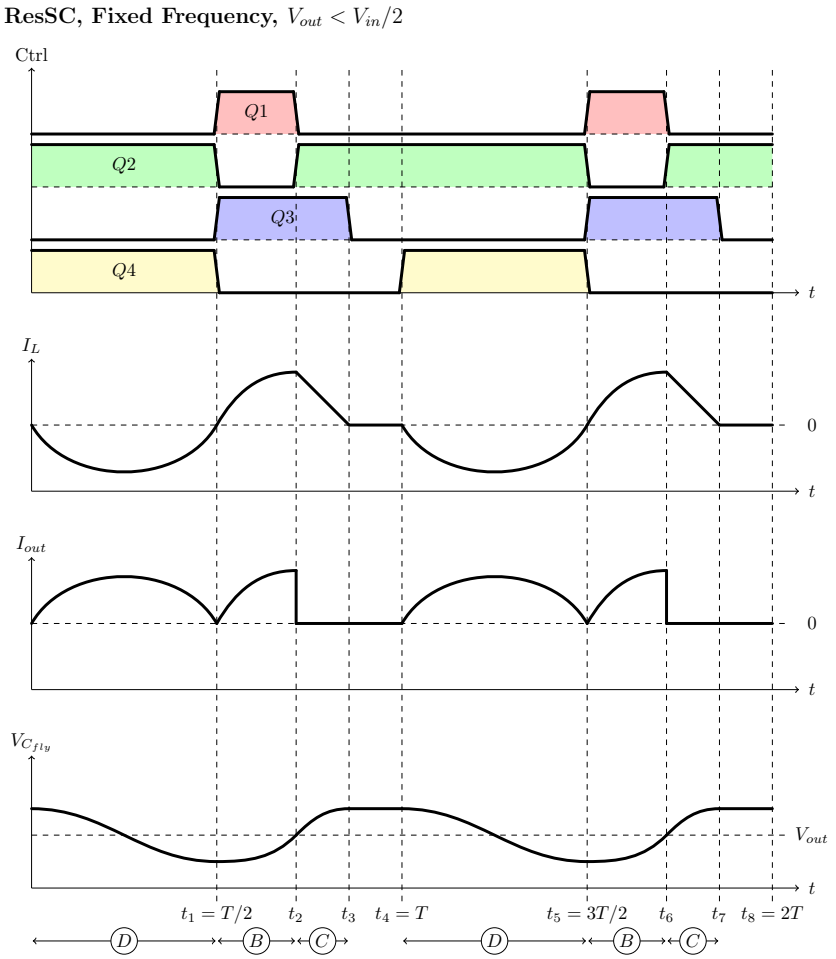


Figure 3-7: A ResSC operating with $V_{out} < V_{in}/2$ at a fixed frequency with period T. This is the same operation as Figure 2-7 with a blank period inserted from t_3 to t_4 where only $Q2$ is on. The flying capacitor must not be above $V_{out} + V_{bodyDiode}$ during this period, or unintended currents will reduce efficiency.

This blank time can by only having one switch active. This work chooses to have Q2 remain active during these blank periods. It is important that if switches with body diodes are used, that the peak to peak voltage swing on the flying capacitor is not larger than two times the forward voltage of these diodes. If so, large currents will flow through the body diodes during this off time and reduce efficiency. This will constrain the capacitor size, switching frequency, and output current. Otherwise the circuit can operate normally.

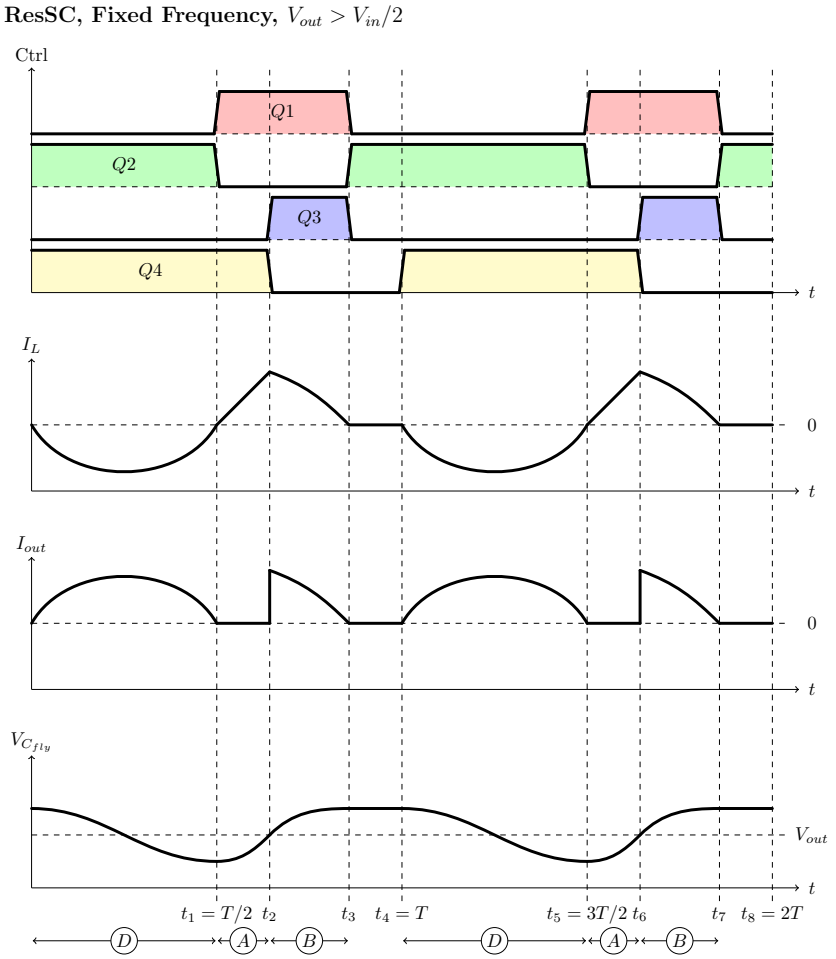


Figure 3-8: A ResSC operating with $V_{out} < V_{in}/2$ at a fixed frequency with period T. This is the same operation as Figure 2-8 with a blank period inserted from t_3 to t_4 where only $Q2$ is on. The flying capacitor must not be above $V_{out} + V_{bodyDiode}$ during this period, or unintended currents will reduce efficiency.

3.2.2 Smooth Transitioning Full-Range Output

The converter operates in two different modes, when $V_{out} > Vin/2$ or $V_{out} > Vin/2$. As V_{out} approaches $V_{in}/2$, there must exist a way to transition smoothly from one mode to the other. The output would jump and possibly oscillate if there was no transition and no hysteresis. Ideally, there should be no visible indication on the output voltage that the mode has switched.

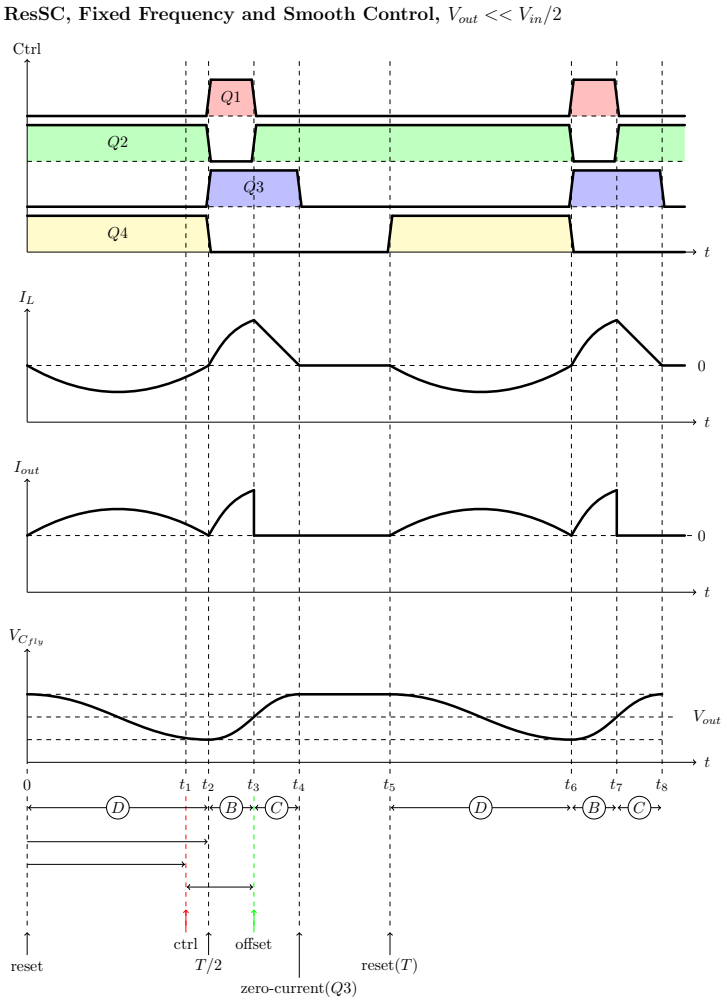


Figure 3-9: A ResSC operating with $V_{out} < V_{in}/2$ at a fixed frequency with period T. This is the same operation as Figure 3-7 with control timing events listed below. A *ctrl* timing is set by feedback and is measured from *reset*. An *offset* timing is offset from *ctrl* by a fixed amount. The timing *ctrl* has no effect when it comes before $T/2$, so it is pictured in red.

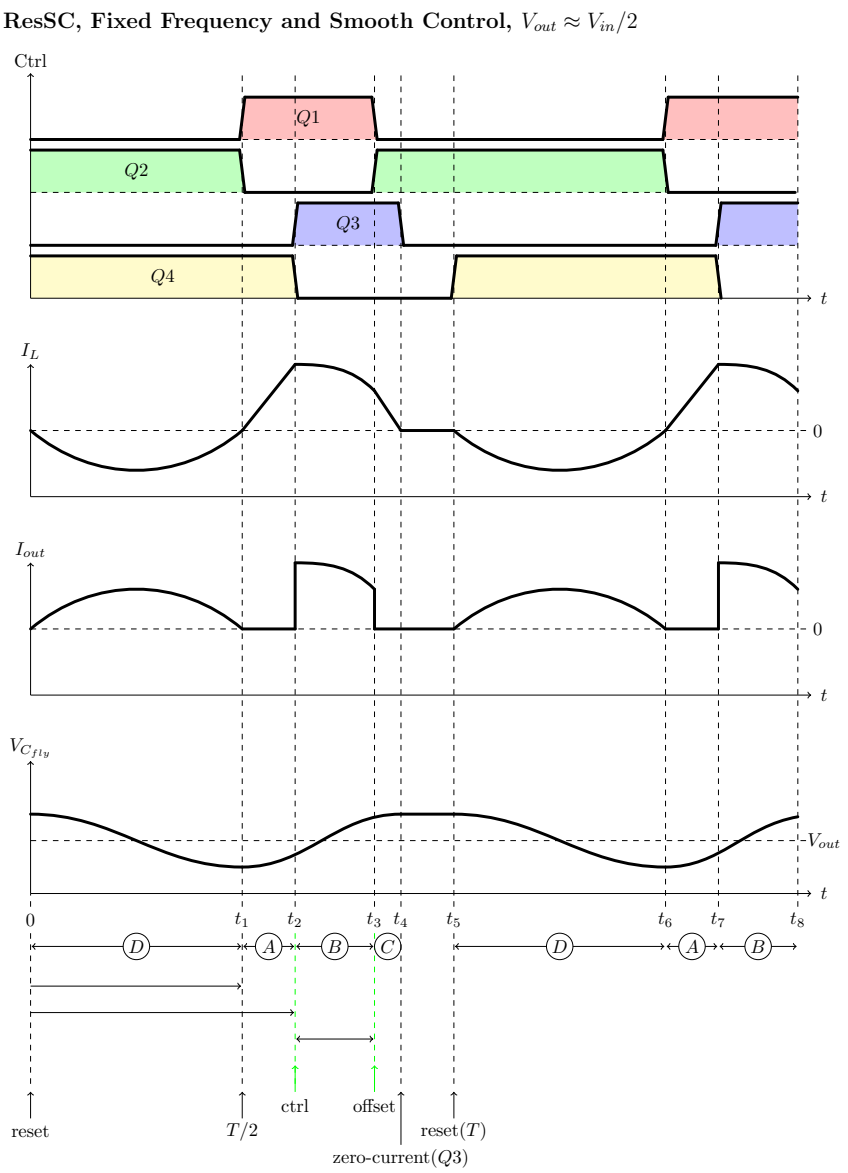


Figure 3-10: A ResSC operating with $V_{out} \approx V_{in}/2$ at a fixed frequency with period T. This is a joined operation with elements from Figure 3-7 and Figure 3-8 with control timing events listed below. A *ctrl* timing is set by feedback and is measured from *reset*. An *offset* timing is offset from *ctrl* by a fixed amount.

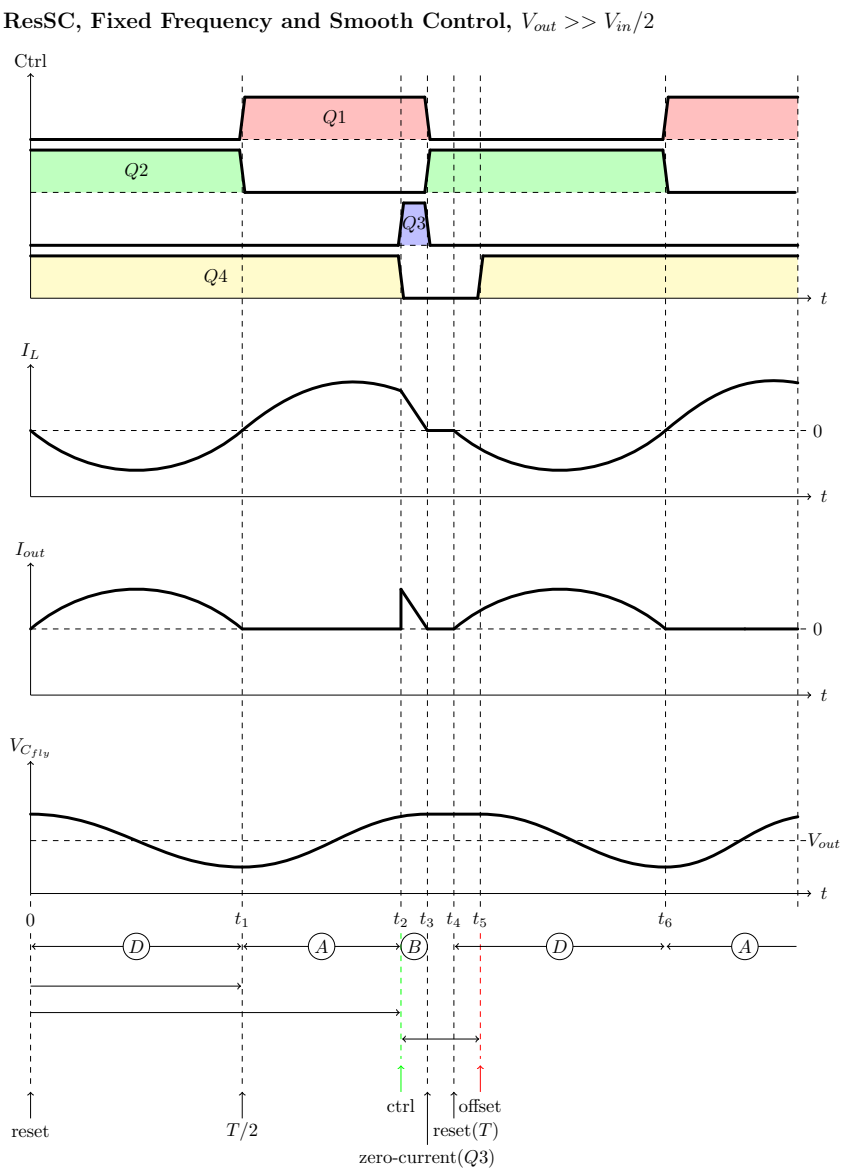


Figure 3-11: A ResSC operating with $V_{out} > V_{in}/2$ at a fixed frequency with period T. This is the same operation as Figure 3-8 with control timing events listed below. A *ctrl* timing is set by feedback and is measured from *reset*. An *offset* timing is offset from *ctrl* by a fixed amount. The timing *offset* has no effect when it comes after *reset(T)* or *zero-current(Q3)*, so it is pictured in red.

3.2.3 Capacitor Balancing

It was found that the flying capacitor can exhibit sub-harmonic oscillation. Oscillation at half the switching frequency was seen and it noticeably impacts output ripple and regulation in simulation. In extreme cases it can even cause the body diodes of switched to conduct where they would not normally.

It was found that the feedback network was integral in producing these oscillations. With no feedback network there is a self-correcting aspect, where the flying capacitor voltage will naturally return to where it should be.

It was found the control voltage varying cycle to cycle can cause this oscillation to occur in an otherwise normally operating converter, so a notch filter was placed at half the switching frequency in the control voltage. This successfully stopped the sub-harmonic oscillations. A sample and hold circuit for the control voltage was also tried, and had the same effect.

3.2.4 "DCM" Mode

When there is not current being pushed to the output (load current is very small or zero), the flying capacitor still needs to be maintained at the output voltage for smooth startup when the converter turns on again. This can be done by only entering modes D and C, which does not transfer any more energy to the output, but maintains the flying capacitor voltage at the output.

This mode is important as there is not continuous conduction mode associated with the control method presented in this thesis. If charge was always transferred to the output, no matter how small, it would cause the output to rise out of regulation if there was no load.

ResSC, Self Balancing, $V_{out} << V_{in}/2$

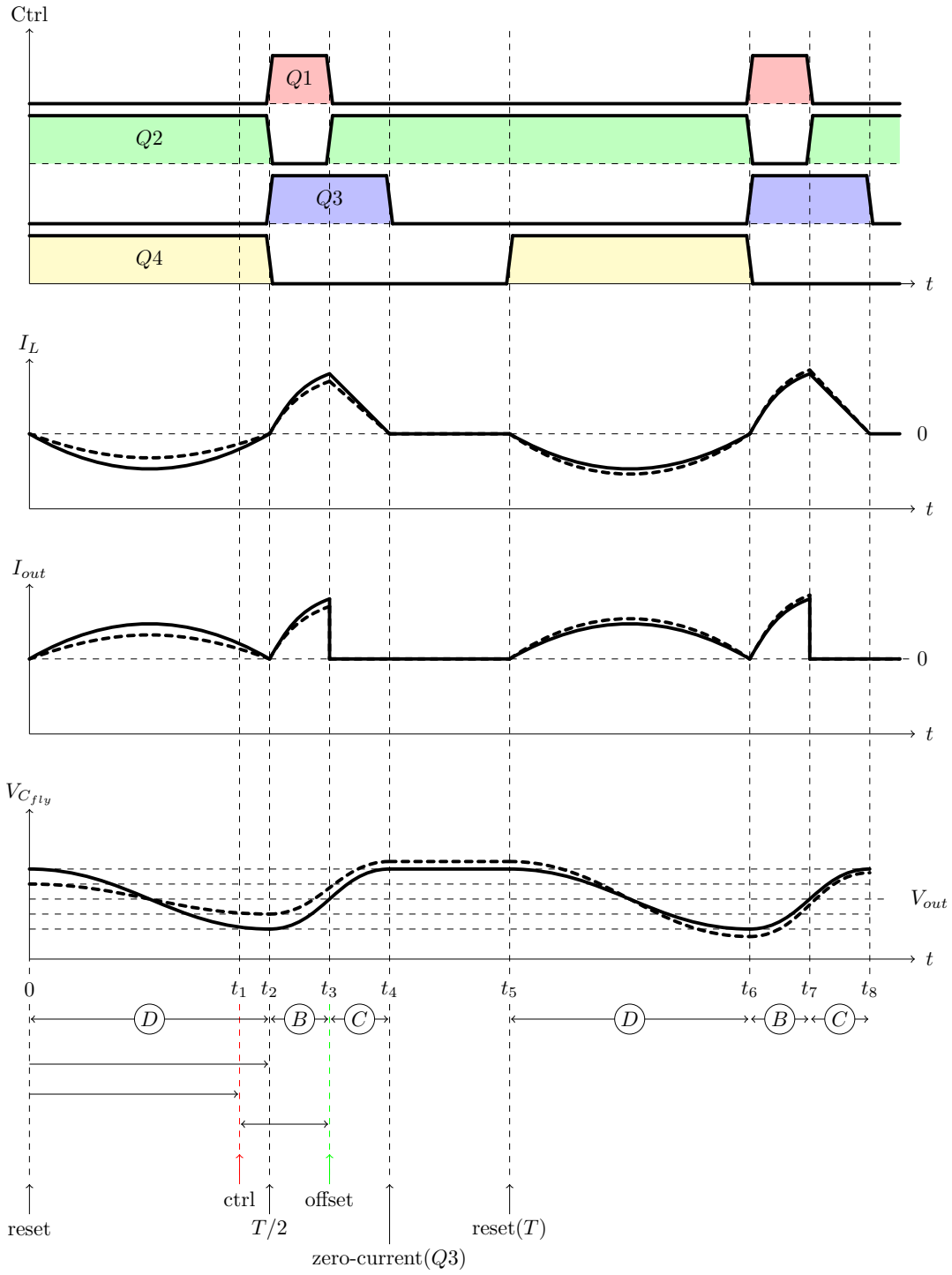


Figure 3-12: A ResSC with $V_{out} < V_{in}/2$. Dashed lines are plotted for I_L, I_{out}, and V_{Cfly} which show the result if $V_{C_{fly}}$ was a bit lower than its nominal value of $V_{in}/2$ at the start. Self-correction of the flying capacitor voltage is seen.

ResSC, Self Balancing, $V_{out} >> V_{in}/2$

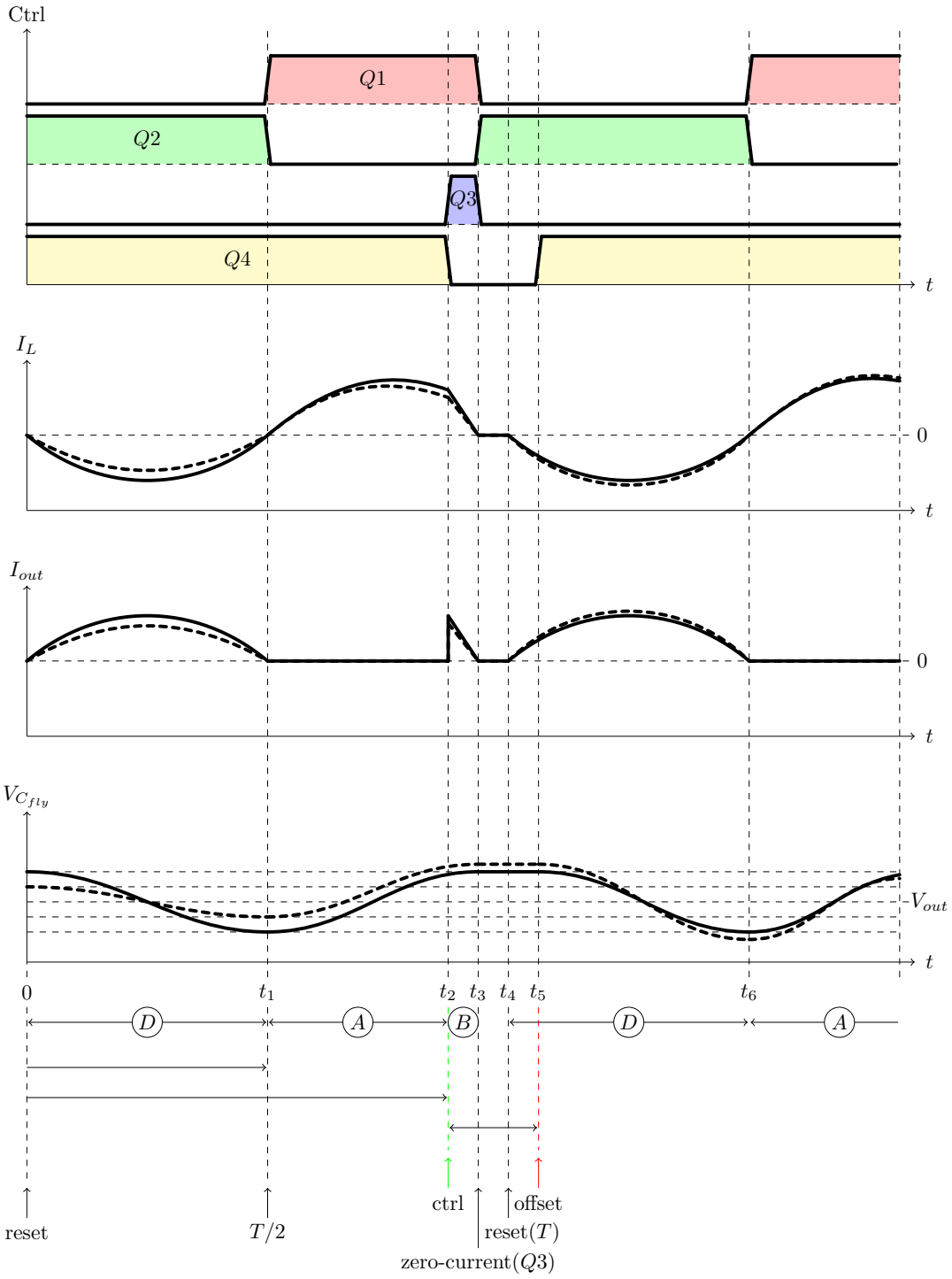


Figure 3-13: A ResSC with $V_{out} > V_{in}/2$. Dashed lines are plotted for I_L, I_{out}, and V_{Cfly} which show the result if $V_{C_{fly}}$ was a bit lower than its nominal value of $V_{in}/2$ at the start. Self-correction of the flying capacitor voltage is seen.

3.2.5 Passive Component Limitations (MLCC, etc)

The table below was populated using Murata's Simsurfing tool to find the highest self-resonant frequency of a given capacitance for standard MLCCs. The maximum output current is calculated using the maximum flying capacitor voltage swing allowed with the switching frequency and capacitance.

$I_{out_{max}} = F_{sw} * C_{fly} * 2 * V_{bodyDiode}$

The table below is populated for $V_{bodyDiode} = 0.7V$.

Cap	Freq Self-Resonant	ESRmin	Max Output Current
100uF	0.55MHz	1.602mOhm	77.0A
47uF	0.8MHz	1.768mOhm	52.6A
10uF	2.5MHz	1.3mOhm	35.0A
4.7uF	4.0MHz	1.687mOhm	26.3A
1uF	10.0MHz	3.622mOhm	14.0A
0.47uF	18.0MHz	5.255mOhm	11.8A
0.1uF	23MHz	18.357mOhm	3.2A
0.047uF	45MHz	26.289mOhm	2.1A
0.010uF	100MHz	57.98mOhm	1.4A

3.2.6 Output Ripple

This topology has discontinuous output currents, which can cause output capacitor parasitic inductance to have a large effect on the output ripple. An additional output filter may be required to reduce ripple to desired levels.

3.3 Ćuk-Buck2

3.3.1 Fifth switch for full range output

A fifth switch was added to the Ćuk-Buck2 [4][9] in order to allow full output range. This lets the input side of the main inductor be connected to V_{in} through Q5 and then to $V_{in} - V_{out}$ or 0V like before. This allows this modified Ćuk-Buck2 to operate with a conversion ratio anywhere from 1:1 to 1:0.

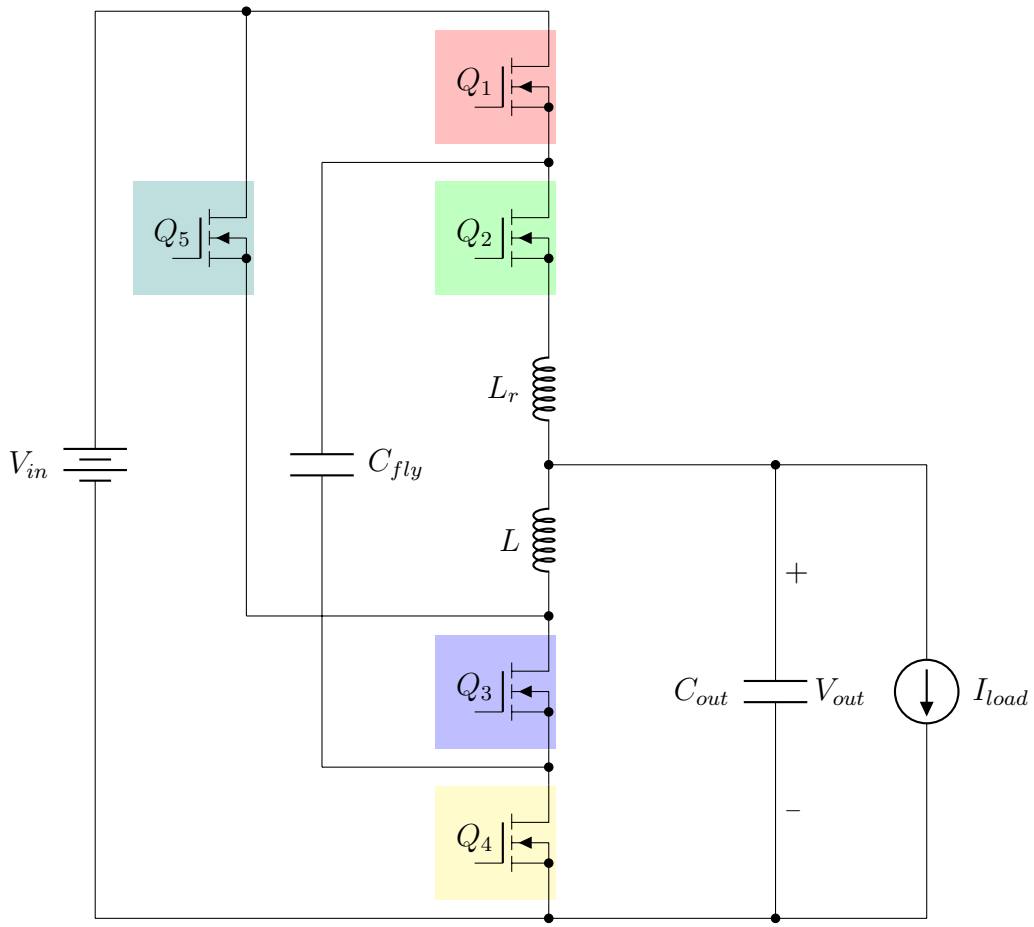


Figure 3-14: A Ćuk-Buck2 with added $Q5$ to enable full-range output conversion ratios of 0 to 1.

It is possible to operate a Ćuk-Buck2 without L_r, but may be more inefficient due to charge sharing loss between the flying capacitor and the output capacitor.

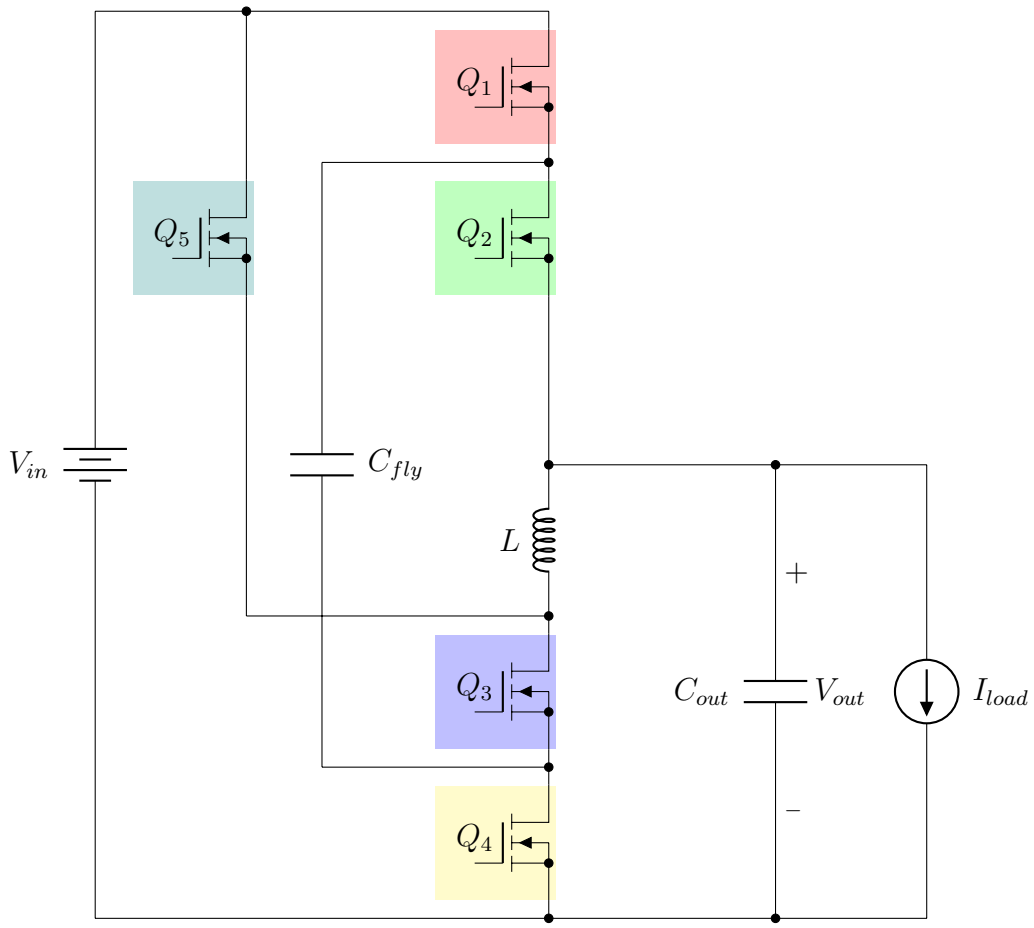


Figure 3-15: A Ćuk-Buck2 with added $Q5$ to enable full-range output conversion ratios of 0 to 1. L_r is set to $0H$ and is shown as a wire.

Ćuk-Buck2 with Lres, $V_{out} > V_{in}/2$

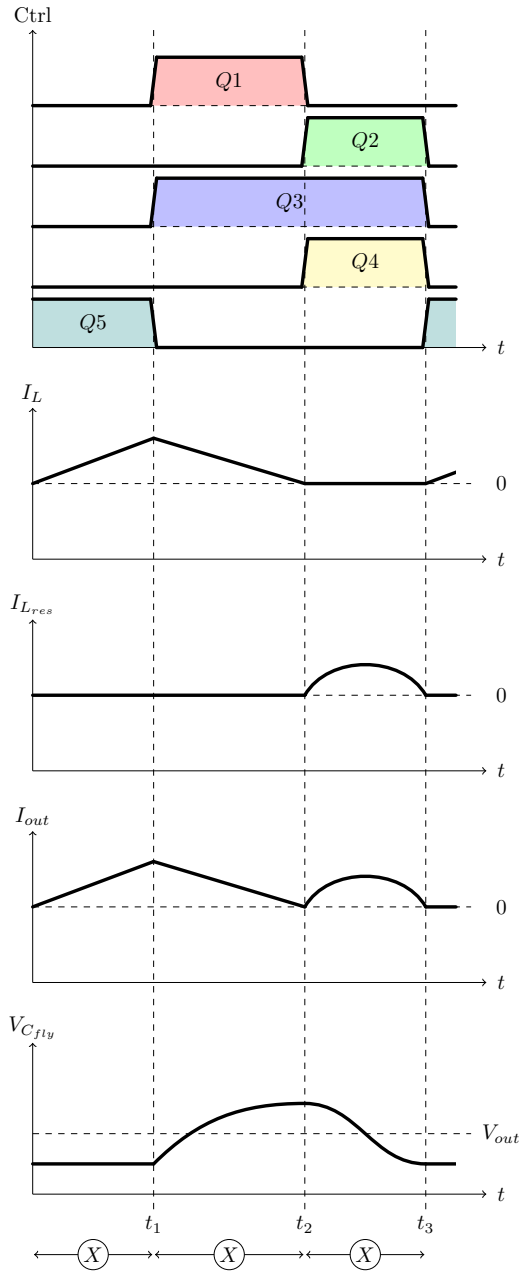


Figure 3-16: Timing diagram for Ćuk-Buck2 with second resonant inductance L_r at low output voltage.

Ćuk-Buck2 without Lres, $V_{out} > V_{in}/2$

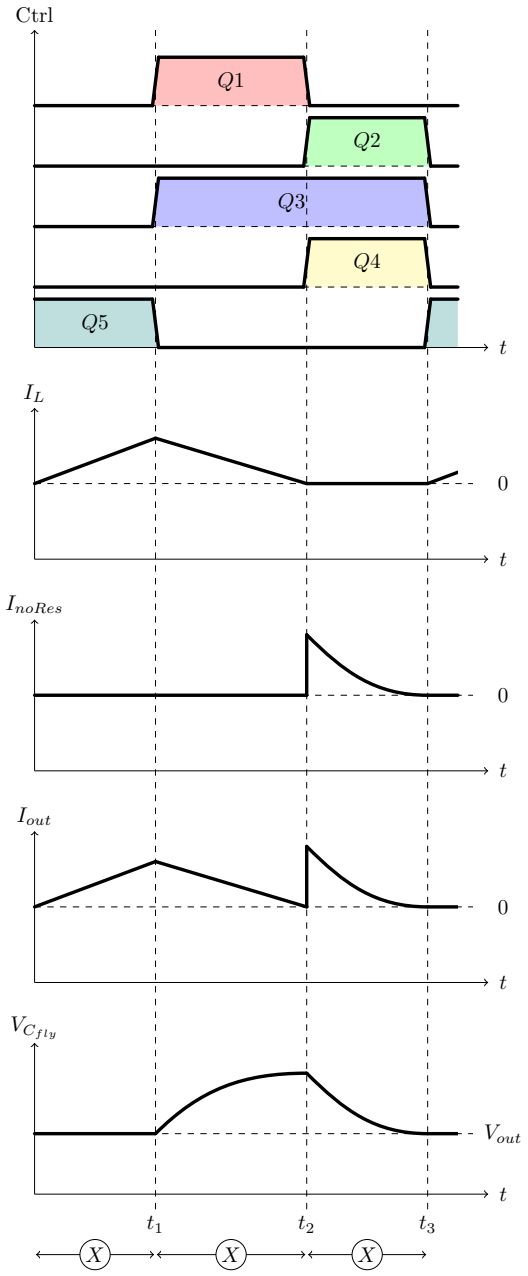


Figure 3-17: Timing diagram for Ćuk-Buck2 without second resonant inductance L_r at low output voltage.

3.3.2 Fixed Frequency Operation

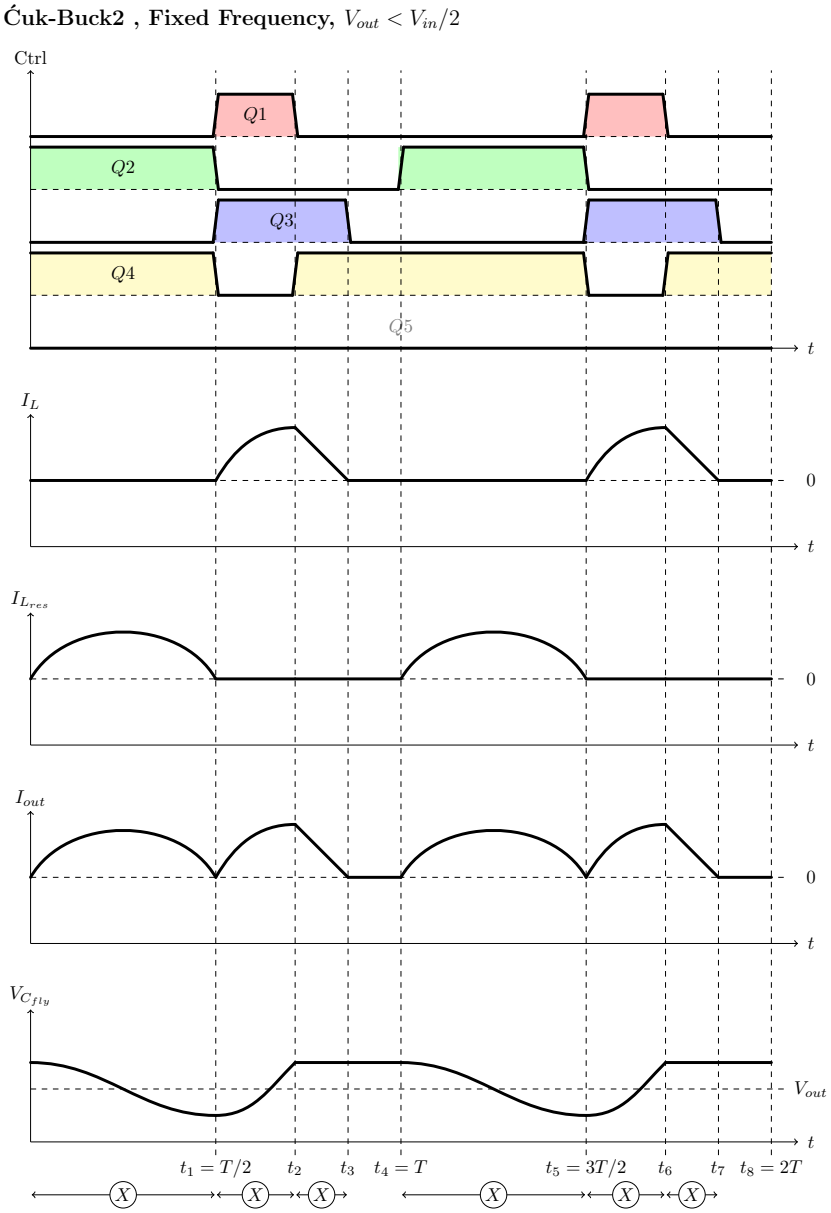


Figure 3-18: A Ćuk-Buck2 operating with $V_{out} < V_{in}/2$ at a fixed frequency with period T. This is the same operation as 2-12 with a blank period inserted from t_3 to t_4 where only $Q4$ is on. The flying capacitor must not be below $V_{out} - 0.7$ during this period, or unintended currents will reduce efficiency.

Ćuk-Buck2 , Fixed Frequency, $V_{out} > V_{in}/2$

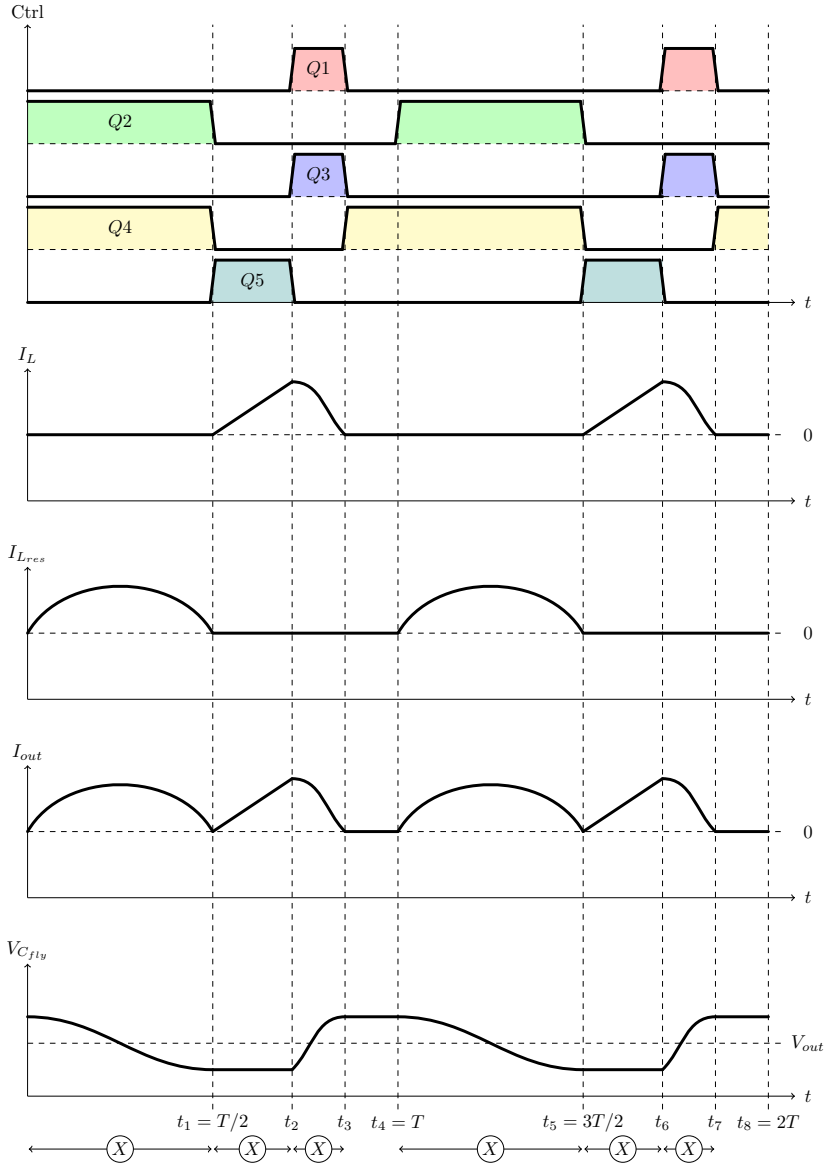


Figure 3-19: A Ćuk-Buck2 operating with $V_{out} > V_{in}/2$ at a fixed frequency with period T. This is the same operation as Figure 3-16 with a blank period inserted from t_3 to t_4 where only $Q4$ is on. The flying capacitor must not be below $V_{out} - 0.7$ during this period, or unintended currents will reduce efficiency.

3.3.3 Smooth Transitioning Full-Range Output

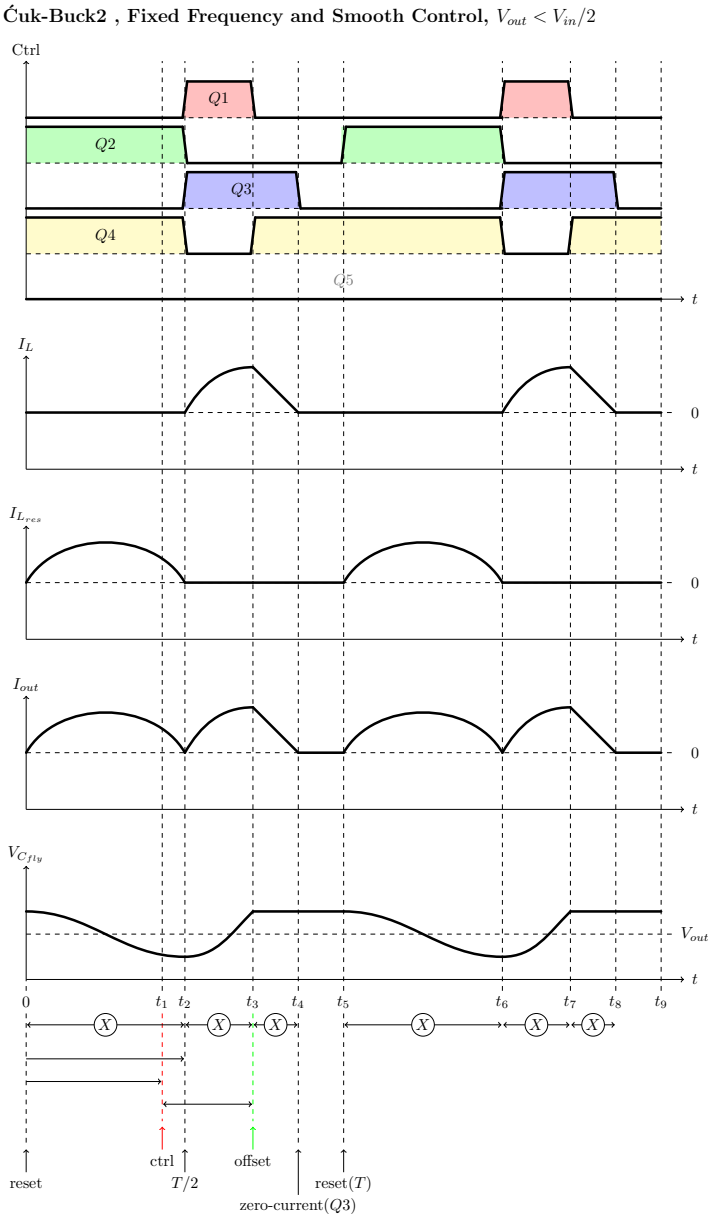


Figure 3-20: A Ćuk-Buck2 operating with $V_{out} < V_{in}/2$ at a fixed frequency with period T. This is the same operation as Figure 3-18 with control timing events listed below. A *ctrl* timing is set by feedback and is measured from *reset*. An *offset* timing is offset from *ctrl* by a fixed amount. The timing *ctrl* has no effect when it comes before $T/2$, so it is pictured in red.

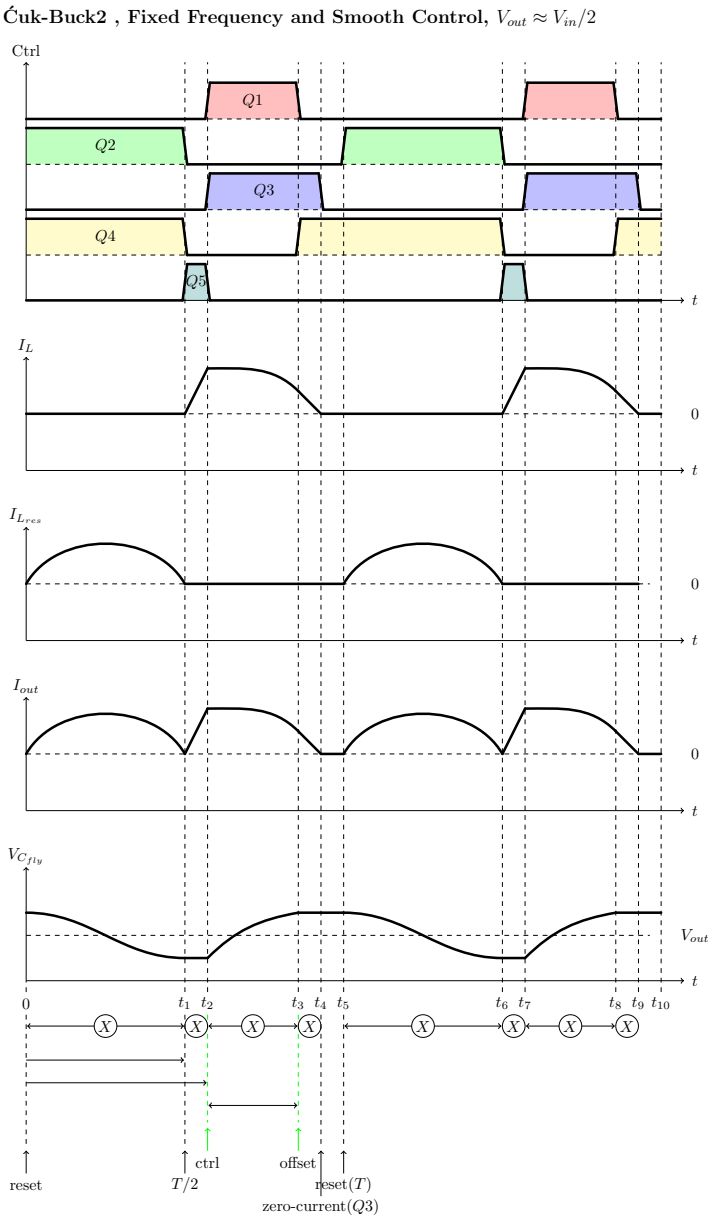


Figure 3-21: A Ćuk-Buck2 operating with $V_{out} \approx V_{in}/2$ at a fixed frequency with period T. This is a joined operation with elements from Figure 3-18 Figure 3-19 with control timing events listed below. A *ctrl* timing is set by feedback and is measured from *reset*. An *offset* timing is offset from *ctrl* by a fixed amount.

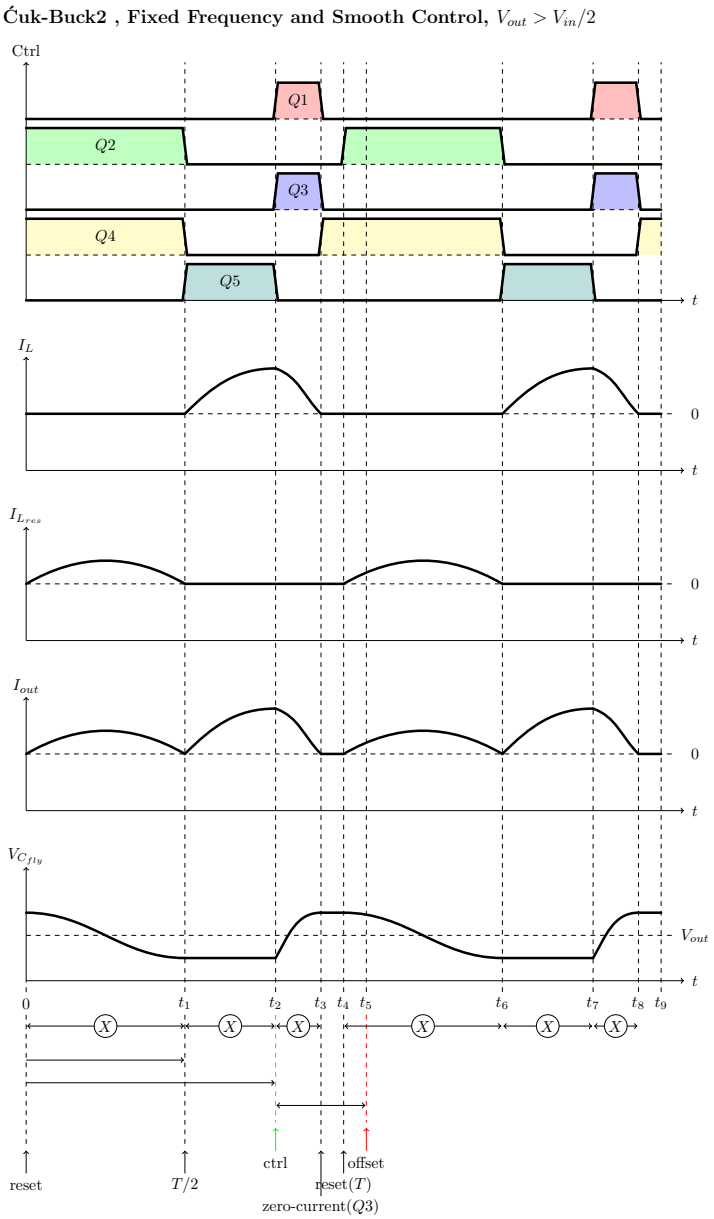


Figure 3-22: A Ćuk-Buck2 operating with $V_{out} > V_{in}/2$ at a fixed frequency with period T. This is the same operation as Figure 3-19 with control timing events listed below. A *ctrl* timing is set by feedback and is measured from *reset*. An *offset* timing is offset from *ctrl* by a fixed amount. The timing *offset* has no effect when it comes after *reset(T)* or *zero-current(Q3)*, so it is pictured in red.

3.3.4 Capacitor Balancing

This is largely identical to the ResSC balancing and can be solved in the same way of filtering at half the switching frequency, by sampling and holding control values for two periods.

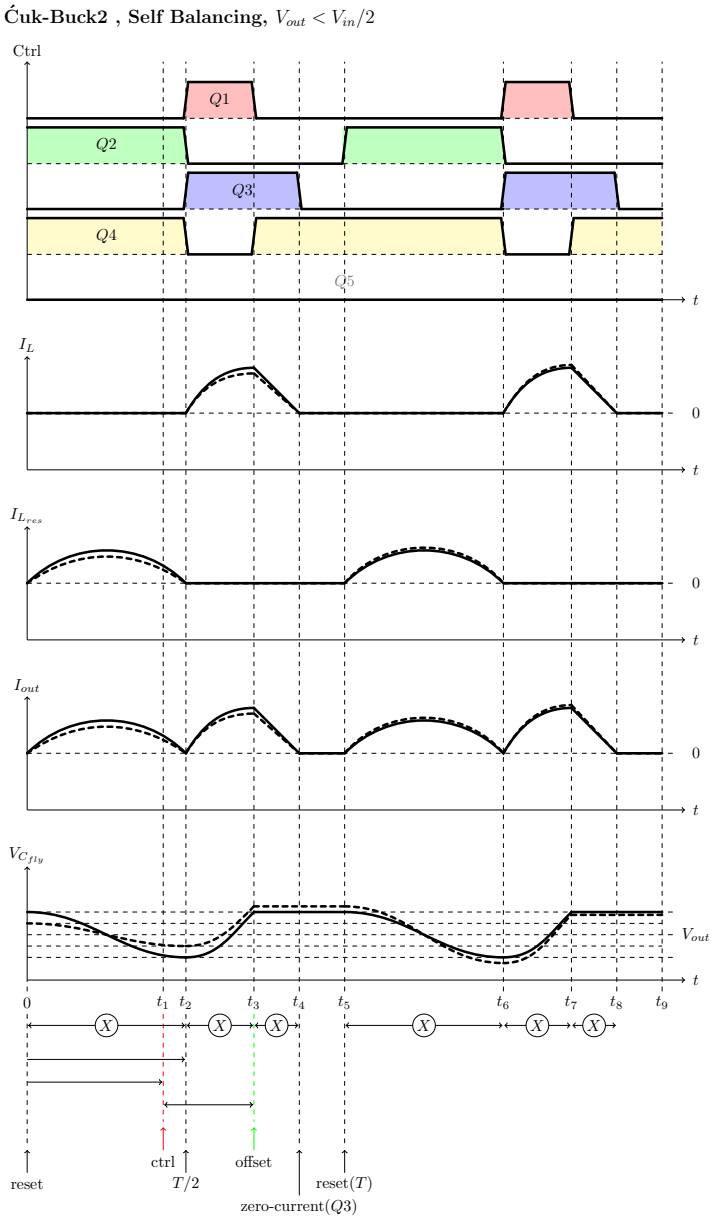


Figure 3-23: A Ćuk-Buck2 with $V_{out} < V_{in}/2$. Dashed lines are plotted for I_L, $I_{L_{res}}$, I_{out}, and V_{Cfly} which show the result if $V_{C_{fly}}$ was a bit lower than its nominal value of $V_{in}/2$ at the start. Self-correction of the flying capacitor voltage is seen.

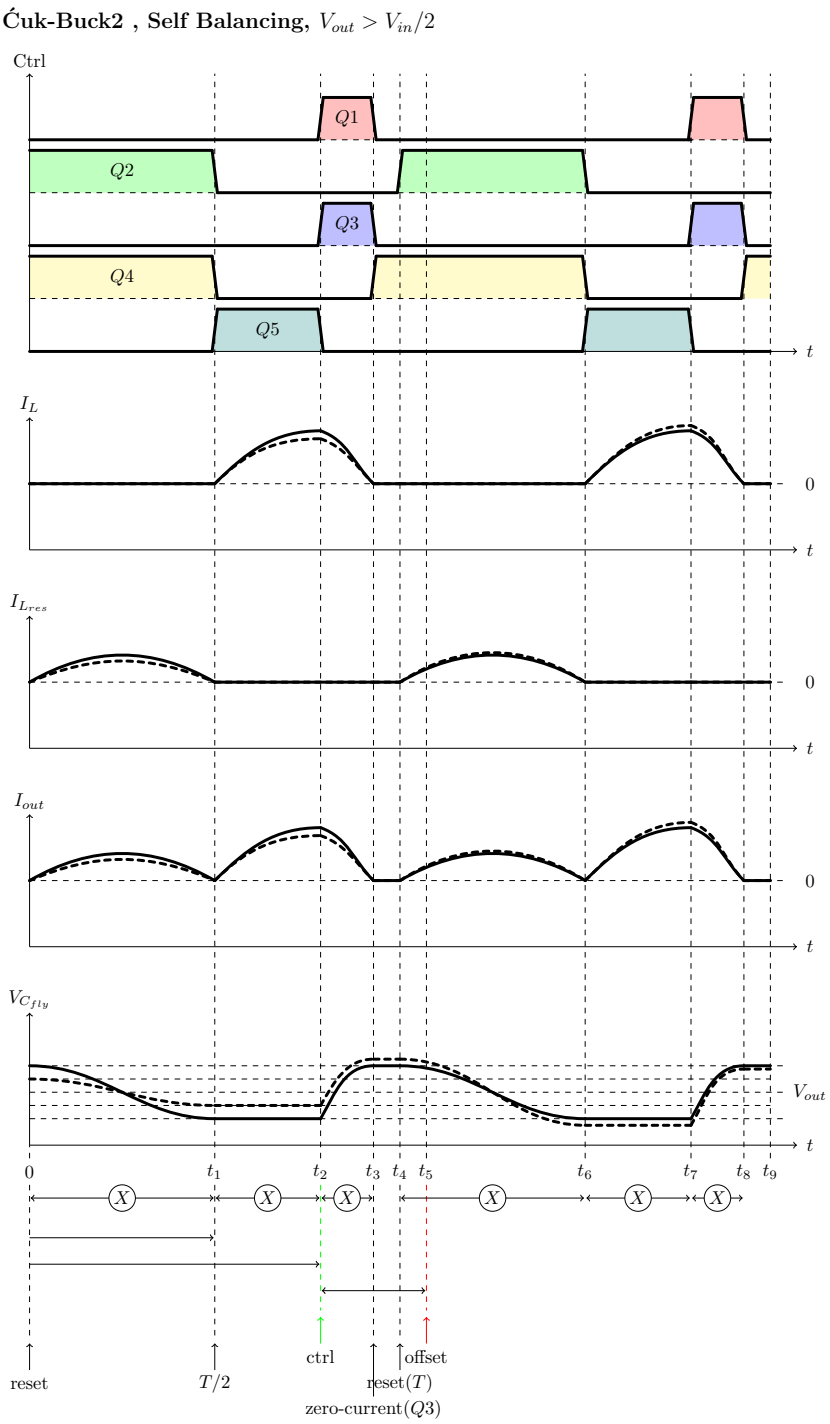


Figure 3-24: A Ćuk-Buck2 with $V_{out} > V_{in}/2$. Dashed lines are plotted for I_L, $I_{L_{res}}$, I_{out}, and V_{Cfly} which show the result if $V_{C_{fly}}$ was a bit lower than its nominal value of $V_{in}/2$ at the start. Self-correction of the flying capacitor voltage is seen.

Chapter 4

Evaluations and Comparisons

4.1 Efficiency

Since it was chosen to focus on converters that could convert 5V down to around 1V needed for modern processors, an application mirroring the capabilities of (some linear part that would be good to compare to). So 5V to 1V 5A output is where efficiency was optimized.

4.2 Additional Complexities

Each of these topolgies needs an additional capacitor that the buck converter does not, and has multiple flying switches. The available capacitors can be a limiting factor for the maximum output current. The multiple flying switches require level conversion for digital control signals and multiple power rails to drive them.

Multiple zero current sensors are also required, some on flying switches, which may be complicated to implement in a monolithic IC.

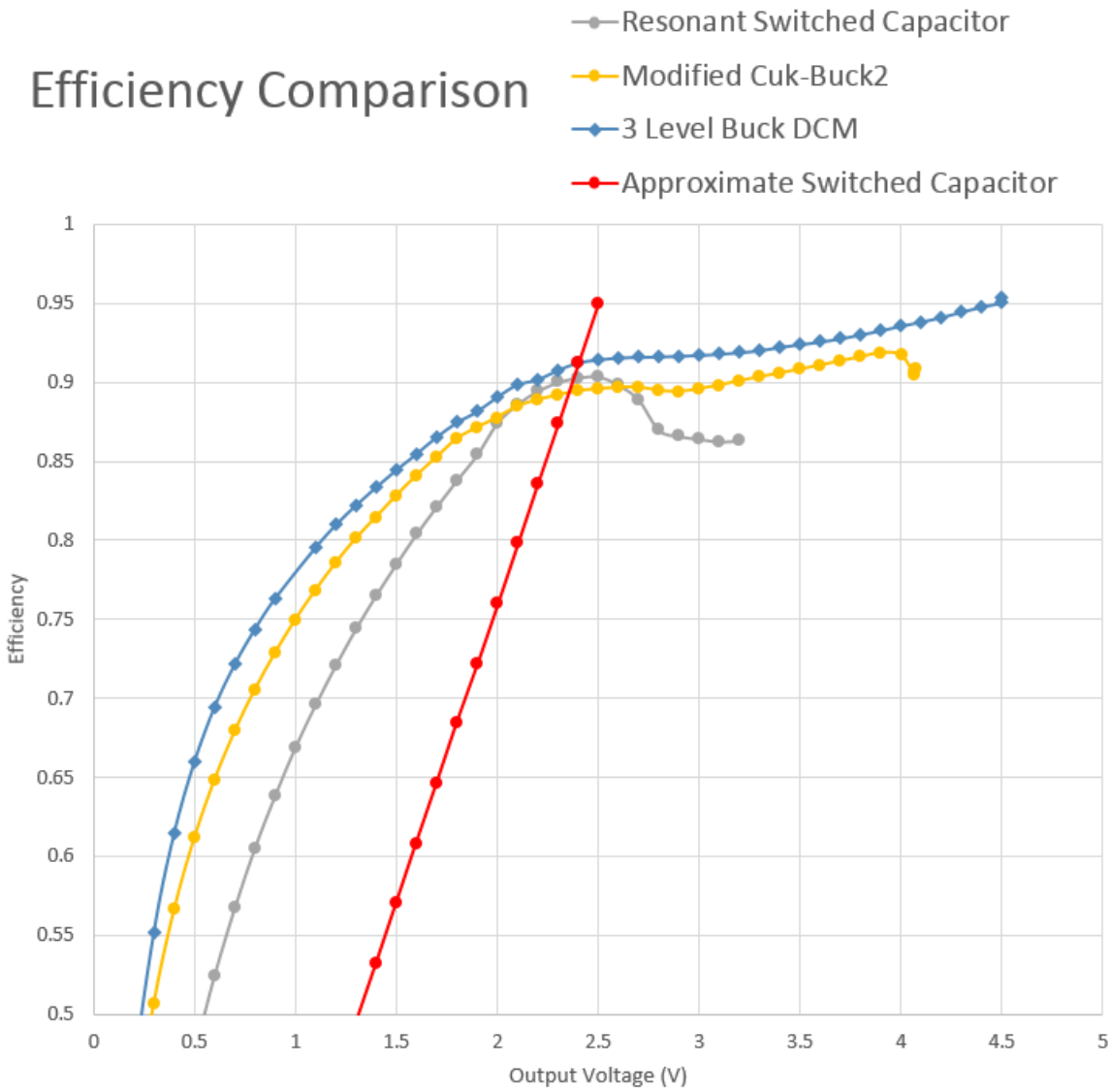


Figure 4-1: Simulated Efficiency of converters running at 2MHz with sub-10nH inductances in Linear Technology's 0.35 µm BCD process. An approximation for a 2-to-1 switched capacitor converter is also shown.

4.3 Output Ripple

Output ripple can be problematic on both the ResSC and Ćuk-Buck2 , since they both have a point of chopped output current. This with parasitic inductances can cause

a large spike in the output voltage. This can be filtered with an additional inductor and capacitor, but that reduces density and efficiency and increases the cost.

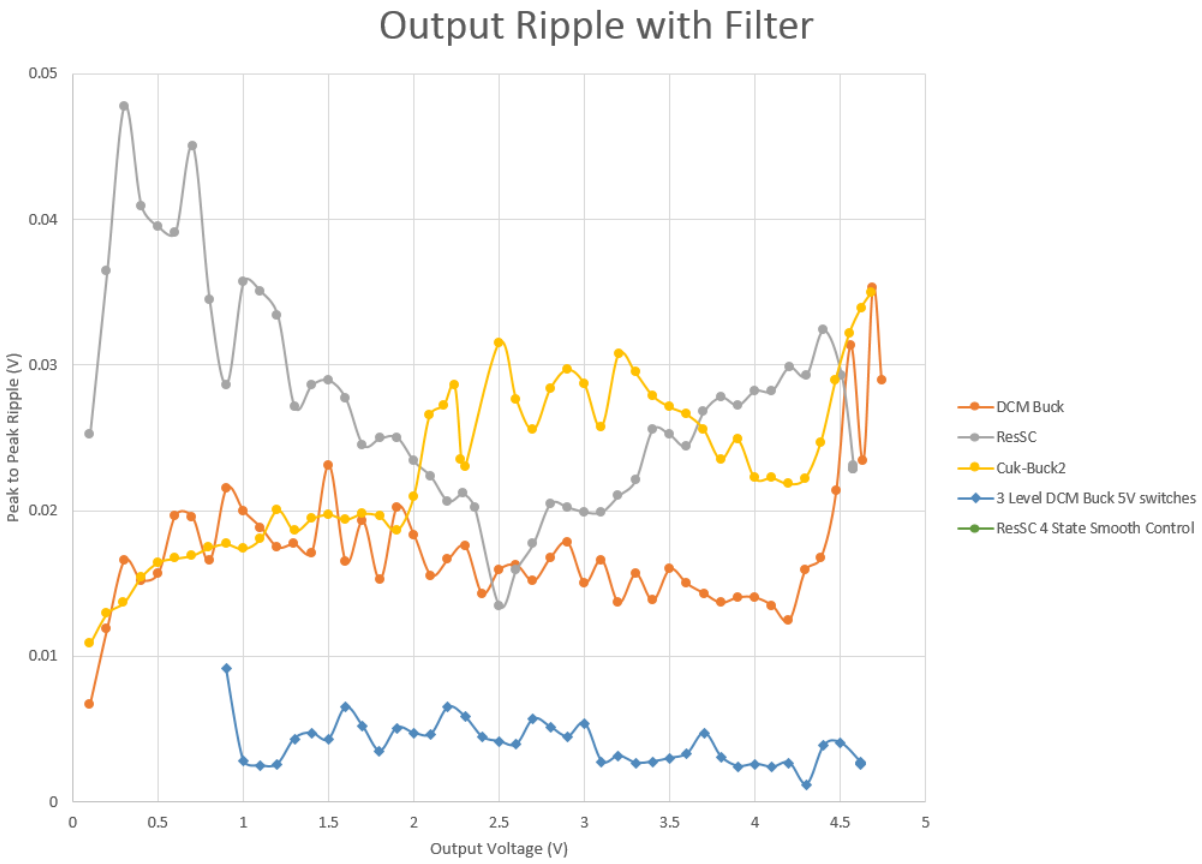


Figure 4-2: Output ripple for different converters simulated in LTSpice that were each hand optimized for efficiency. Each converter has a sub-10nH inductances and single digit MHz switching frequencies. All simulations had the same output capacitance and filtering, which can be seen in Figure 4-3.

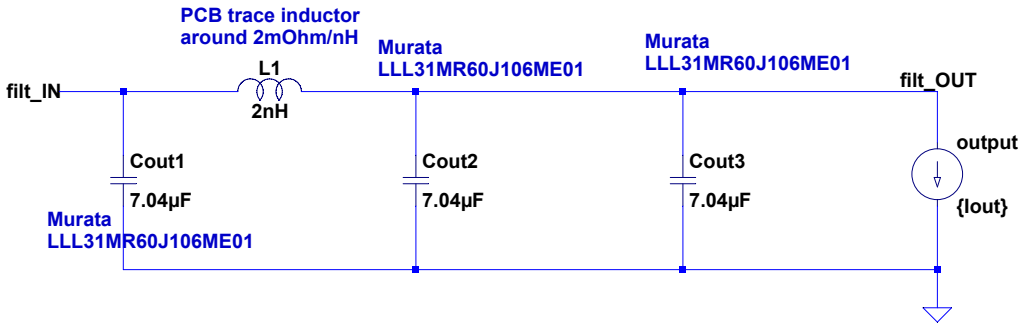


Figure 4-3: Output capacitor (Cout1) and output filter used for the simulations for Figure 4-2. Parasitic inductance and resistance were included for the capacitors and resistance for the inductor was included to best estimate a PCB trace inductor.

4.4 Transient Response

All three converters can have approximately equivalent transient responses if controlled in the manner described in this thesis and the feedback network is tuned correctly. Work was performed in the area, but will not be discussed in this thesis.

Chapter 5

Prototype

5.1 Purpose

The purpose of making a physical prototype in this case is solely to determine how and if un-simulated real world effects impact how the control schemes function. Some of these effects are frequency dependent wire resistance and voltage dependent capacitance of MLCCs. The prototypes were scaled down in frequency by approximately ten times from the simulations, which means that the inductances were increased by ten times and the capacitances were increased by ten times in order to provide equivalent operation to higher frequency. The switch resistances were constrained by other requirements, but ended up being approximately the same as the simulated switches.

5.2 Overview

A modular power PCB was designed and manufactured to test the new control methods on the three topologies. It is also able to work as a normal buck converter. Particular attention was paid to reducing parasitic inductance as low as possible, which made it not possible to incorporate the 5th switch for the full range Ćuk-Buck2 on the PCB. It would be able to be added in dead-bug style for testing.

A secondary control PCB was also designed and manufactured. It utilizes the con-

figurable analog and digital hardware of a PSoC 5LP chip in order to have flexible analog feedback with changeable digital logic that can operate without a microcontroller in the loop.

This means that all control loops are entirely performed in hardware for this prototype, despite the fact that it may appear to be controlled by a microcontroller, as the PSoC chip has many independent analog and digital circuits that can be connected to its pins like a mixed-signal FPGA.

5.3 Logic

5.3.1 Shared

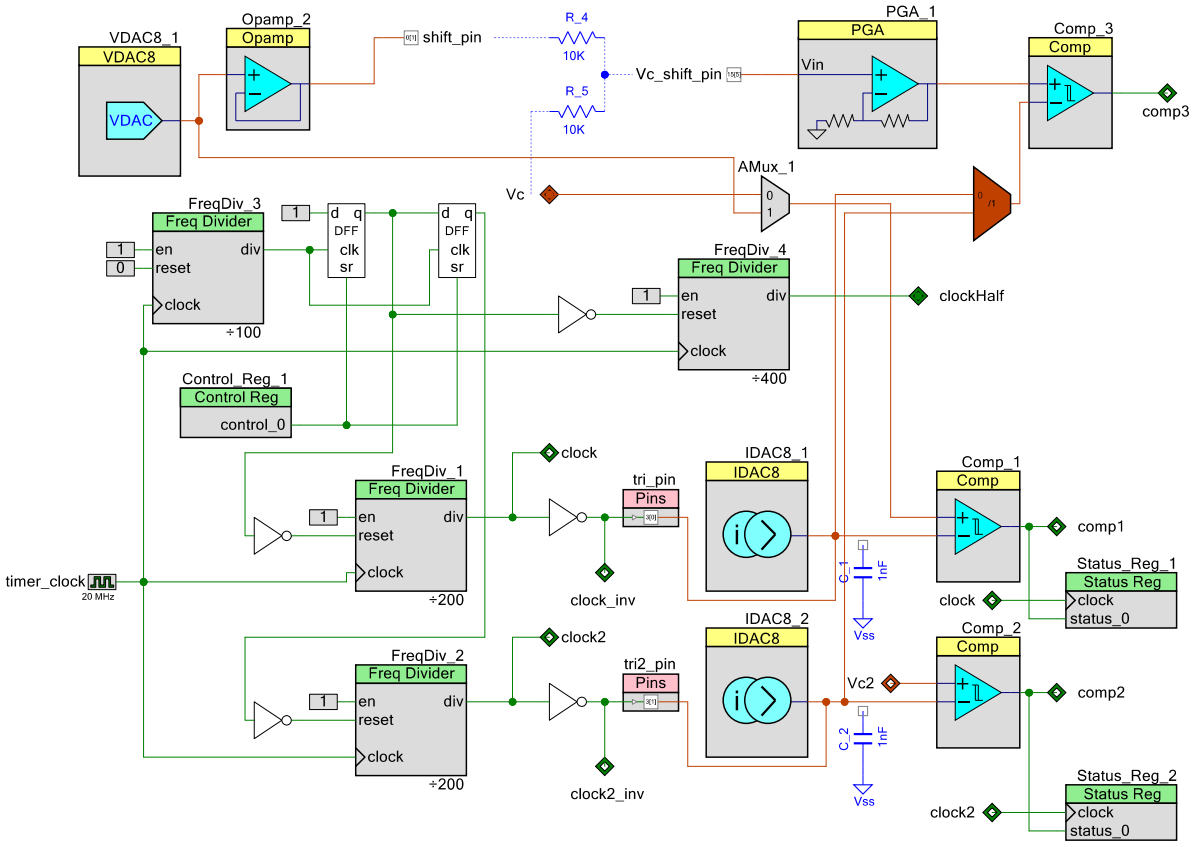


Figure 5-1: PWM circuits running on PSoC in hardware. Elements in blue represent hardware on the PCBs that is connected to the PSoC externally.

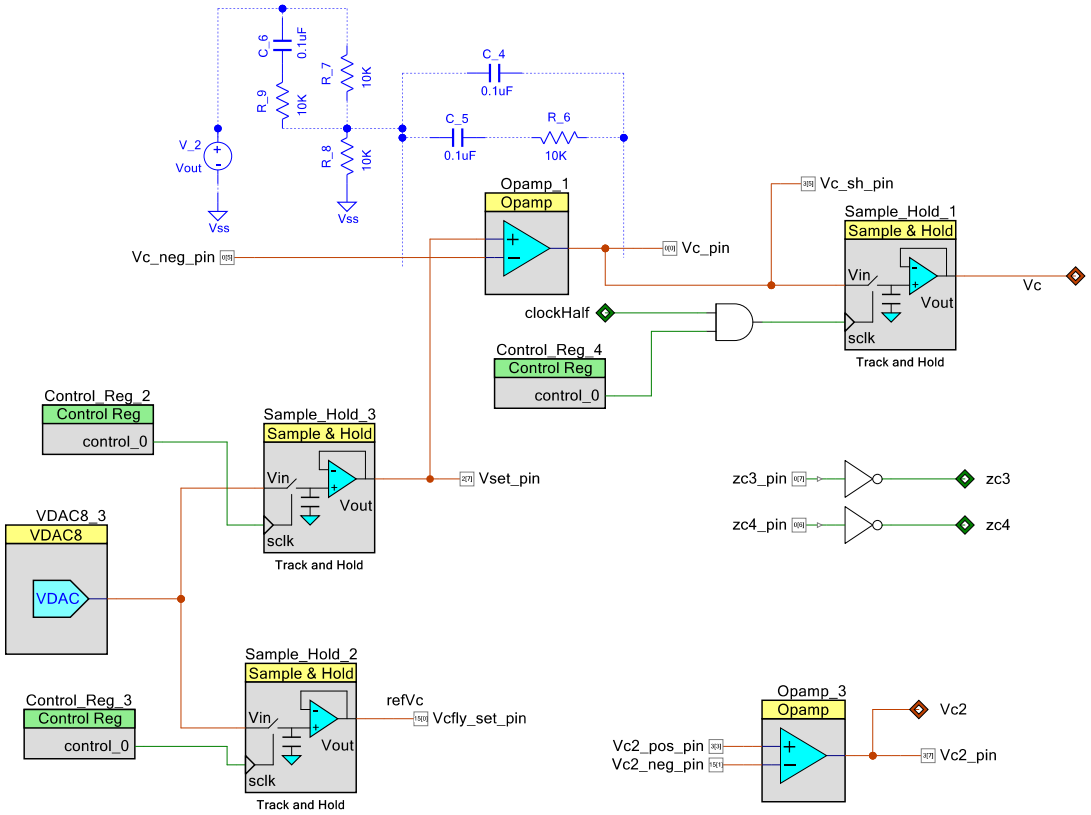


Figure 5-2: Feedback circuits running on PSoC in hardware. Elements in blue represent hardware on the PCBs that is connected to the PSoC externally.

5.3.2 Three Level Buck (DCM)

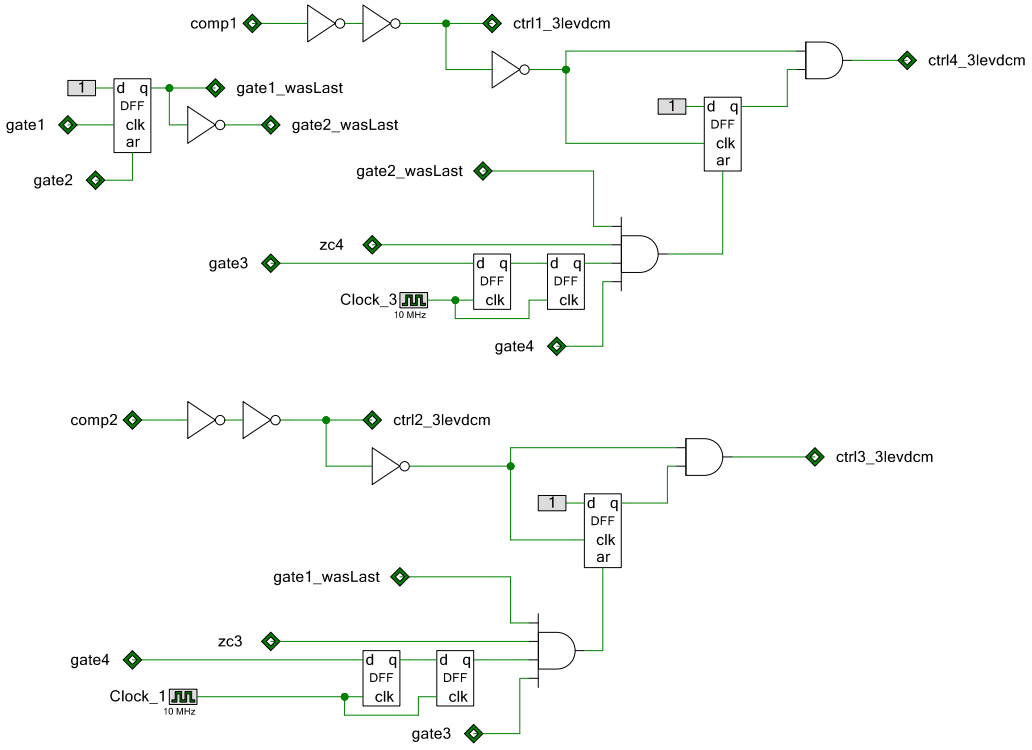


Figure 5-3: Control logic for the 3 Level Buck running on PSoC in hardware.

5.3.3 ResSC

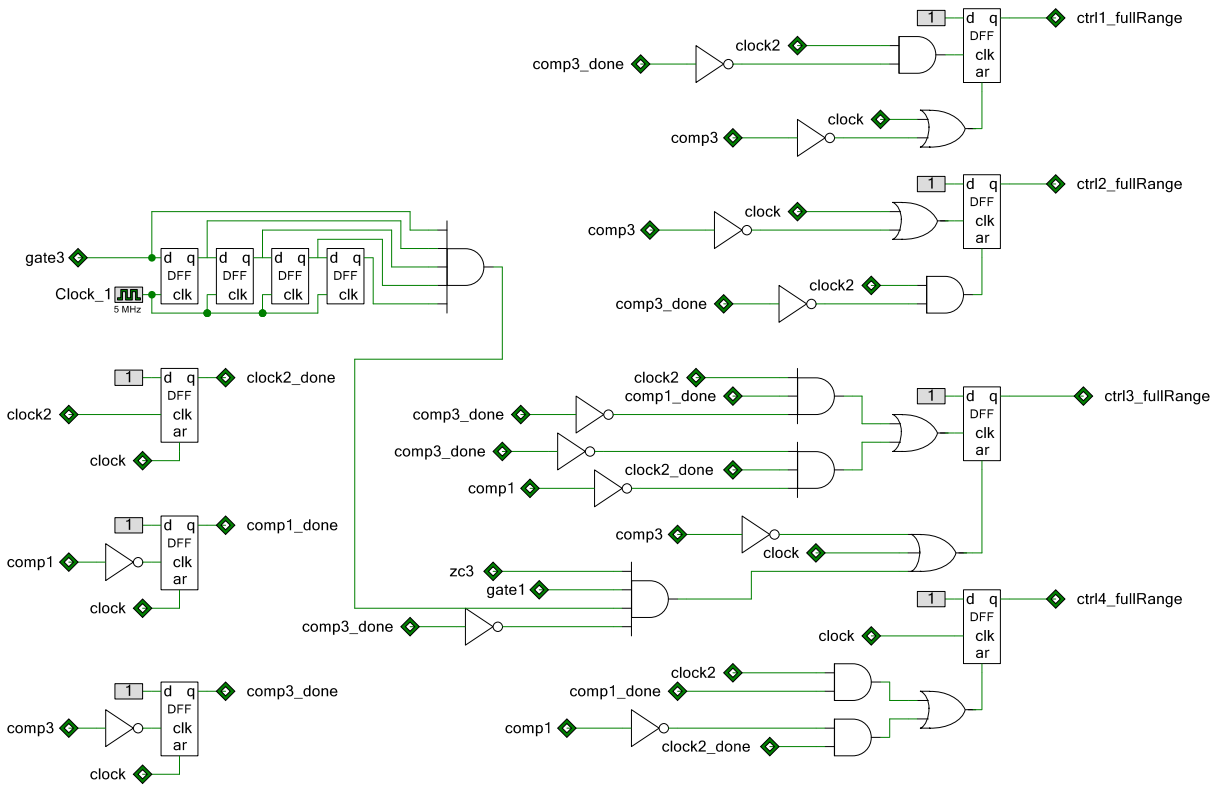


Figure 5-4: Control logic for the ResSC running on PSoC in hardware.

5.4 Schematic

5.4.1 Power

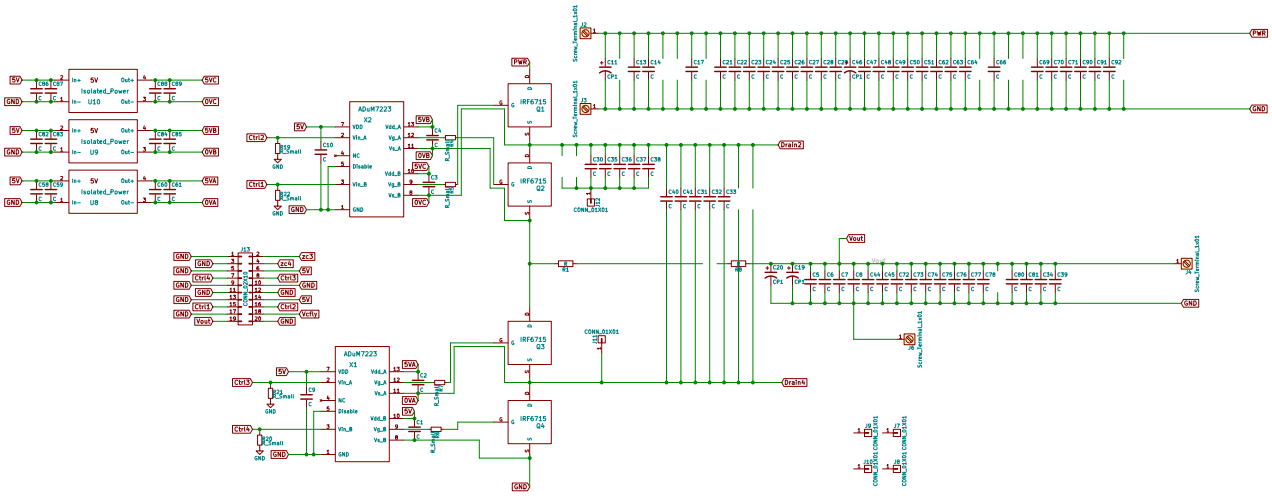


Figure 5-5: Power stage section of the Power PCB schematic from KiCAD.

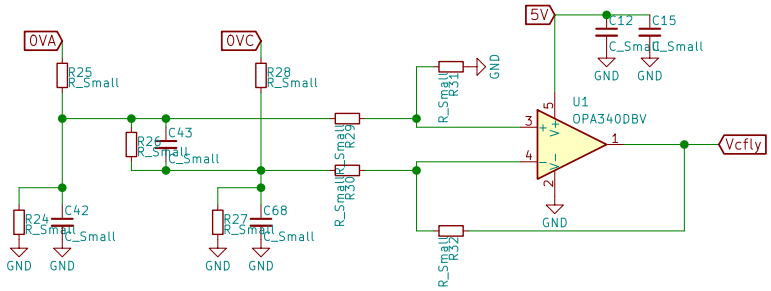


Figure 5-6: $V_{C_{fly}}$ sensing section of the Power PCB schematic from KiCAD.

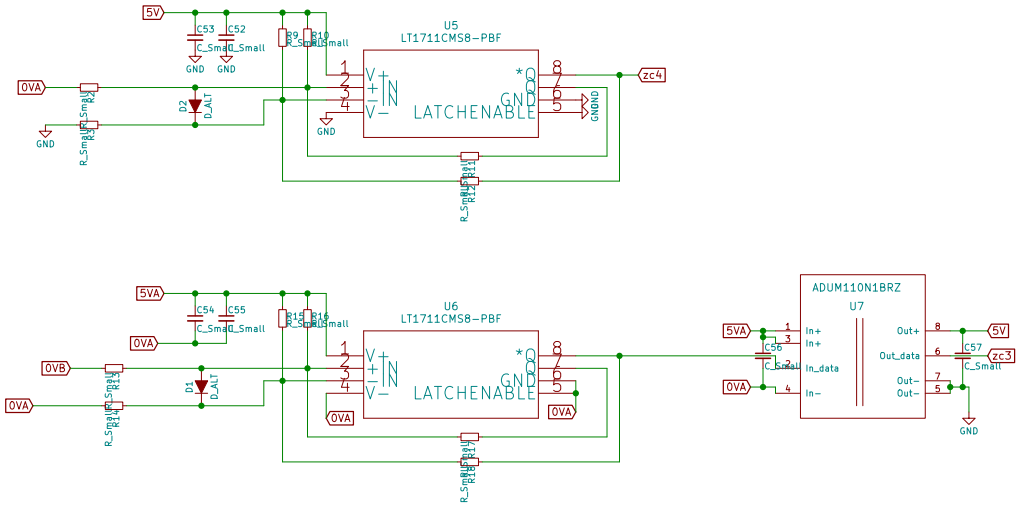


Figure 5-7: Zero current sensing section of the Power PCB schematic from KiCAD.

5.4.2 Control

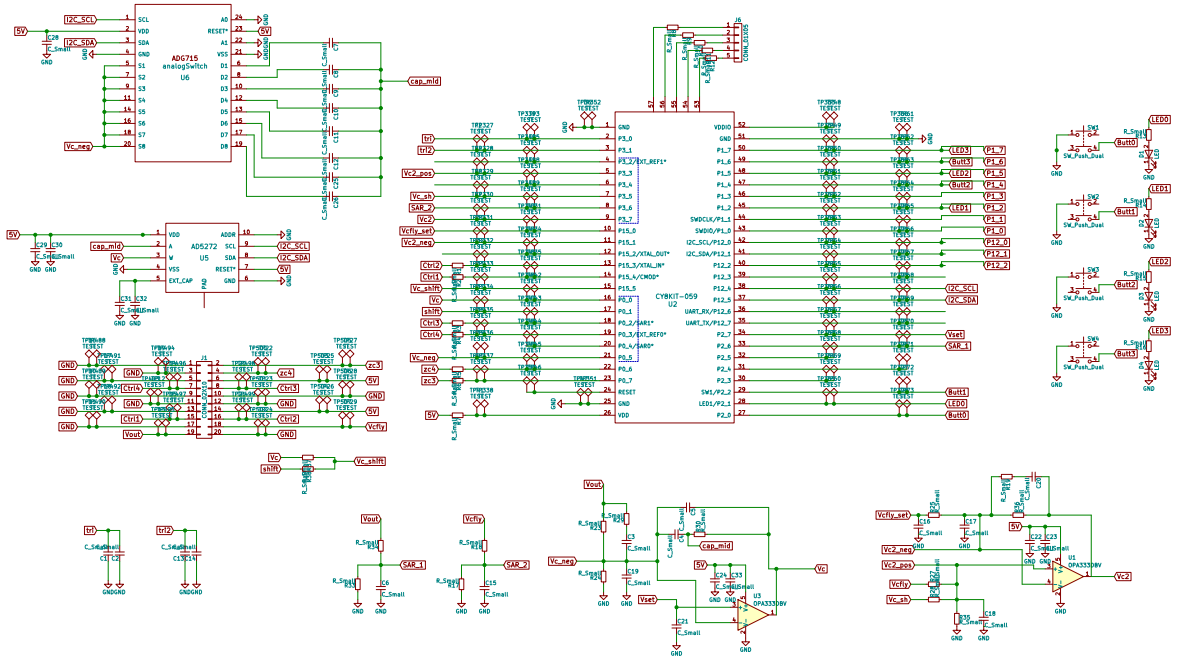


Figure 5-8: Schematic of the control PCB from KiCAD.

5.5 Bill-of-Materials (BOM)

5.5.1 Power

Designation	Part Number	Description
X1,X2	ADuM7223A	5V 4A Half-bridge Isolated Gate Driver
Q1,Q2,Q3,Q4	IRF6616	5mOhm 40V N-Fet
U8,U9,U10	ROE-0505S	5V 0.2A Isolated Power Supply
U5,U6	LT1711CMS8	4.5ns Rail-to-Rail Comparator
U7	ADuM110N1BRZ	Single Channel Digital Isolator
U1	AD8531ARTZ	3MHz Op Amp
C11,C46,C19,C20	16SVF1000M	16V 1000uF 12mOhm Al-Polymer Capacitor
Passives	varied	varied
Connectors	various 0.1" pitch	various 0.1" pitch

5.5.2 Control

Designation	Part Number	Description
U2	CY8KIT-059	Cypress PSoC-5LP Breakout
SW1,SW2,SW3,SW4	B3F-1000	Tactile Switch Through Hole
D1,D2,D3,D4	0603 Led	0603 Led
U5	AD5272	1024 20K digitally programmable resistor
U6	ADG715	Analog Octal SPST CMOS Switches
Passives	varied	varied
Connectors	various 0.1" pitch	various 0.1" pitch

5.6 Layout

5.6.1 Power

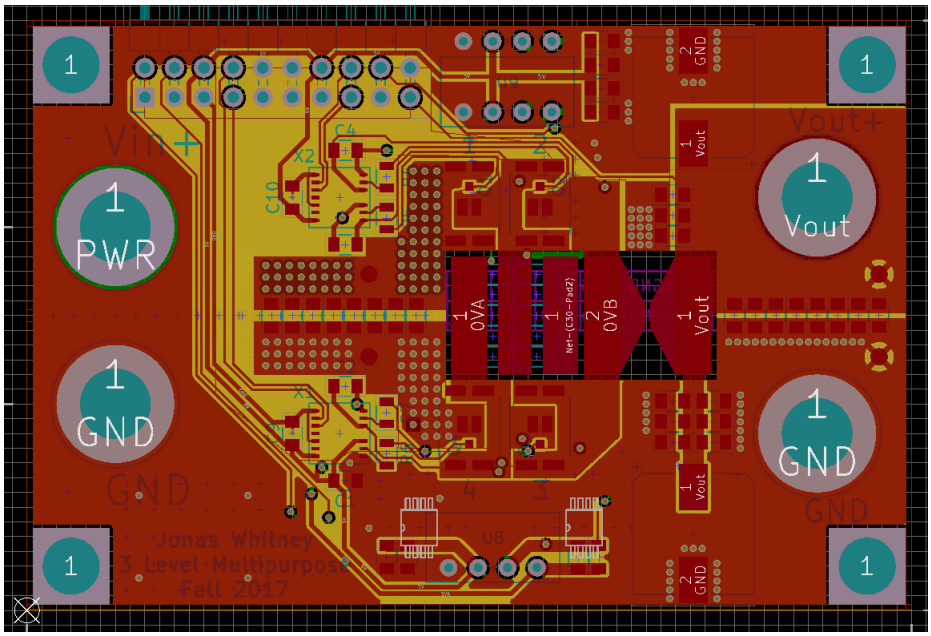


Figure 5-9: Top view of the layout for the power PCB in KiCAD.

5.6.2 Control

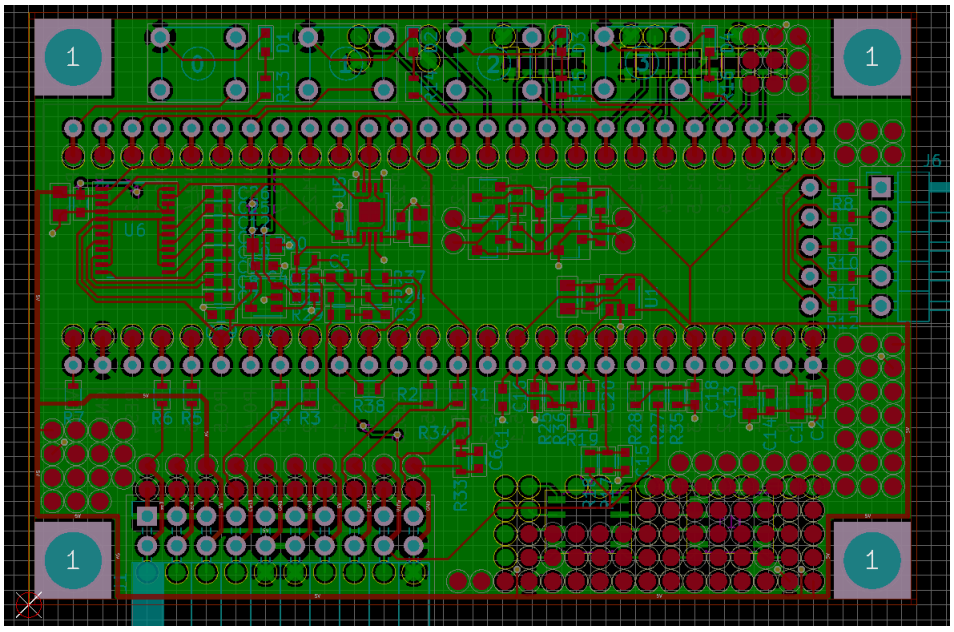


Figure 5-10: Top view of the layout for the control PCB in KiCAD.

5.7 Outcome

5.7.1 Three Level Buck (DCM)

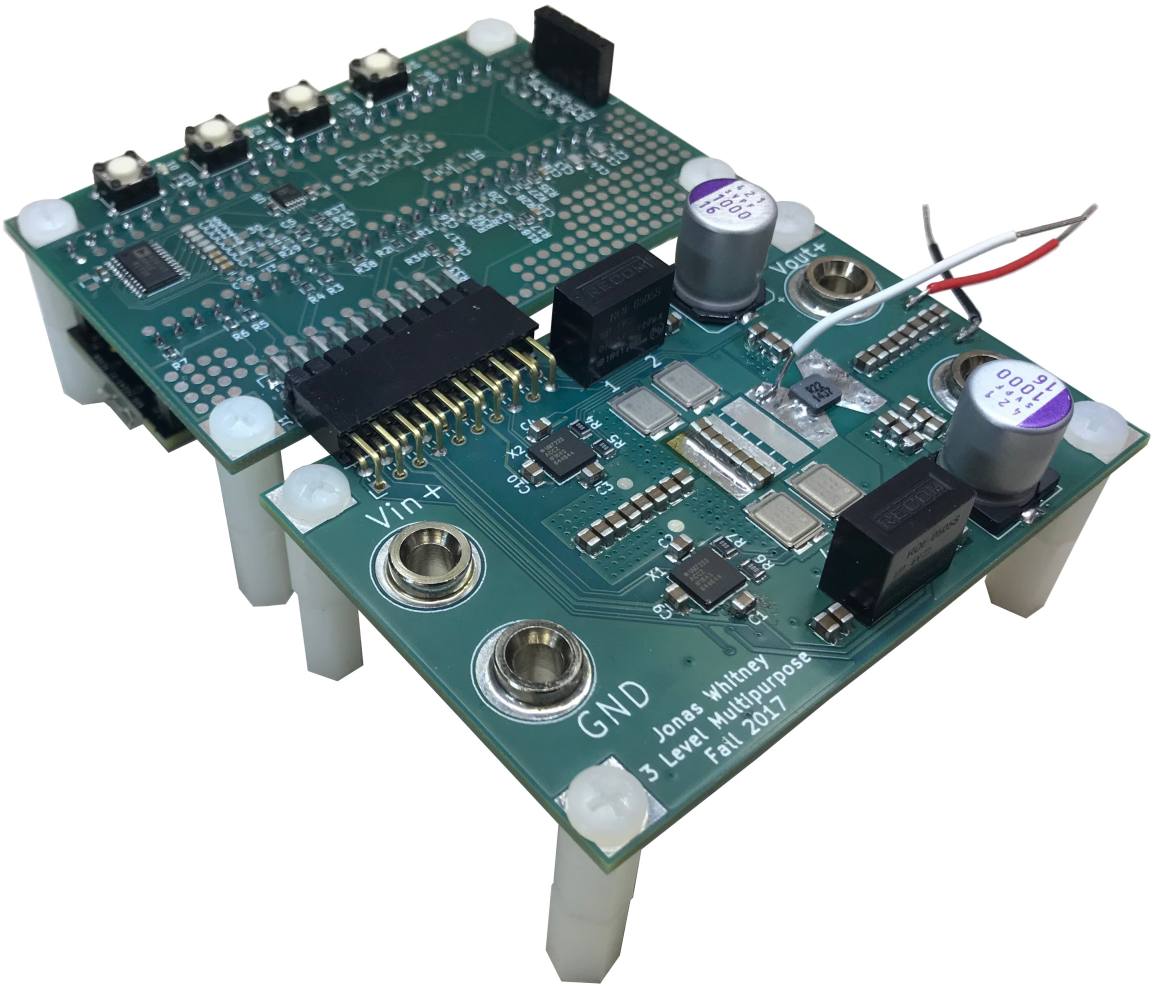


Figure 5-11: Completed Three Level Buck with 220nH inductor and 100uF flying capacitance.

Figure 5-12: Three Level Buck control signals when $V_{out} < V_{in}/2$. From top to bottom the signals are $Q1$, $Q2$, $Q3$, and $Q4$.

Figure 5-13: Three Level Buck control signals when $V_{out} \approx V_{in}/2$. From top to bottom the signals are $Q1$, $Q2$, $Q3$, and $Q4$.

Figure 5-14: Three Level Buck control signals when $V_{out} > V_{in}/2$. From top to bottom the signals are $Q1$, $Q2$, $Q3$, and $Q4$.

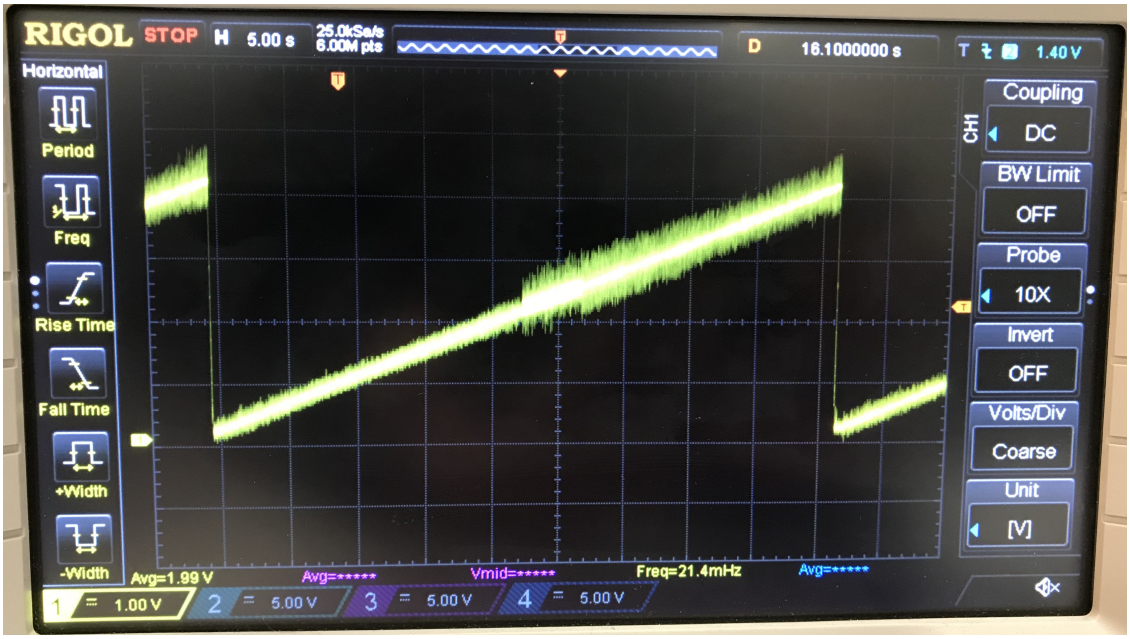


Figure 5-15: Output voltage ramp of Three Level Buck over 45s with a 5% offset in phase from 180deg. A 4Ohm load is connected with 5V input.

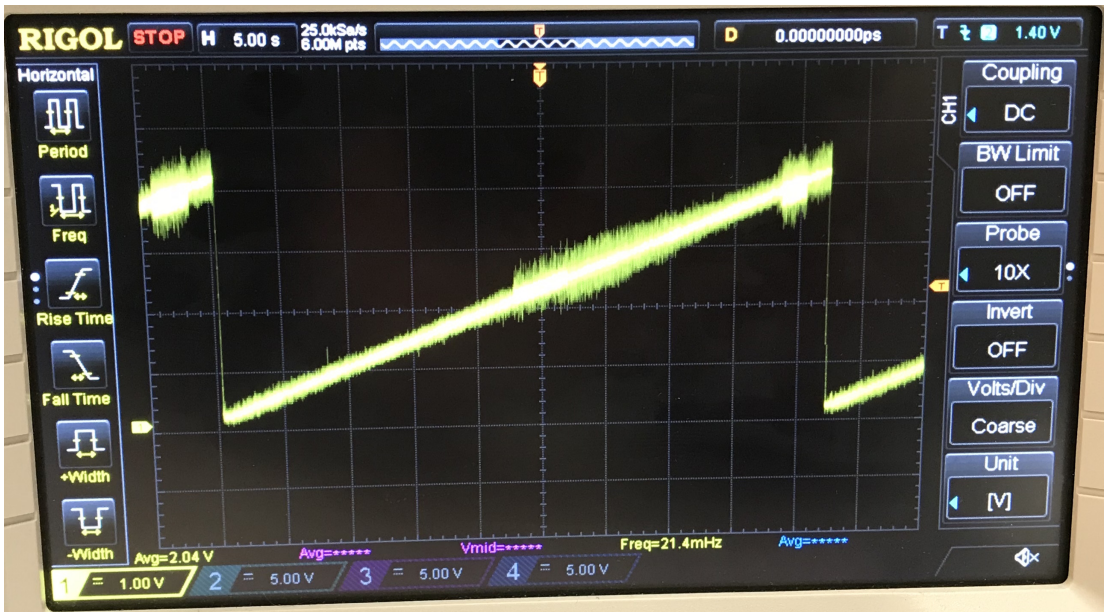


Figure 5-16: Output voltage ramp of Three Level Buck over 45s with no offset in phase from 180deg.

Figure 5-17: Output ripple for the Three Level Buck at 1V 5A output.

5.7.2 ResSC

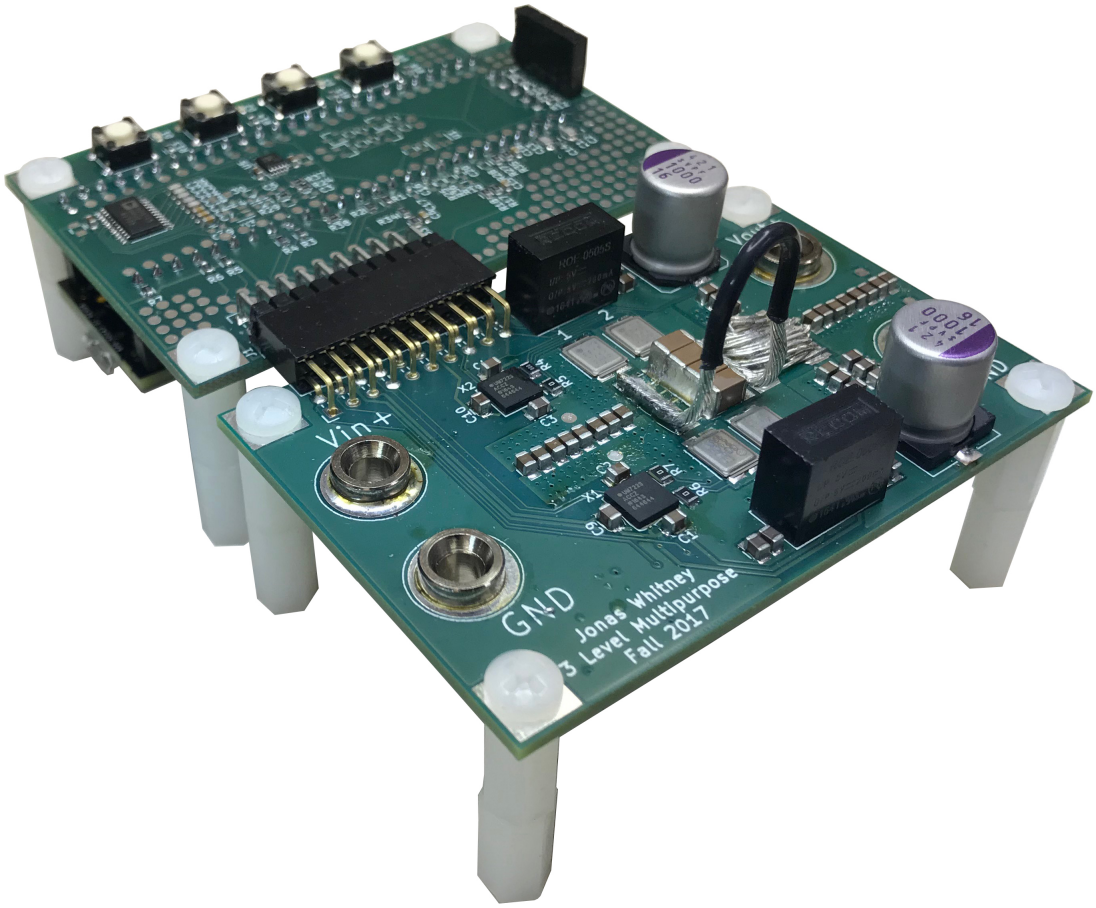


Figure 5-18: Completed ResSC with 47nH inductor and 82uF flying capacitance.

Figure 5-19: ResSC control signals when $V_{out} < V_{in}/2$. From top to bottom the signals are $Q1$, $Q2$, $Q3$, and $Q4$.

Figure 5-20: ResSC control signals when $V_{out} \approx V_{in}/2$. From top to bottom the signals are $Q1$, $Q2$, $Q3$, and $Q4$.

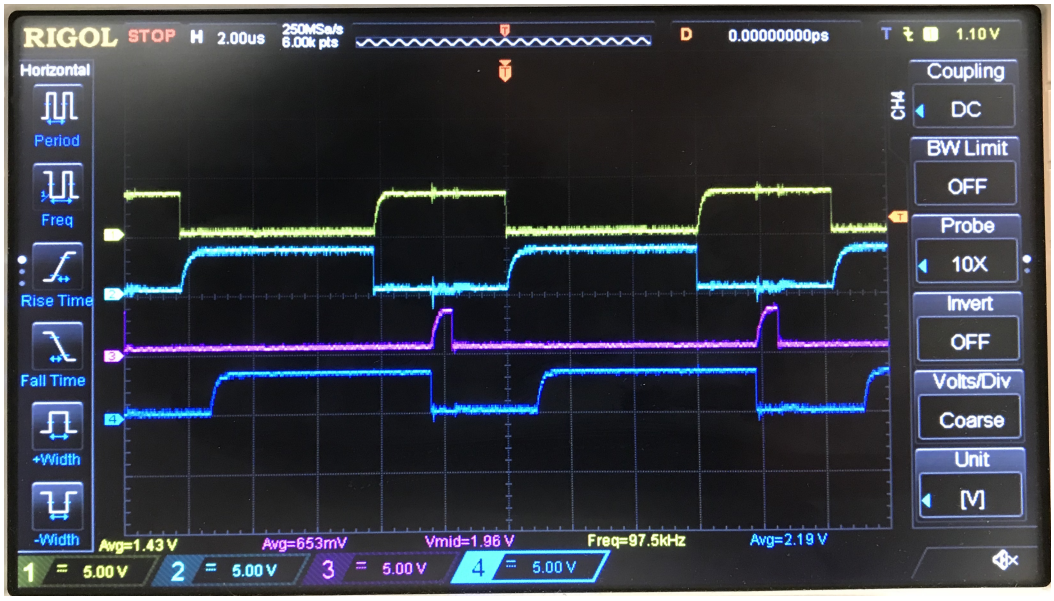


Figure 5-21: ResSC control signals when $V_{out} > V_{in}/2$. From top to bottom the signals are $Q1$, $Q2$, $Q3$, and $Q4$.

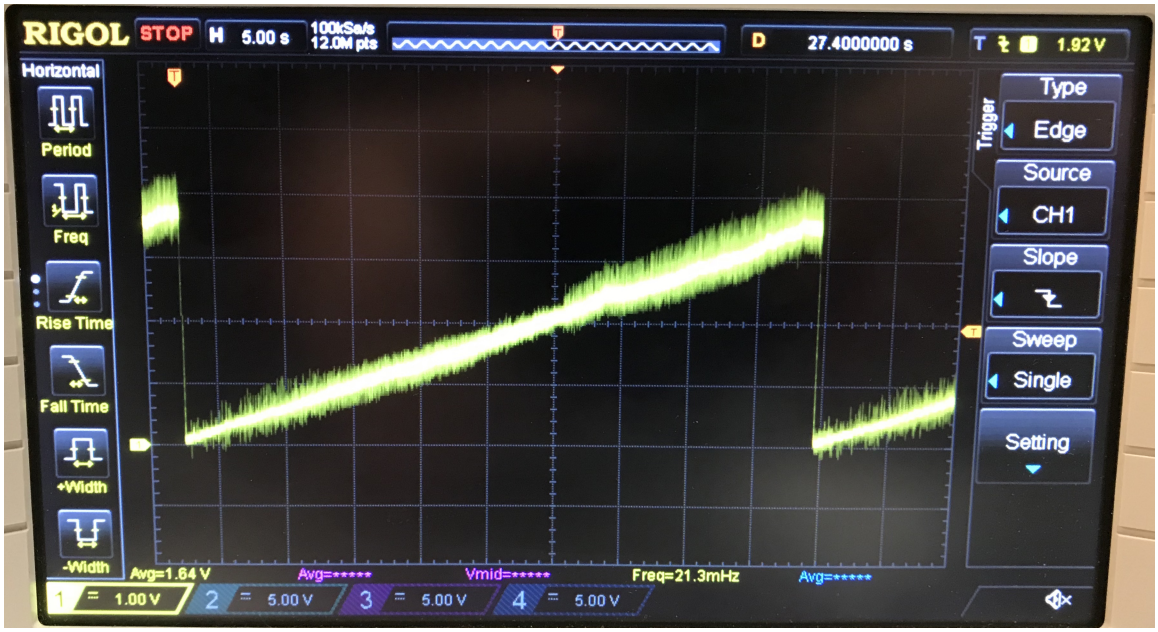


Figure 5-22: Output voltage ramp of 3 Level Buck over 45s with no offset in phase from 180deg. A 4Ohm load is connected with 5V input.

Figure 5-23: Output ripple for the ResSC at 1V 5A output.

5.7.3 Ćuk-Buck2

Time constraints prevented the completion of a prototype Ćuk-Buck2 circuit, but one could be constructed using the same PCBs as for the Three Level Buck and ResSC.

Chapter 6

Conclusions

6.1 Practical Takeaway

To the author of this work, it appears the takeway is that each converter had benefits and downsides and each could be most useful in different situations.

The Three Level Buck has very low output ripple, so it would be good for cases where output ripple is important. It however requires additional circuitry to sense the flying capacitor voltage, which could make it undesirable when simplicity is needed for feedback circuits.

The ResSC appears to have the lowest efficiency out of the three converters, but it does not require any voltage sensing circuity or a fifth switch to operate. It could be useful for applications like LED lighting that are more tolerant of output ripple and utilize parasitic inductance to operate.

The Ćuk-Buck2 appears to be generally more efficiency than the ResSC, but requires a fifth switch to operate with full range output. This could make the Ćuk-Buck2 the best choice for cases that need high efficiency and also require parasitic inductance to be used, as parasitic inductance of the flying capacitor can easily be absorbed into the Ćuk-Buck2 topology, but not the Three Level Buck.

6.2 Future Work

Additional work for feedback compensation for the Ćuk-Buck2 and ResSC will need to be performed. Work on feedback compensation was done as part of this research, but was not included in this thesis.

Comparisons were performed with silicon-based MOSFETs in simulation and real life, but other switching devices like GaN may need to be evaluated separately for these toplogies, as optimization could reveal different comparative results with different switch characteristics.

Appendix A

Equations

A.1 Simplified Converter Timing and Current Equations

ResSC - Buck Mode

Timing

$t_2 = \frac{V_{in}-2V_{out}}{V_{out}} * t_1 + t_1$

$t_3 = \pi * \sqrt{L * C} + t_2$

$t_4 = \frac{1}{f_{sw}}$

$Q_{in} = \frac{1}{2} * t_1 * \frac{V_{in}-2V_{out}}{L} * t_1 = \frac{V_{in}-2V_{out}}{2L} * (t_1)^2$

$Q_{out} = 2Q_{in} + Q_{in} * \frac{V_{in}-2V_{out}}{V_{out}} = (2 + \frac{V_{in}-2V_{out}}{V_{out}}) * Q_{in}$

$Q_{out} = (2 + \frac{V_{in}-2V_{out}}{V_{out}})(\frac{V_{in}-2V_{out}}{2L})(t_1)^2$

$I_{out} = f_{sw} * Q_{out} = f_{sw} * (2 + \frac{V_{in}-2V_{out}}{V_{out}})(\frac{V_{in}-2V_{out}}{2L})(t_1)^2$

$(t_1)^2 = \frac{I_{out}}{f_{sw} * (2+\frac{V_{in}-2V_{out}}{V_{out}})(\frac{V_{in}-2V_{out}}{2L})}$

$t_1 = \sqrt{\frac{I_{out}}{f_{sw} * (2+\frac{V_{in}-2V_{out}}{V_{out}})(\frac{V_{in}-2V_{out}}{2L})}}$

Currents

From 0 to t_1:
$|i_L| = \frac{V_{in}-2V_{out}}{L} * t$

From t_1 to t_2:
$|i_L| = \frac{V_{in}-2V_{out}}{L} * t_1 - (\frac{V_{out}}{L})(t - t_1)$

From t_2 to t_3:
Half period of sinusoid that has length $\pi\sqrt{L * C}$ and integral that equals the integral of charge from 0 to t_2.
$Integral = \frac{1}{2}(\frac{V_{in}-2V_{out}}{V_{out}} * t_1 + t_1) * \frac{V_{in}-2V_{out}}{L} * t_1$
$\omega = \frac{1}{\sqrt{L*C}}$
$A = (\frac{Integral}{2}) * \omega$
$|i_L| = A * \sin(\omega(t - t_2))$

RMS Currents

From 0 to t_1:
$(i_L)^2 = (\frac{V_{in}-2V_{out}}{L})^2 * t^2$
mean of the square of a triangular waveform is the amplitude squared over 3...
$\overline{(i_L)^2} = \frac{1}{3}(\frac{V_{in}-2V_{out}}{L})^2 * t_1^2$
$i_{L_{rms}} = \frac{1}{\sqrt{3}}(\frac{V_{in}-2V_{out}}{L}) * t_1$

From t_1 to t_2:
$(i_L)^2 = [(\frac{V_{in}-2V_{out}}{L})t_1 - (\frac{V_{out}}{L})(t - t_1)]^2 = [(\frac{V_{in}-V_{out}}{L})t_1 - (\frac{V_{out}}{L})t]^2$
mean of the square of a triangular waveform is the amplitude squared over 3, and amplitude is the same as from 0 to t_1...
$\overline{(i_L)^2} = \frac{1}{3}(\frac{V_{in}-2V_{out}}{L})^2 * t_1^2$
$i_{L_{rms}} = \frac{1}{\sqrt{3}}(\frac{V_{in}-2V_{out}}{L}) * t_1$

From t_2 to t_3:
$(i_L)^2 = A^2 * \sin^2(\omega(t - t_2))$
mean of the square of a sinusoidal waveform over half a period is the amplitude squared over 2...
$\overline{(i_L)^2} = \frac{A^2}{2}$
$i_{L_{rms}} = \frac{A}{\sqrt{2}}$

DCM 3-Level Buck - Buck Mode

Timing

$t_2 = \frac{V_{in}-2V_{out}}{2V_{out}} * t_1 + t_1$

$t_3 = \frac{1}{2}\frac{1}{f_{sw}}$

$t_4 = t_3 + t_1$

$t_5 = t_4 + t_2$

$t_6 = \frac{1}{f_{sw}}$

$Q_{in} = \frac{1}{2} * t_1 * \frac{V_{in}-2V_{out}}{2L} * t_1 = \frac{V_{in}-2V_{out}}{4L} * (t_1)^2$

$Q_{out} = 2(Q_{in} + Q_{in} * \frac{V_{in}-2V_{out}}{2V_{out}}) = (2 + \frac{V_{in}-2V_{out}}{V_{out}}) * Q_{in}$

$Q_{out} = (2 + \frac{V_{in}-2V_{out}}{V_{out}})(\frac{V_{in}-2V_{out}}{4L})(t_1)^2$

$I_{out} = f_{sw} * Q_{out} = f_{sw} * (2 + \frac{V_{in}-2V_{out}}{V_{out}})(\frac{V_{in}-2V_{out}}{4L})(t_1)^2$

$(t_1)^2 = \frac{I_{out}}{f_{sw} * (2+\frac{V_{in}-2V_{out}}{V_{out}})(\frac{V_{in}-2V_{out}}{4L})}$

$t_1 = \sqrt{\frac{I_{out}}{f_{sw} * (2+\frac{V_{in}-2V_{out}}{V_{out}})(\frac{V_{in}-2V_{out}}{4L})}}$

Currents

From 0 to t_1:
$|i_L| = \frac{V_{in}-2V_{out}}{2L} * t$

From t_1 to t_2:
$|i_L| = \frac{V_{in}-2V_{out}}{2L} * t_1 - (\frac{V_{out}}{L})(t - t_1)$

RMS Currents

From 0 to t_1:
$(i_L)^2 = (\frac{V_{in}-2V_{out}}{2L})^2 * t^2$
mean of the square of a triangular waveform is the amplitude squared over 3...
$\overline{(i_L)^2} = \frac{1}{3}(\frac{V_{in}-2V_{out}}{2L})^2 * (t_1)^2$
$i_{L_{rms}} = \frac{1}{\sqrt{3}}(\frac{V_{in}-2V_{out}}{2L}) * t_1$

From t_1 to t_2:
$(i_L)^2 = [\frac{V_{in}-2V_{out}}{2L} * t_1 - (\frac{V_{out}}{L})(t - t_1)]^2$
mean of the square of a triangular waveform is the amplitude squared over 3...
$\overline{(i_L)^2} = \frac{1}{3}(\frac{V_{in}-2V_{out}}{2L})^2 * (t_1)^2$
$i_{L_{rms}} = \frac{1}{\sqrt{3}}(\frac{V_{in}-2V_{out}}{2L}) * t_1$

ResCuk - Buck Mode

Timing

$t_2 = \frac{V_{in}-2V_{out}}{V_{out}} * t_1 + t_1$

$t_3 = \pi * \sqrt{L_{res} * C} + t_1$

$t_4 = \frac{1}{f_{sw}}$

$Q_{in} = \frac{1}{2} * t_1 * \frac{V_{in}-2V_{out}}{L} * t_1 = \frac{V_{in}-2V_{out}}{2L} * (t_1)^2$

$Q_{out} = 2Q_{in} + Q_{in} * \frac{V_{in}-2V_{out}}{V_{out}} = (2 + \frac{V_{in}-2V_{out}}{V_{out}}) * Q_{in}$

$Q_{out} = (2 + \frac{V_{in}-2V_{out}}{V_{out}})(\frac{V_{in}-2V_{out}}{2L})(t_1)^2$

$I_{out} = f_{sw} * Q_{out} = f_{sw} * (2 + \frac{V_{in}-2V_{out}}{V_{out}})(\frac{V_{in}-2V_{out}}{2L})(t_1)^2$

$(t_1)^2 = \frac{I_{out}}{f_{sw}*(2+\frac{V_{in}-2V_{out}}{V_{out}})(\frac{V_{in}-2V_{out}}{2L})}$

$t_1 = \sqrt{\frac{I_{out}}{f_{sw}*(2+\frac{V_{in}-2V_{out}}{V_{out}})(\frac{V_{in}-2V_{out}}{2L})}}$

Currents

From 0 to t_1:
$|i_L| = \frac{V_{in}-2V_{out}}{L} * t$

From t_1 to t_2:
$|i_L| = \frac{V_{in}-2V_{out}}{L} * t_1 - (\frac{V_{out}}{L})(t - t_1)$

From t_1 to t_3:
Half period of sinusoid that has length $\pi\sqrt{L * C}$ and integral that equals the integral of charge from 0 to t_1.
$Integral = \frac{1}{2}(\frac{V_{in}-2V_{out}}{L}) * (t_1)^2$
$\omega = \frac{1}{\sqrt{L_{res}*C}}$
$A = (\frac{Integral}{2}) * \omega$
$|i_C| = A * \sin(\omega(t - t_1))$

RMS Currents

From 0 to t_1:
$(i_L)^2 = (\frac{V_{in}-2V_{out}}{L})^2 * t^2$
mean of the square of a triangular waveform is the amplitude squared over 3...
$\overline{(i_L)^2} = \frac{1}{3}(\frac{V_{in}-2V_{out}}{L})^2 * (t_1)^2$
$i_{L_{rms}} = \frac{1}{\sqrt{3}}(\frac{V_{in}-2V_{out}}{L}) * t_1$

From t_1 to t_2:
$(i_L)^2 = [\frac{V_{in}-2V_{out}}{L} * t_1 - (\frac{V_{out}}{L})(t - t_1)]^2$
mean of the square of a triangular waveform is the amplitude squared over 3...
$\overline{(i_L)^2} = \frac{1}{3}(\frac{V_{in}-2V_{out}}{L})^2 * (t_1)^2$
$i_{L_{rms}} = \frac{1}{\sqrt{3}}(\frac{V_{in}-2V_{out}}{L}) * t_1$

From t_1 to t_3:
$(i_C)^2 = A^2 * \sin^2(\omega(t - t_1))$
mean of the square of a sinusoidal waveform over half a period is the amplitude squared over 2...
$\overline{(i_C)^2} = \frac{A^2}{2}$
$i_{C_{rms}} = \frac{A}{\sqrt{2}}$

Buck Mode Comparison

Timing

If:
$t_{1_{base}} = \sqrt{\frac{I_{out}}{f_{sw} * (2 + \frac{V_{in} - 2V_{out}}{V_{out}})(\frac{V_{in} - 2V_{out}}{2L})}}$
Then:
$t_{1_{ResSC}} = t_{1_{base}}$
$t_{1_{Buck}} = \sqrt{2} * t_{1_{base}}$
$t_{1_{ResCuk}} = t_{1_{base}}$

If:
$(t_2 - t_1)_{base} = \frac{V_{in} - 2V_{out}}{V_{out}} * t_{1_{base}}$
Then:
$(t_2 - t_1)_{ResSC} = (t_2 - t_1)_{base}$
$(t_2 - t_1)_{Buck} = \frac{1}{\sqrt{2}}(t_2 - t_1)_{base}$
$(t_2 - t_1)_{ResCuk} = (t_2 - t_1)_{base}$

RMS Currents

From 0 to t_1 and t_2 to t_3:
If:
$i_{L_{rms_{base}}} = \frac{1}{\sqrt{3}}(\frac{V_{in} - 2V_{out}}{L}) * t_{1_{base}}$
Then:
$i_{L_{rms_{ResSC}}} = i_{L_{rms_{base}}}$
$i_{L_{rms_{Buck}}} = \frac{1}{\sqrt{2}} i_{L_{rms_{base}}}$
$i_{L_{rms_{ResCuk}}} = i_{L_{rms_{base}}}$

Component RMS Comparison: $V_{out} << V_{in}/2$

$t_{1_{base}} = \sqrt{\frac{I_{out}}{f_{sw}*(2+\frac{V_{in}-2V_{out}}{V_{out}})(\frac{V_{in}-2V_{out}}{2L})}} = \sqrt{\frac{2*V_{out}*I_{out}}{f_{sw}*(V_{in})(\frac{V_{in}-2V_{out}}{L})}}$

MOSFET 1

$i_{ResSC_{rms}} = \frac{2^{0.75}}{\sqrt{3}} * (\frac{V_{out}*I_{out}}{V_{in}})^{0.75} * (\frac{V_{in}-2V_{out}}{L})^{0.25} * (f_{sw})^{-0.25} \approx 0.971 * (\frac{V_{out}*I_{out}}{V_{in}})^{0.75} * (\frac{V_{in}-2V_{out}}{L})^{0.25} * (f_{sw})^{-0.25}$

$i_{Buck_{rms}} = \frac{2^{0.5}}{\sqrt{3}} * (\frac{V_{out}*I_{out}}{V_{in}})^{0.75} * (\frac{V_{in}-2V_{out}}{L})^{0.25} * (f_{sw})^{-0.25} \approx 0.816 * (\frac{V_{out}*I_{out}}{V_{in}})^{0.75} * (\frac{V_{in}-2V_{out}}{L})^{0.25} * (f_{sw})^{-0.25}$

$i_{ResCuk_{rms}} = \frac{2^{0.75}}{\sqrt{3}} * (\frac{V_{out}*I_{out}}{V_{in}})^{0.75} * (\frac{V_{in}-2V_{out}}{L})^{0.25} * (f_{sw})^{-0.25} \approx 0.971 * (\frac{V_{out}*I_{out}}{V_{in}})^{0.75} * (\frac{V_{in}-2V_{out}}{L})^{0.25} * (f_{sw})^{-0.25}$

MOSFET 2

$i_{ResSC_{rms}} = (L)^{-0.25} * \sqrt{\frac{2^{1.5}}{3}(f_{sw})^{-0.5} * (V_{in}-2V_{out})^{1.5} * (\frac{I_{out}}{V_{in}})^{1.5} * (V_{out})^{0.5} + \frac{\pi}{8}(f_{sw})^{-1} * (\frac{I_{out}}{V_{in}})^{2} * (V_{in}-V_{out})^{2} * (C)^{-0.5}}$

$---- \approx (L)^{-0.25} * \sqrt{0.943(f_{sw})^{-0.5} * (V_{in}-2V_{out})^{1.5} * (\frac{I_{out}}{V_{in}})^{1.5} * (V_{out})^{0.5} + 0.393(f_{sw})^{-1} * (\frac{I_{out}}{V_{in}})^{2} * (V_{in}-V_{out})^{2} * (C)^{-0.5}}$

$i_{Buck_{rms}} = \frac{2^{0.5}}{\sqrt{3}} * (\frac{V_{out}*I_{out}}{V_{in}})^{0.75} * (\frac{V_{in}-2V_{out}}{L})^{0.25} * (f_{sw})^{-0.25} \approx 0.816 * (\frac{V_{out}*I_{out}}{V_{in}})^{0.75} * (\frac{V_{in}-2V_{out}}{L})^{0.25} * (f_{sw})^{-0.25}$

$i_{ResCuk_{rms}} = \sqrt{\frac{\pi}{8}} * (\frac{V_{out}*I_{out}}{V_{in}}) * (L_{res}*C)^{-0.25} * (f_{sw})^{-0.5} \approx 0.627 * (\frac{V_{out}*I_{out}}{V_{in}}) * (L_{res}*C)^{-0.25} * (f_{sw})^{-0.5}$

MOSFET 3

$i_{ResSC_{rms}} = \frac{2^{0.75}}{\sqrt{3}} * (\frac{V_{out}*I_{out}}{V_{in}})^{0.75} * (\frac{V_{in}-2V_{out}}{L})^{0.25} * (\frac{V_{in}-V_{out}}{V_{out}})^{0.5} * (f_{sw})^{-0.25}$

$---- \approx 0.971 * (\frac{V_{out}*I_{out}}{V_{in}})^{0.75} * (\frac{V_{in}-2V_{out}}{L})^{0.25} * (\frac{V_{in}-V_{out}}{V_{out}})^{0.5} * (f_{sw})^{-0.25}$

$i_{Buck_{rms}} = \frac{2^{0.5}}{\sqrt{3}} * (\frac{V_{out}*I_{out}}{V_{in}})^{0.75} * (\frac{V_{in}-2V_{out}}{L})^{0.25} * (\frac{V_{in}-V_{out}}{V_{out}})^{0.5} * (f_{sw})^{-0.25}$

$---- \approx 0.816 * (\frac{V_{out}*I_{out}}{V_{in}})^{0.75} * (\frac{V_{in}-2V_{out}}{L})^{0.25} * (\frac{V_{in}-V_{out}}{V_{out}})^{0.5} * (f_{sw})^{-0.25}$

$i_{ResCuk_{rms}} = \frac{2^{0.75}}{\sqrt{3}} * (\frac{V_{out}*I_{out}}{V_{in}})^{0.75} * (\frac{V_{in}-2V_{out}}{L})^{0.25} * (\frac{V_{in}-V_{out}}{V_{out}})^{0.5} * (f_{sw})^{-0.25}$

$---- \approx 0.971 * (\frac{V_{out}*I_{out}}{V_{in}})^{0.75} * (\frac{V_{in}-2V_{out}}{L})^{0.25} * (\frac{V_{in}-V_{out}}{V_{out}})^{0.5} * (f_{sw})^{-0.25}$

MOSFET 4

$i_{ResSC_{rms}} = \frac{\sqrt{\pi}}{2^{1.5}} * (\frac{V_{out}*I_{out}}{V_{in}}) * (\frac{V_{in}-V_{out}}{V_{out}}) * (L*C)^{-0.25} * (f_{sw})^{-0.5} \approx 0.627 * (\frac{V_{out}*I_{out}}{V_{in}}) * (\frac{V_{in}-V_{out}}{V_{out}}) * (L*C)^{-0.25} * (f_{sw})^{-0.5}$

$i_{Buck_{rms}} = \frac{2^{0.5}}{\sqrt{3}} * (\frac{V_{out}*I_{out}}{V_{in}})^{0.75} * (\frac{V_{in}-2V_{out}}{L})^{0.25} * (\frac{V_{in}-V_{out}}{V_{out}})^{0.5} * (f_{sw})^{-0.25}$

$---- \approx 0.816 * (\frac{V_{out}*I_{out}}{V_{in}})^{0.75} * (\frac{V_{in}-2V_{out}}{L})^{0.25} * (\frac{V_{in}-V_{out}}{V_{out}})^{0.5} * (f_{sw})^{-0.25}$

$i_{ResCuk_{rms}} = \sqrt{\frac{2^{1.5}}{3}(\frac{V_{in}-2V_{out}}{L})^{0.5} * (\frac{V_{out}*I_{out}}{V_{in}})^{1.5} * (\frac{V_{in}-2V_{out}}{V_{out}}) * (f_{sw})^{-0.5} + \frac{\pi}{8} * (\frac{V_{out}*I_{out}}{V_{in}})^{2} * (L_{res}*C)^{-0.5} * (f_{sw})^{-1}}$

L

$i_{ResSC_{rms}} = (L)^{-0.25} * \sqrt{\frac{2^{1.5}}{3}(f_{sw})^{-0.5}(V_{in}-2V_{out})^{0.5} * (\frac{I_{out}}{V_{in}})^{1.5} * (V_{in}-V_{out}) + \frac{\pi}{8}(f_{sw})^{-1}(V_{in}-V_{out})^{2} * (\frac{I_{out}}{V_{in}})^{2} * (C)^{-0.5}}$

$---- \approx (L)^{-0.25} * \sqrt{0.943(f_{sw})^{-0.5}(V_{in}-2V_{out})^{0.5} * (\frac{I_{out}}{V_{in}})^{1.5} * (V_{in}-V_{out}) + 0.393(f_{sw})^{-1}(V_{in}-V_{out})^{2} * (\frac{I_{out}}{V_{in}})^{2} * (C)^{-0.5}}$

$i_{Buck_{rms}} = \frac{2^{0.5}}{\sqrt{3}} * (I_{out})^{0.75} * (\frac{V_{in}-2V_{out}}{L})^{0.25} * (\frac{V_{out}}{V_{in}})^{0.25} * (f_{sw})^{-0.25} \approx 0.816 * (I_{out})^{0.75} * (\frac{V_{in}-2V_{out}}{L})^{0.25} * (\frac{V_{out}}{V_{in}})^{0.25} * (f_{sw})^{-0.25}$

$i_{ResCuk_{rms}} = \frac{2^{0.75}}{\sqrt{3}} * (\frac{V_{out}*I_{out}}{V_{in}})^{0.75} * (\frac{V_{in}-2V_{out}}{L})^{0.25} * (\frac{V_{in}-V_{out}}{V_{out}})^{0.5} * (f_{sw})^{-0.25}$

$---- \approx 0.971 * (\frac{V_{out}*I_{out}}{V_{in}})^{0.75} * (\frac{V_{in}-2V_{out}}{L})^{0.25} * (\frac{V_{in}-V_{out}}{V_{out}})^{0.5} * (f_{sw})^{-0.25}$

$i_{ResCuk_{resLrms}} = \sqrt{\frac{\pi}{8}} * (\frac{V_{out}*I_{out}}{V_{in}}) * (L_{res}*C)^{-0.25} * (f_{sw})^{-0.5} \approx 0.627 * (\frac{V_{out}*I_{out}}{V_{in}}) * (L_{res}*C)^{-0.25} * (f_{sw})^{-0.5}$

C

$i_{ResSC_{rms}} = (L)^{-0.25} * \sqrt{\frac{2^{1.5}}{3}(f_{sw})^{-0.5}(V_{in}-2V_{out})^{0.5} * (\frac{I_{out}}{V_{in}})^{1.5} * (V_{in}-V_{out}) + \frac{\pi}{8}(f_{sw})^{-1}(V_{in}-V_{out})^{2} * (\frac{I_{out}}{V_{in}})^{2} * (C)^{-0.5}}$

$---- \approx (L)^{-0.25} * \sqrt{0.943(f_{sw})^{-0.5}(V_{in}-2V_{out})^{0.5} * (\frac{I_{out}}{V_{in}})^{1.5} * (V_{in}-V_{out}) + 0.393(f_{sw})^{-1}(V_{in}-V_{out})^{2} * (\frac{I_{out}}{V_{in}})^{2} * (C)^{-0.5}}$

$i_{Buck_{rms}} = \frac{2}{\sqrt{3}} * (\frac{V_{out}*I_{out}}{V_{in}})^{0.75} * (\frac{V_{in}-2V_{out}}{L})^{0.25} * (f_{sw})^{-0.25} \approx 1.155 * (\frac{V_{out}*I_{out}}{V_{in}})^{0.75} * (\frac{V_{in}-2V_{out}}{L})^{0.25} * (f_{sw})^{-0.25}$

$i_{ResCuk_{rms}} = \sqrt{\frac{2^{1.5}}{3} * (\frac{V_{out}*I_{out}}{V_{in}})^{1.5} * (\frac{V_{in}-2V_{out}}{L})^{0.5} * (f_{sw})^{-0.5} + \frac{\pi}{8} * (\frac{V_{out}*I_{out}}{V_{in}})^{2} * (L_{res}*C)^{-0.5} * (f_{sw})^{-1}}$

Appendix B

Figures

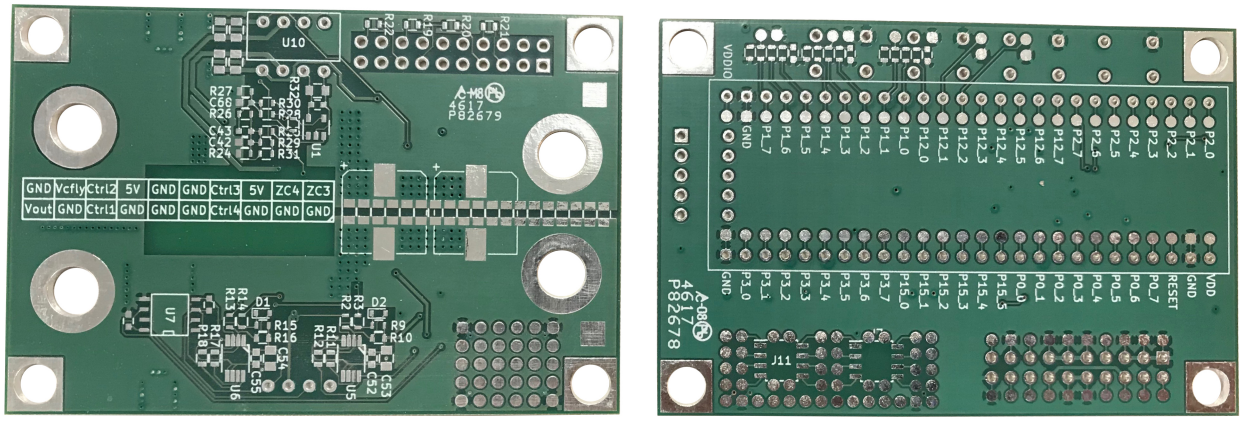


Figure B-1: Bottom sides of both Power and Control PCBs.

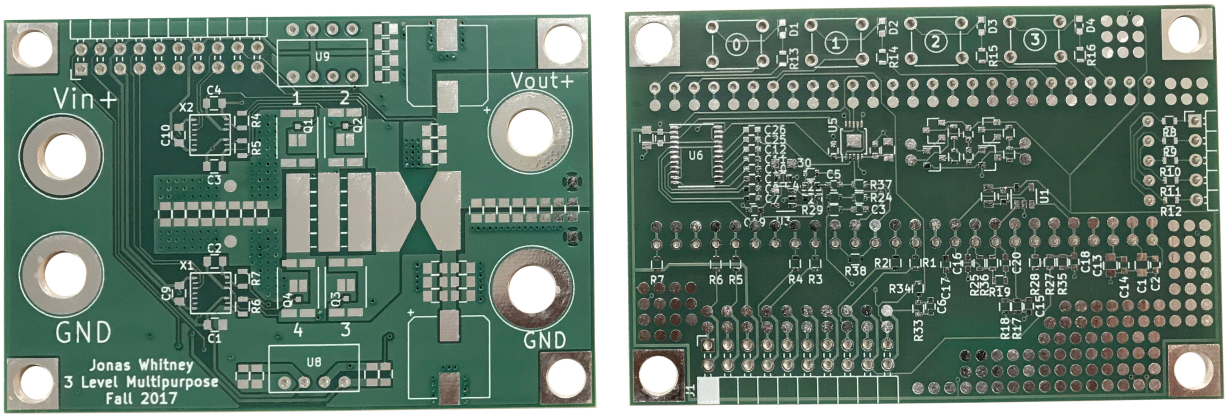


Figure B-2: Bottom sides of both Power and Control PCBs.

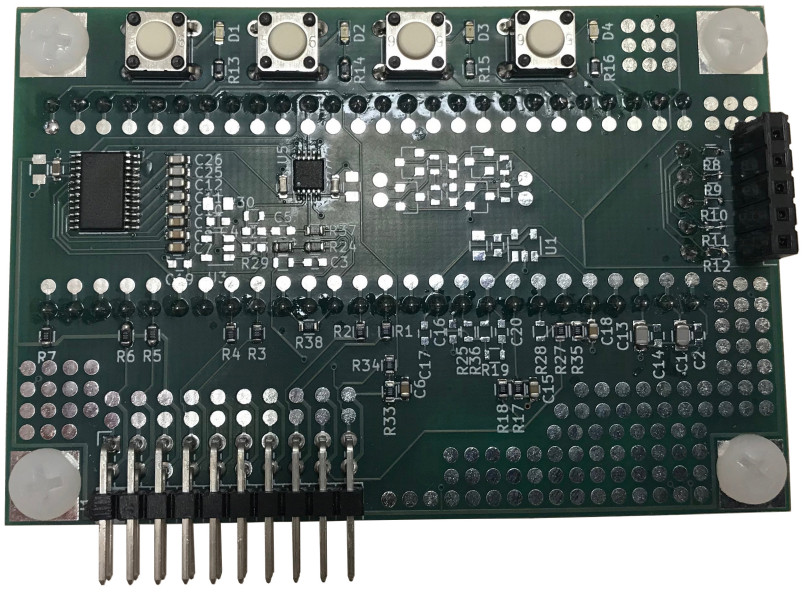


Figure B-3: Top of a populated control PCB.

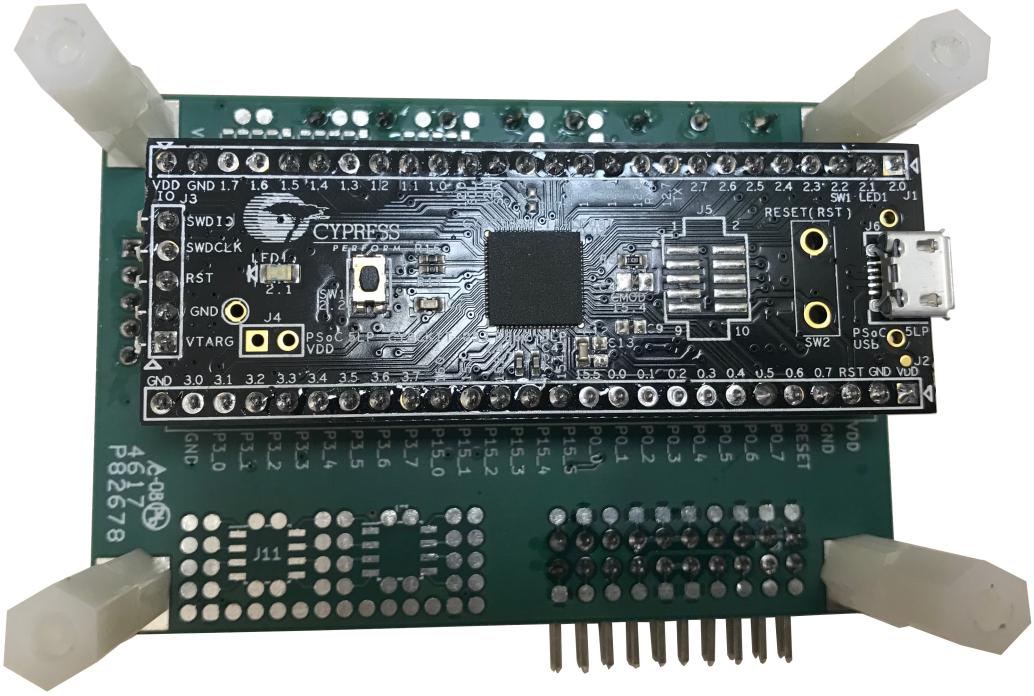


Figure B-4: Bottom of a populated control PCB.

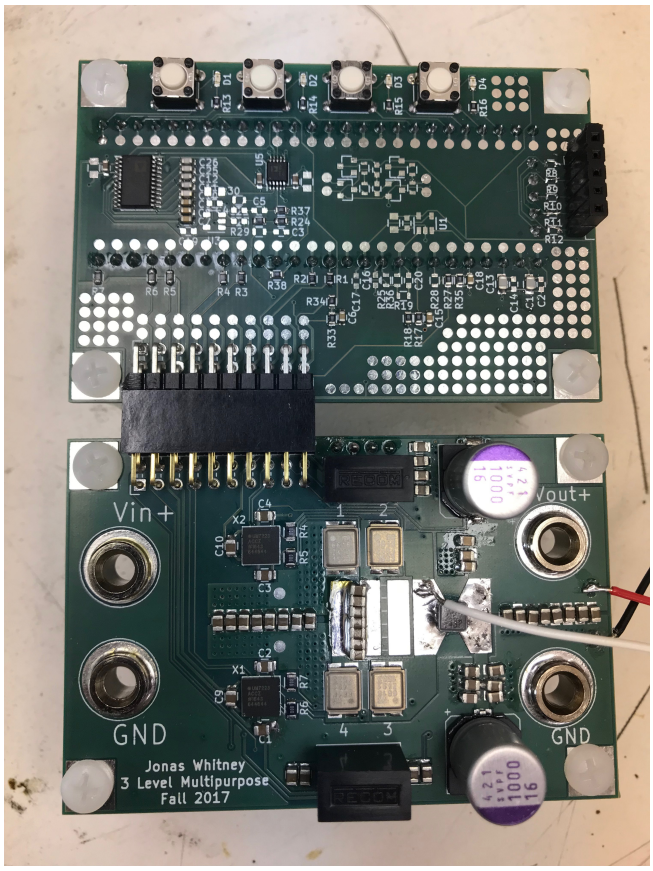


Figure B-5: Top of the 3 Level Buck power PCB connected to a control PCB.

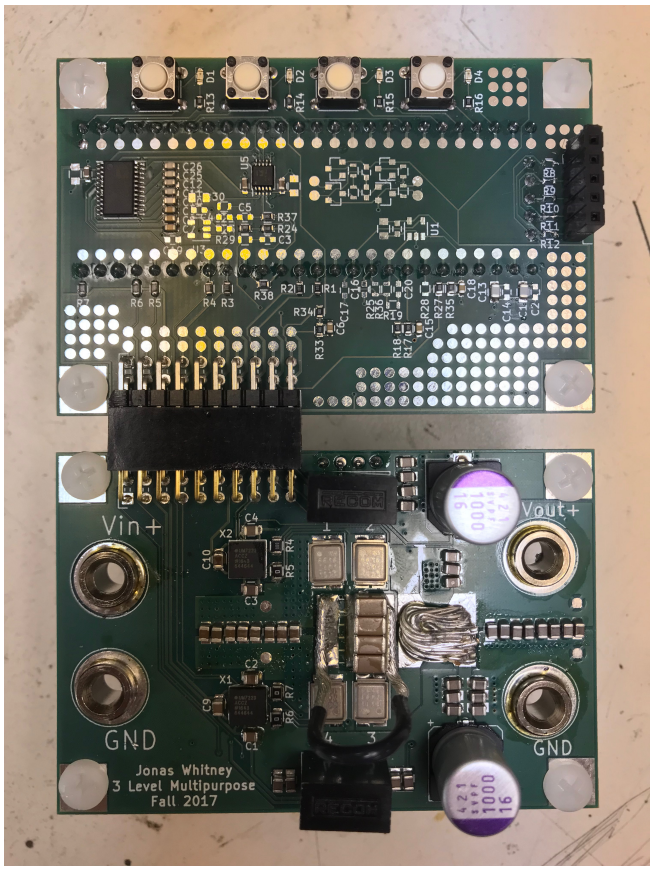


Figure B-6: Top of the ResSC power PCB connected to a control PCB.

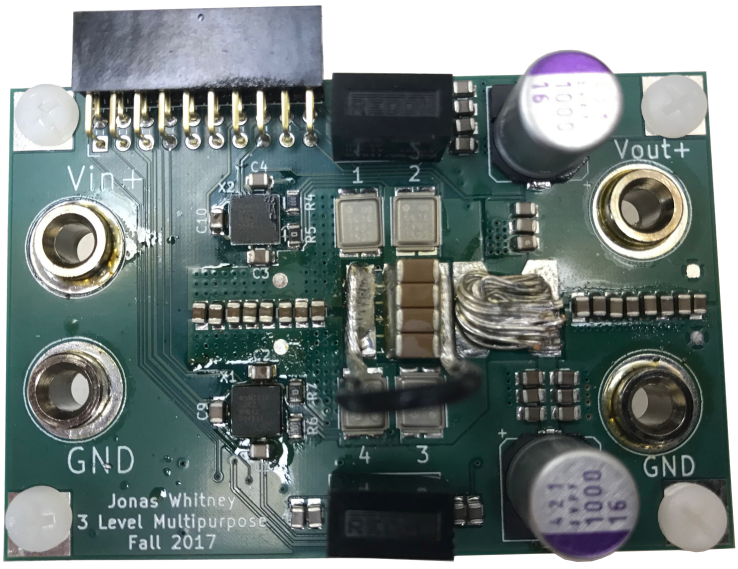


Figure B-7: Top of the populated ResSC power PCB.

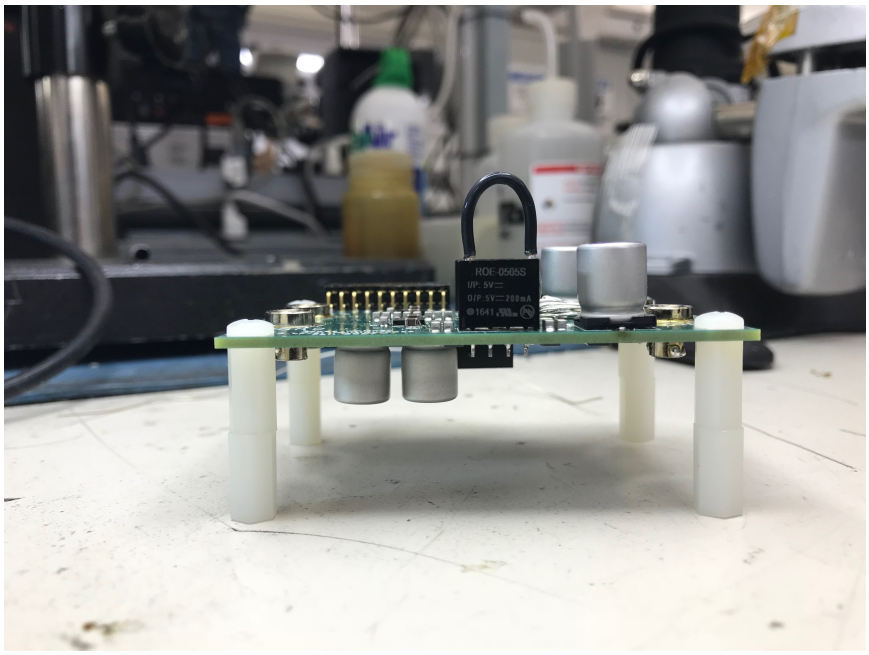


Figure B-8: Side of the populated ResSC power PCB.

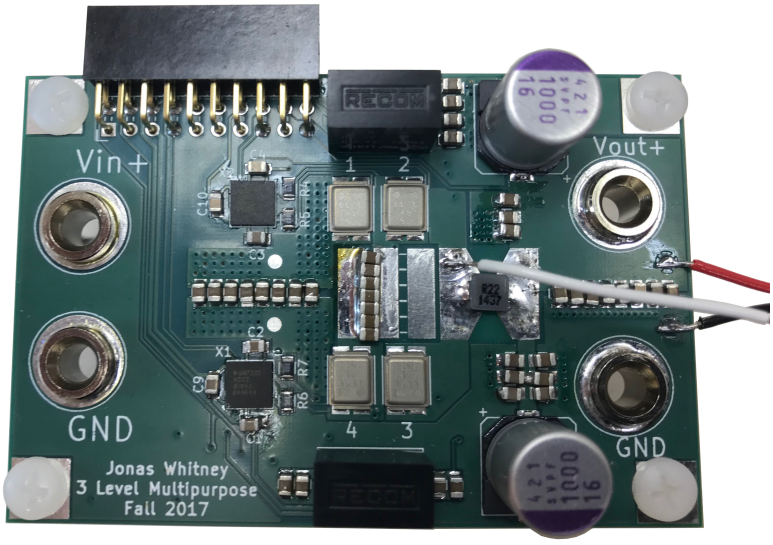


Figure B-9: Top of the populated 3 Level Buck power PCB.

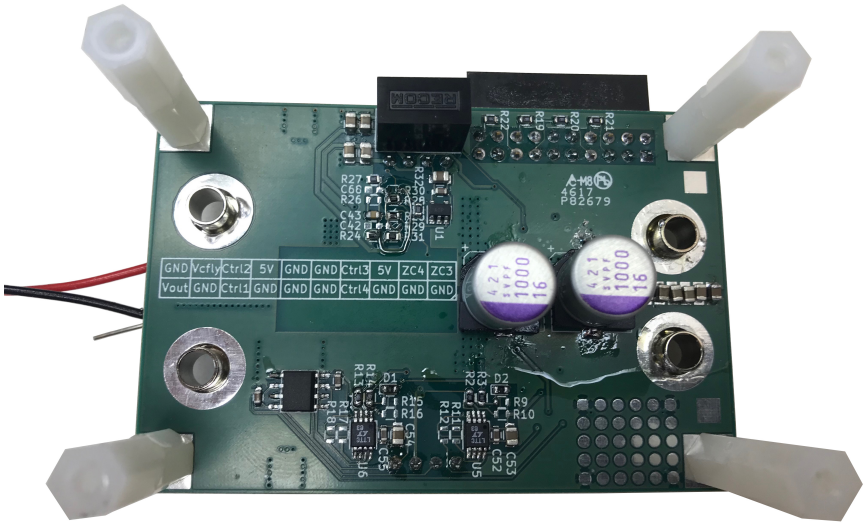


Figure B-10: Bottom of a populated power PCElementsReRe

Appendix C

Code

C.1 LTSpice RAW File Processing (1/3)

```
#reads the raw file from LTSpice and extracts data

class rawFile:

  #generated by this class:

  #headerNames - list of titles of data in the header
  #headerValues - list of the header data
  #varNames - list of variable names
  #varData - list of lists of data

  def __init__(self, fileName):
    self.fileName = fileName; #should not have an extension on it, will
      ↪ look for .raw


    #=== open file ===
    f = open(fileName+".raw", 'r');
    rawText = f.read();
    #split into parts
    #header
    rawHeader = rawText.split("Variables:\n", 1)[0];
    #variables
    rawVariables = rawText.split("Values:", 1)[0].split("Variables:\n", 1)
      ↪ [-1];
    #data
    rawData = rawText.split("Values:", 1)[-1];
    del rawText;

    #=== extract header data ===
    #split into lines
    header = rawHeader.split('\n');
    del rawHeader;
    #remove empty
    header2 = [];
```

```
    for line in header:
      if(len(line)>0):
        header2.append(line);
    del header;
    #get names and values, remove whitespace
    self.headerNames = [];
    self.headerValues = [];
    for line in header2:
      self.headerNames.append(line.split(": ",1)[0].lstrip());
      self.headerValues.append(line.split(": ",1)[-1].lstrip());
    del header2;
    #=== extract variable data ===
    #split into lines
    variables = rawVariables.split('\n');
    del rawVariables;
    #remove empty
    variables2 = [];
    for line in variables:
      if(len(line)>0):
        variables2.append(line);
    del variables;
    #get names, remove whitespace
    self.varNames = [];
    for line in variables2:
      self.varNames.append(line.split("\t")[2].lstrip());
    del variables2;

    #=== extract data data ===
    #split into lines
    data = rawData.split('\n');
    del rawData;
    #remove empty
    data2 = [];
    for line in data:
      if(len(line)>0):
        data2.append(line);
    del data;
    #get values, remove whitespace
    varDataTemp = [];
    for line in data2:
      varDataTemp.append(float(line.split("\t")[-1].lstrip()));
    del data2;
    #split into different variables
    self.varData = [];
    for x in range(len(self.varNames)):
      self.varData.append([]);
    for y in range( int(len(varDataTemp)/len(self.varNames)) ):
      for z in range(len(self.varNames)):
        self.varData[z].append(varDataTemp[y*len(self.varNames)+z]);
    del varDataTemp;
    #unequal timesteps, so calculate time per data point
    self.delTime = [i - j for i, j in zip(self.getVar("time")[1:], self.
      ↪ getVar("time")[:-1])];
    self.endTime = self.getVar("time")[-1];

  #get variable data from variable name
  def getVar(self, varName):
    try:
```

```
      return self.varData[self.varNames.index(varName)];
    except:
      return [float('NaN')];
  #get average value from variable name
  def avgVar(self, varName):
    try:
      return sum([i*j for i,j in zip(self.getVar(varName)[:-1],self.delTime
          ↪ )])/self.endTime;
    except:
      return float('NaN');


#=====================================
```

C.2 LTSpice Multithreaded Batch Simulation and Processing (2/3)

```
import subprocess; #for calling command line
import spiceRaw;
import itertools;
import pp;
import numpy;
import time;
import gc;
import glob;
import os;
#from concurrent.futures import ProcessPoolExecutor;

ppservers = ();
job_server = pp.Server(ncpus=10, ppservers=ppservers);
#job_server = pp.Server(ppservers=ppservers);
#print ("Starting pp with", job_server.get_ncpus(), "workers");

class spiceBatch:

  global job_server;

  def __init__(self, file, outFile, leaveFiles = False):
    #create a netlist from asc
    #subprocess.call('"C:\Program Files (x86)\LTC\LTspiceIV\scad3.exe" -
        ↪ netlist ' + file + ".asc", shell=True);
    #open the generated netlist file
    #try:
    # self.netlist = open(file + ".cir", "r");
    #except:
    # self.netlist = open(file + ".net", "r");
    #read the netlist file as lines
    #self.netlistLines = self.netlist.readline();

    #open the provided netlist file
    with open(file + ".net") as f:
      self.netlistLines = f.readlines();
```

```
    #check to see if ".save" and not "*.save" exists, otherwise memory
        ↪ usage is too large
    for line in self.netlistLines:
      if("*.save" in line):
        print("No valid save statement in netlist, please add .save
            ↪ statement to reduce memory usage");
        exit();
    look = 0;
    for line in self.netlistLines:
      if(".save" in line):
        look = 1;
    if look==0:
      print("No valid save statement in netlist, please add .save statement
          ↪  to reduce memory usage");
      exit();

    self.outFile = outFile;
    self.first = 0;
    self.leaveFiles = leaveFiles;
    self.times = [];
    self.startTime = 0;
  def sim(self, paramNames, paramValues, num):

    outputNetlist = [];

    #modify template netlist
    for line in self.netlistLines:
      replaced = False;
      for x in range(len(paramNames)):
        if( (".param" in line) and (line.count(" "+paramNames[x]+" ")==1) )
            ↪ :
          outputNetlist.append(line.split(paramNames[x])[0] + paramNames[x]
              ↪  + " " +str(paramValues[x]) + '\n');
          replaced = True;
      if(not replaced):
          outputNetlist.append(line);

    #write circuit file
    fileName = "spiceBatch"
    for x in paramValues:
      fileName += "-"+str(x);
    #fileName = "SC_"+str(width)+'-'+str(voltage)+'-'+str(inductor);
    textFile = open(fileName+".cir", 'w');
    for line in outputNetlist:
      textFile.write(line);
    textFile.close();
    del outputNetlist;

    #simulate the circuit
    subprocess.call('"C:\Program Files (x86)\LTC\LTspiceIV\scad3.exe" -
        ↪ ascii -b ' + fileName + ".cir", shell=True);

    #process the result
    a = spiceRaw.rawFile(fileName);

    #remove files created
    if(not self.leaveFiles):
      try:
```

```
        os.remove(fileName+".cir");
      except:
        pass;
      try:
        os.remove(fileName+".log");
      except:
        pass;
      try:
        os.remove(fileName+".op.raw");
      except:
        pass;
      try:
        os.remove(fileName+".raw");
      except:
        pass;

    #signal sim is complete with file
    doneFile = open(str(num)+".done", 'w');
    doneFile.write(" ");
    doneFile.close();

    #return the rawfile object
    return (list(paramValues),a);



  def run(self, paramNames, paramValues):

    self.paramNames = paramNames;

    global job_server;
    self.first = 1;

    jobs = {};
    #figure out the number of total combinations
    numberOfJobs = 1;
    for x in range(len(paramNames)):
      numberOfJobs = numberOfJobs * len(paramValues[x]);

    #jobCorrelation = {};
    #enumerate the combinations and start running
    num = 0;
    for combo in list(itertools.product(*paramValues)):
      #print((paramNames, combo));
      #jobs.append(job_server.submit(self.sim, (paramNames,combo,num), (
          ↪ self.sim,), ("spiceRaw","subprocess")))
      jobs[str(num)] = (job_server.submit(self.sim, (paramNames,combo,num),
          ↪  (self.sim,), ("spiceRaw","subprocess")));
      num = num + 1;
    self.startTime = time.time();

    results = 0;
    while(results<numberOfJobs):
      print '\r' + str(int(results))+"/"+str(numberOfJobs),
      #get results

      #see if any sims are done
      done = glob.glob("*.done")
```

```
      if(len(done)>0):
        doneNum = int(done[0].split(".")[0]);
        result = jobs[str(doneNum)]();
        self.writeToFile(result, paramNames);
        del jobs[str(doneNum)];
        results = results + 1;
        del result;
        #print progress
        secRemain = ((time.time()-self.startTime)/results * (numberOfJobs-
            ↪ results));
        print '\r' + str(int(results))+"/"+str(numberOfJobs) + '\tMin
            ↪ Remaining: ' + str(int(secRemain/60.0)),
        gc.collect()

        #print("Removing file")
        #print(done[0])
        try:
          os.remove(done[0]);
        except:
          pass;

      #with ProcessPoolExecutor() as p:
      # f = p.submit(jobs[0]())
      # result = f.result(timeout=5)

      time.sleep(1);
      #del jobs[0];

      #save result to file
      #writeSuccess = 0;

      #print("here")

      #print("here2")


    return results;



  def writeToFile(self, output, inputParamNames):

    #array to hold names of data
    dataNames = [];
    #array to hold extracted outputs of spiceRaw objects
    extractedOutputs = [];

    #print(output);
    (conditions, a) = output;
    del output;
    #unequal timesteps, so get time per data point
    times = a.delTime;
    #stopping time, data starts at 0
    endTime = a.getVar("time")[-1];

    avgOutputVoltage = a.avgVar("V(out)");
    outputVoltageRipplePtP = abs(numpy.amax(a.getVar("V(outfilt)")) - numpy
        ↪ .amin(a.getVar("V(outfilt)")));
```

```
    #avgOutputCurrent = a.avgVar("I(Output)");
    outputCurrent = conditions[inputParamNames.index("Iout")];

    avgPowerIn = a.avgVar("V(inputpower)");

    powerOutArray = numpy.multiply(a.getVar("V(out)"), outputCurrent)[:-1];
    del a;
    avgPowerOut = abs(numpy.dot(powerOutArray, times))/endTime;
    del times;

    temp = list(conditions);
    del conditions;
    temp2 = list(inputParamNames);
    temp2.append("Efficiency");
    temp.append( avgPowerOut/avgPowerIn );
    temp2.append("Output Voltage (V)");
    temp.append( avgOutputVoltage );
    temp2.append("Output Ripple PtP (V)");
    temp.append( outputVoltageRipplePtP );

    #temp2.append("Avg Pow In");
    #temp.append( avgPowerIn );
    #temp2.append("Avg Pow Out");
    #temp.append( avgPowerOut );

    dataNames = list(temp2);
    extractedOutputs.append( temp );
    #print("here3")
    #write to file
    writeSuccess = 0;
    while(writeSuccess==0):
      try:
        fileOut = open(self.outFile, 'a');
        #write header labels
        if(self.first == 1):
          fileOut.write(",".join(dataNames) + "\n");
          self.first = 0;
        #write data
        for result in extractedOutputs:
          for chunk in result:
            fileOut.write(str(chunk)+',');
          fileOut.write('\n');
        fileOut.close();
        writeSuccess = 1;
      except:
        junk=input("Error writing file, press enter to try again.");
```

C.3 LTSpice Example Batch File Run (3/3)

```
import spiceBatch;
import numpy;
import math;
import time;
```

```
#input parameters
inputParamNames = [];
inputParamValues = [];

inputParamNames.append("freq");
inputParamValues.append([3.0e6]);
#inputParamValues.append(numpy.arange(1.0e6,3.5e6,0.5e6));

inputParamNames.append("size");
inputParamValues.append([750e3]);
#inputParamValues.append(numpy.arange(100e3,2000e3,50e3));
inputParamNames.append("ind");
inputParamValues.append([10e-9]);
#inputParamValues.append(numpy.arange(1e-9,11e-9,1e-9));

inputParamNames.append("set");
#inputParamValues.append([1.0]);
inputParamValues.append(numpy.arange(0.1,5.0,0.1));
inputParamNames.append("gateMult");
inputParamValues.append([0.6]);
inputParamNames.append("Iout");
inputParamValues.append([1,5,10]);

#simulation timing
#inputParamNames.append("simTime");
#inputParamValues.append(["500/{freq}"]);
#inputParamNames.append("startTime");
#inputParamValues.append(["400/{freq}"]);

#name of output file
c = "output_3_3Vdev.csv";
#Run the simulations, output is array of spiceRaw objects from the
    ↪ simulations
b = spiceBatch.spiceBatch("3_level_template", c);
startTime = time.time();
b.run(inputParamNames, inputParamValues);
elapTime = (time.time()-startTime)
print("");
print("Finished. Elapsed Time: " + str(int(elapTime/60.0)) + "min " + str(
    ↪ int(elapTime)%60) + "sec.");
```

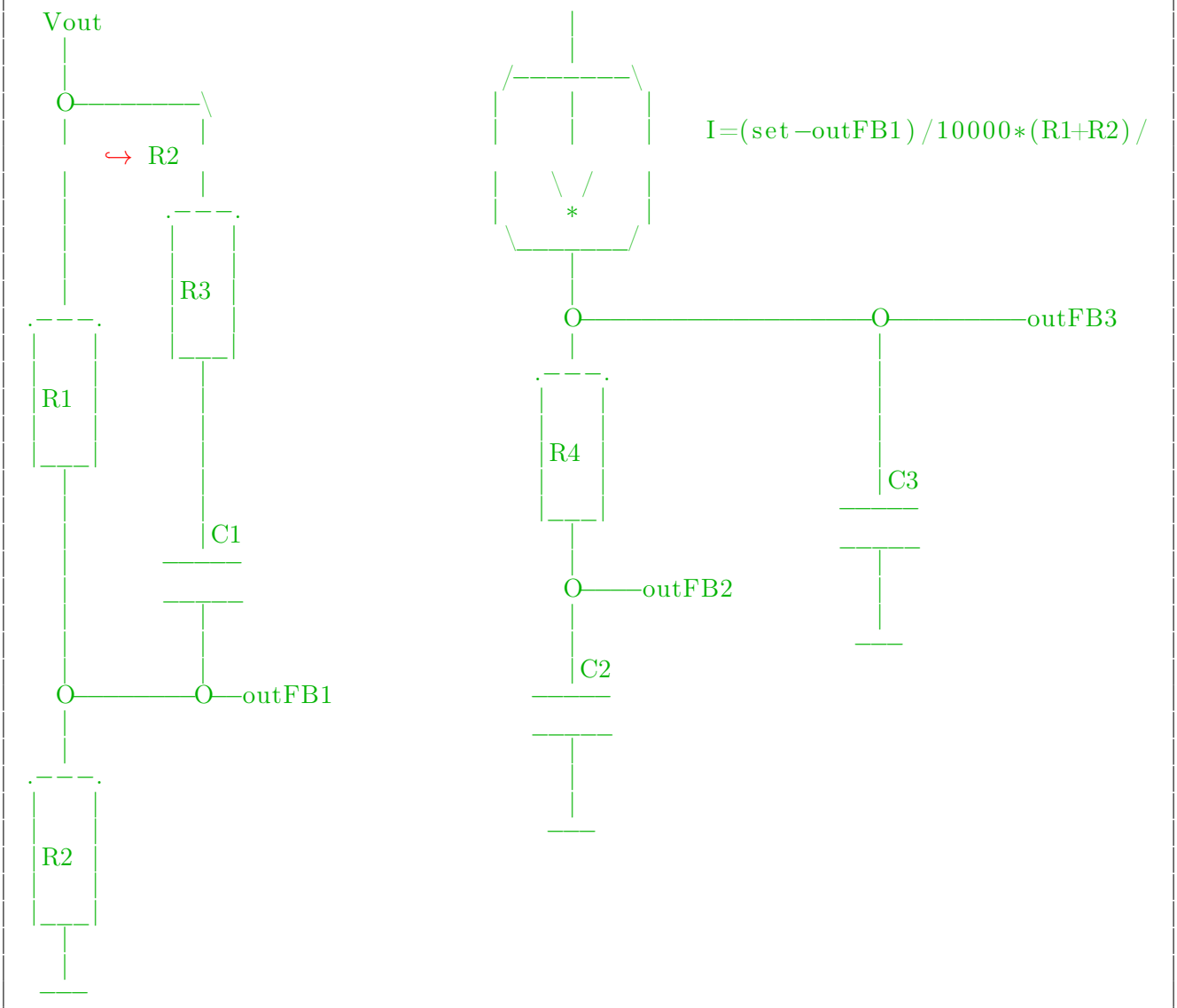
C.4 Python ResSC State Space Modeling

```
import math;
import graph;
import random;
import numpy as np;
import control;
import matplotlib.pyplot as plt;
from matplotlib.widgets import Slider, Button, RadioButtons;
import matplotlib.lines as lines;
```

```
#circuit parameters
Cfly = 4e-6;
Cout = 100e-6;
L = 2e-9;
T = 2*math.pi*(L*(Cfly**-1+Cout**-1)**-1)**0.5;
fsw = 2e6;
Iout = 5.0;
Vin = 5.0;
Vset = 1.0;

#initial conditions (will be re-written)
Vcfly = float(Vset);
Vout = float(Vset);
outFB3 = 0.0;
outFB2 = 0.0;
outFB1 = 0.0;

#feedback system
"""
```



```
d/dt(outFB1) = d/dt(out)*(R3/R1+1) + (R3/R1-1)/C1*out - (R3/R1+R3/R2)/C1*outFB1
d/dt(outFB2) = 1/(R4*C2)*outFB3 - 1/(R4*C2)*outFB2
d/dt(outFB3) = set*(R1+R2)/(10000*C3*R2) - (R1+R2)/(10000*C3*R2)*outFB1 - 1/(R4*C3)*outFB3 + 1/(R4*C3)*outFB2
"""
#initial feedback circuit values
R1 = 19e3;
R2 = 1e3;
R3 = 10.0;
R4 = 2000.0;
C1 = 1.0e-12;
C2 = 4e-9;
C3 = 0.5e-12;



def advancePeriod(inputArray):
  #simulate a period by using math equations
  #Vout is inputArray[0]
  #Vcfly is inputArray[1]
  #outFB3 is inputArray[2]

  #state D
  Vcfly_1 = (-1.0+2.0*Cfly/(Cout+Cfly))*inputArray[1] + (2.0*Cout/(Cout+Cfly))*inputArray[0] + (-0.5*T/(Cout+Cfly)*Iout);
  Vout_1 = (2.0*Cfly/(Cout+Cfly))*inputArray[1] + (-1.0+2.0*Cout/(Cout+Cfly))*inputArray[0] + (-0.5*T/(Cout+Cfly)*Iout);
  #state B
  Vcfly_2 = (1.0-2.0*(math.sin(math.pi*inputArray[2]))**2*Cout/(Cout+Cfly))*Vcfly_1 - (2.0*Cout/(Cout+Cfly)*(math.sin(math.pi*inputArray[2]))**2)*Vout_1 + 2.0*Cout/(Cout+Cfly)*Vin*(math.sin(math.pi*inputArray[2]))**2 + inputArray[2]*Iout*T/(Cout+Cfly)*(math.sin(math.pi*inputArray[2]))**2;
  Vout_2 = (-2.0*Cfly/(Cout+Cfly)*(math.sin(math.pi*inputArray[2]))**2)*Vcfly_1 + (1.0 - 2.0*Cfly/(Cout+Cfly)*(math.sin(math.pi*inputArray[2]))**2)*Vout_1 + (2.0*Cfly/(Cout+Cfly)*Vin*(math.sin(math.pi*inputArray[2]))**2 - inputArray[2]*Iout*T/(Cout+Cfly)*(math.sin(math.pi*inputArray[2]))**2);
  iLtemp = math.sqrt((-Cfly*(Vcfly_2**2-Vcfly_1**2) - Cout*(Vout_2**2-Vout_1**2) + Iout*(Vout_1+Vout_2)*T*inputArray[2] + 2*Cfly*Vin*(Vcfly_2-Vcfly_1))/L);
  #state C
  Vcfly_3 = math.sqrt(Vcfly_2**2+L/Cfly*iLtemp**2);
  Vout_3 = Vout_2 - Iout/Cout*(1.0/fsw -0.5*T-inputArray[2]*T);
  #update variables
  return [float(Vout_3), float(Vcfly_3), float(inputArray[2])];


#sets outFB3 and Vcfly for current Iout Vout via bisection search
def updateFB3():
  global outFB3;
  global Vout;
  global Vcfly;
  #save what Vout was before running
  temp_Vout = float(Vout);
  #duty cycle steps will be taken down to 2**-levels
  levels = 40;
```

```
  #start searching with duty cycle of one quarter (as ResSC is only being
     ↪ looked at in buck mode)
  outFB3 = 0.25;
  maxDuty = 0.5;
  minDuty = 0.0;
  while(levels>=1):
    #sim to get output voltage
    diff = 1.0;
    diff2 = 1.0;
    num = 0;
    while(((diff>0.001) or (diff2>0.001)) and num<10000):
      diff = float(Vout);
      diff2 = float(Vcfly);
      for i in range(100):
        [Vout,Vcfly,outFB3] = advancePeriod([Vout,Vcfly,outFB3]);
      diff = abs(diff-Vout);
      diff2 = abs(diff2-Vcfly);
      num += 1;
    if(Vout>temp_Vout):
      maxDuty = float(outFB3);
      outFB3 = float(0.5*outFB3 + 0.5*minDuty);
    else:
      minDuty = float(outFB3);
      outFB3 = float(0.5*outFB3 + 0.5*maxDuty);

    levels = levels - 1;
  #return Vout to what it was set to before
  #Vcfly and outFB3 have been re-written as part of the simulations
  Vout = float(temp_Vout);


#get what outFB3 and Vcfly should be
updateFB3();
#save original values
orig_Vout = float(Vout);
orig_Vcfly = float(Vcfly);
orig_Vin = float(Vin);
orig_Iout = float(Iout);
orig_Vset = float(Vset);
orig_outFB3 = float(outFB3);

#data save for plotting
t = [0.0];
##############
outFB3_t = [float(outFB3)];
Vfly_t = [float(Vcfly)];
Vout_t = [float(Vout)];
##############################
Vfly_t2 = [0*float(Vcfly)];
Vout_t2 = [0*float(Vout)];
outFB3_t2 = [0*float(outFB3)];
outFB2_t2 = [float(outFB2)];
outFB1_t2 = [float(outFB1)];
```

```
"""
    [ Vout ]    [
       ↪ a,                                  b,                                  c,
       ↪           0,                                  0] [ Vout ]    [g]
       ↪    [i]         [k]         [           0           ]
 d  [ Vcfly]    [
    ↪ d,                                  e,                                  f,
    ↪           0,                                  0] [ Vcfly]    [h]    [j
    ↪ ]         [l]         [           0           ]
d_t [outFB3] = [
   ↪ 0,                                  0,                        -1/(R4*C3),   1/(
   ↪ R4*C3),              -(R1+R2)/(10000*C3*R2)]*[outFB3] + [0] + [0]*Vin
   ↪  + [0]*Iout + [(R1+R2)/(10000*C3*R2)]*set
    [outFB2]    [
       ↪ 0,                                  0,                         1/(R4*C2),
       ↪ -1/(R4*C2),                                  0] [outFB2]    [0]
       ↪    [0]         [0]         [           0           ]
    [outFB1]    [((R3/R1)/(R3/R1+R3/R2+1))/C1 + a*((R3/R1+1)/(R3/R1+R3/R2+1)
       ↪ ), b*((R3/R1+1)/(R3/R1+R3/R2+1)), c*((R3/R1+1)/(R3/R1+R3/R2+1)),
       ↪           0, -((R3/R1+R3/R2)/(R3/R1+R3/R2+1))/C1] [outFB1]    [0]
       ↪    [0]         [0]         [           0           ]
"""
"""
 d  [ Vout ]    [a, b, c] [ Vout ]    [j]
d_t [ Vcfly] = [d, e, f]*[ Vcfly] + [k]
    [outFB3]    [g, h, i] [outFB3]    [l]
"""
#number of periods to average over
numPeriods = 2;
#takes an array of the initial states of the state space variables
#and a function that returns the next period of the system
def getStateSpace(initialState, getNextPeriod):
  initialState = np.array(initialState);
  n = len(initialState);
  #returns linearized state space models of current system
  #has the form dx/dt = A*delta_x and solves for A
  dxdt = np.zeros((n,n));
  delta_x = np.zeros((n,n));

  newState = np.array(initialState);
  #get how these move over numPeriods periods
  for x in range(numPeriods):
    newState = advancePeriod(newState);
  oneP = np.array(newState);
  #get how these move over another numPeriods periods
  for x in range(numPeriods):
    newState = advancePeriod(newState);
  twoP = np.array(newState);


  for ssv in range(n):
    newState = np.array(initialState);
    newState[ssv] = 1.01*newState[ssv];

    dxdt = np.transpose(dxdt);
    dxdt[ssv] = np.array(-1*newState);
    dxdt = np.transpose(dxdt);
```

```
    delta_x = np.transpose(delta_x);
    delta_x[ssv] = np.array(-1*initialState);
    delta_x = np.transpose(delta_x);


    for x in range(numPeriods):
      newState = advancePeriod(newState);
    delta_x = np.transpose(delta_x);
    delta_x[ssv] = delta_x[ssv] + np.array(newState);
    delta_x = np.transpose(delta_x);


    for x in range(numPeriods):
      newState = advancePeriod(newState);
    dxdt = np.transpose(dxdt);
    dxdt[ssv] = dxdt[ssv] + np.array(newState);
    dxdt = np.transpose(dxdt);



  #print(dxdt)
  #print(delta_x);
  dxdt = dxdt*fsw/(2.0*numPeriods)
  A = np.dot(dxdt,np.linalg.inv(delta_x));
  #print(A);
  return A;

  """
  #get how move with a wiggle in Vin
  Vout = float(orig2_Vout);
  Vcfly = float(orig2_Vcfly);
  outFB3 = float(orig2_outFB3);
  Vin = float(orig2_Vin);
  Vin = 1.01*Vin;
  for x in range(numPeriods):
    advancePeriod();
  wVin_twoP_Vout = float(Vout);
  wVin_twoP_Vcfly = float(Vcfly);
  b1_Vin = (wVin_twoP_Vout-twoP_Vout)/(0.01*orig2_Vin)/(1.0/fsw)/float(
      ↪ numPeriods);
  b2_Vin = (wVin_twoP_Vcfly-twoP_Vcfly)/(0.01*orig2_Vin)/(1.0/fsw)/float(
      ↪ numPeriods);
  b3_Vin = 0;

  B_Vin = np.array([[b1_Vin],[b2_Vin],[b3_Vin]]);

  #get how move with a wiggle in Iout
  Vout = float(orig2_Vout);
  Vcfly = float(orig2_Vcfly);
  outFB3 = float(orig2_outFB3);
  Vin = float(orig2_Vin);
  Iout = float(orig2_Iout);
  Iout = 1.01*Iout;
  for x in range(numPeriods):
    advancePeriod();
```

```
  wIout_twoP_Vout = float(Vout);
  wIout_twoP_Vcfly = float(Vcfly);
  b1_Iout = (wIout_twoP_Vout-twoP_Vout)/(0.01*orig2_Iout)/(1.0/fsw)/float(
     ↪ numPeriods);
  b2_Iout = (wIout_twoP_Vcfly-twoP_Vcfly)/(0.01*orig2_Iout)/(1.0/fsw)/float
     ↪ (numPeriods);
  b3_Iout = 0;

  B_Iout = np.array([[b1_Iout],[b2_Iout],[b3_Iout]]);

  #get the three constants by comparing the output of the matrix
     ↪ linearization to the full discrete time estimate
  matrix_Vout_1 = orig2_Vout + float(numPeriods)*(1.0/fsw)*(a*orig2_Vout +
     ↪ b*orig2_Vcfly + c*orig2_outFB3 + b1_Vin*orig2_Vin + b1_Iout*
     ↪ orig2_Iout );
  matrix_Vcfly_1 = orig2_Vcfly + float(numPeriods)*(1.0/fsw)*(d*orig2_Vout
     ↪ + e*orig2_Vcfly + f*orig2_outFB3 + b2_Vin*orig2_Vin + b2_Iout*
     ↪ orig2_Iout);
  matrix_outFB3_1 = orig2_outFB3 + float(numPeriods)*(1.0/fsw)*(g*
     ↪ orig2_Vout + h*orig2_Vcfly + i*orig2_outFB3 + b3_Vin*orig2_Vin +
     ↪ b3_Iout*orig2_Iout);
  j = (twoP_Vout - matrix_Vout_1)/(1.0/fsw)/float(numPeriods);
  k = (twoP_Vcfly - matrix_Vcfly_1)/(1.0/fsw)/float(numPeriods);
  l = 0;

  B = np.array([[j],[k],[l]]);

  #restore state variables
  Vout = float(orig2_Vout);
  Vcfly = float(orig2_Vcfly);
  outFB3 = float(orig2_outFB3);
  Vin = float(orig2_Vin);
  Iout = float(orig2_Iout);

  return (A, B, B_Vin, B_Iout);
  """

A = getStateSpace([Vout,Vcfly,outFB3],advancePeriod);


#add feedback network to linear model
def addFeedback(A):
  #resize A matrix
  A = np.concatenate((A,np.zeros((3,2))),1);
  A = np.concatenate((A,np.zeros((2,5))),0);
  #add outFB3 terms
  A[2][2] = -1.0/(R4*C3);
  A[2][3] = 1.0/(R4*C3);
  A[2][4] = -(R1+R2)/(10000.0*C3*R2);
  #add outFB2 terms
  A[3][2] = 1.0/(R4*C2);
  A[3][3] = -1.0/(R4*C2);
  #add outFB1 terms
  A[4][0] = ((R3/R1)/(R3/R1+R3/R2+1.0))/(C1*R3) + A[0][0]*((R3/R1+1.0)/(R3/
     ↪ R1+R3/R2+1.0));
  A[4][1] = A[0][1]*((R3/R1+1)/(R3/R1+R3/R2+1.0));
  A[4][2] = A[0][2]*((R3/R1+1)/(R3/R1+R3/R2+1.0));
```

```
  A[4][4] = -((R3/R1+R3/R2)/(R3/R1+R3/R2+1.0))/(C1*R3);

  #add new B matrix for V_set
  #B_Vset = np.zeros((5,1));
  #B_Vset[2][0] = 1.0/(10000.0*C3);

  #add new B matrix for feedback
  B_feedback = np.zeros((5,1));
  B_feedback[4][0] = ((R3/R1)/(R3/R1+R3/R2+1.0))/(C1*R3) + A[0][0]*((R3/R1
     ↪ +1.0)/(R3/R1+R3/R2+1.0));

  return (A,B_feedback)

#print(addFeedback(A));

zoom = 1;
#plot interactive pole zero diagram of system with no feedback
def plotOpenLoop():

  #get state space system with Vout as output
  A = getStateSpace([orig_Vout,orig_Vcfly,orig_outFB3], advancePeriod);
  B = [[0],[0],[0]];
  C = [1.0,0.0,0.0];
  D = 0.0;
  #get poles and zeros
  #sys = control.ss(A,B,C,D);
  #transfunc = control.tf(sys);
  #poles = control.pole(transfunc);
  poles = np.linalg.eig(A)[0];
  #print(poles);
  #zeros = control.zero(transfunc);
  #print(zeros)
  #now plot pole zero map
  fig, ax = plt.subplots()
  plt.subplots_adjust(bottom=0.25)
  l = plt.scatter(np.real(poles), np.imag(poles), s=50, marker='x',
     ↪ edgecolors='b')
  #m = plt.scatter(np.real(zeros), np.imag(zeros), s=50, marker='o',
     ↪ facecolors='none', edgecolors='g')
  plt.axhline(0, color='black')
  plt.axvline(0, color='black')
  plt.grid()
  axcolor = 'lightgoldenrodyellow'
  axI = plt.axes([0.25, 0.1, 0.65, 0.03], facecolor=axcolor)
  axV = plt.axes([0.25, 0.15, 0.65, 0.03], facecolor=axcolor)
  axB = plt.axes([0.05, 0.05, 0.14, 0.1], facecolor=axcolor)
  #add interactive sliders
  sI = Slider(axI, 'Iout', 0.1, 10.0, valinit=orig_Iout)
  sV = Slider(axV, 'Vout', 0.1, 2.5, valinit=orig_Vout)
  sB = RadioButtons(axB, ('Zoom Fit', 'Fix Zoom'))

  def update(val):
    global Iout;
    global Vset;
    global Vout;
    global Vcfly;
```

```
    I = sI.val;
    Iout = float(I);
    V = sV.val;
    Vset = float(V);
    Vout = float(V);
    Vcfly = float(V);
    updateFB3();
    #print(outFB3);
    A = getStateSpace([Vout,Vcfly,outFB3], advancePeriod);
    B = [[0.00001],[0.0],[0.0]];
    C = [1.0,0.0,0.0];
    D = 0;
    #sys = control.ss(A,B,C,D);
    #transfunc = control.tf(sys);
    #print(control.pole(transfunc));
    poles = np.linalg.eig(A)[0];
    #zeros = control.zero(transfunc);
    zeros = [];
    l.set_offsets(np.column_stack((np.real(poles), np.imag(poles))));
    #m.set_offsets(np.column_stack((np.real(zeros), np.imag(zeros))));
    #auto reset axis limits:
    if(zoom==1):
      xmin=min(np.min(np.append(np.real(poles),np.real(zeros))),0,ax.
          ↪ get_xlim()[0]);
      xmax=max(np.max(np.append(np.real(poles),np.real(zeros))),0,ax.
          ↪ get_xlim()[1]);
      ymin=min(np.min(np.append(np.imag(poles),np.imag(zeros))),-0.001,ax.
          ↪ get_ylim()[0]);
      ymax=max(np.max(np.append(np.imag(poles),np.imag(zeros))),0.001,ax.
          ↪ get_ylim()[1]);
      ax.set_xlim(xmin,xmax)
      ax.set_ylim(ymin,ymax)
    plt.draw()
  def button(val):
    global zoom;
    if(val=='Zoom Fit'):
      zoom = 1;
      xmin=min(np.min(np.append(np.real(poles),np.real(zeros))),0,ax.
          ↪ get_xlim()[0]);
      xmax=max(np.max(np.append(np.real(poles),np.real(zeros))),0,ax.
          ↪ get_xlim()[1]);
      ymin=min(np.min(np.append(np.imag(poles),np.imag(zeros))),-0.001,ax.
          ↪ get_ylim()[0]);
      ymax=max(np.max(np.append(np.imag(poles),np.imag(zeros))),0.001,ax.
          ↪ get_ylim()[1]);
      ax.set_xlim(xmin,xmax)
      ax.set_ylim(ymin,ymax)
      plt.draw()
    else:
      zoom = 0;

  sI.on_changed(update);
  sV.on_changed(update);
  sB.on_clicked(button);

  plt.show()
  #return state variables to what they were before
```

```
  Iout = float(orig_Iout);
  Vset = float(orig_Vset);
  Vout = float(orig_Vout);
  Vcfly = float(orig_Vcfly);
  updateFB3();

#plotOpenLoop();
#exit();

#frequency should be in degrees, output is in degrees
def freqResp((A,B,C,D),inputFreqs):
  outputMag = [];
  outputPhase = [];
  for freq in inputFreqs:
    #print(C);
    temp = np.dot(np.dot(C,np.linalg.inv(complex(0,freq*2.0*3.14159)*np.
        ↪ identity(len(A))-A)),B) + D;
    #print(temp);
    #exit();
    #C*(freq*np.identity(len(A))-A)^-1 + D
    #temp2 = temp / (temp - temp/float(np.absolute(temp)));
    #outputMag.append(float(np.absolute(temp))/(float(np.absolute(temp))-1)
        ↪ );
    outputMag.append(float(np.absolute(temp)));
    outputPhase.append(float(np.angle(temp))/2.0/3.14159*360.0);
  return np.array(outputMag), np.array(outputPhase);

#plot interactive pole zero diagram of system with feedback
def plotClosedLoop():
  global Iout;
  global Vset;
  global Vout;
  global Vcfly;
  #get state space system with Vout as output

  (A,B) = addFeedback(A = getStateSpace([orig_Vout,orig_Vcfly,orig_outFB3],
      ↪  advancePeriod));
  C = [1.0,0.0,0.0,0.0,0.0];
  D = 0.0;
  #get poles and zeros
  sys = control.ss(A,B,C,D);
  transfunc = control.tf(sys);
  poles = control.pole(transfunc);
  #poles2 = np.linalg.eig(A)[0];
  zeros = control.zero(transfunc);
  #now plot pole zero map
  fig = plt.figure(1);
  #fig, [ax,ax1] = plt.subplots(ncols=2)
  ax = plt.subplot(121);
  ax1 = plt.subplot(222);
  ax2 = plt.subplot(224);
  plt.subplots_adjust(bottom=0.35)
  l = ax.scatter(np.real(poles), np.imag(poles), s=50, marker='x',
      ↪ edgecolors='b')
  m = ax.scatter(np.real(zeros), np.imag(zeros), s=50, marker='o',
      ↪ facecolors='none', edgecolors='g')
```

```
#q = ax.scatter(np.real(poles2), np.imag(poles2), s=50, marker='o',
    ↪ facecolors='none', edgecolors='r')
#ax.set_xscale('symlog');
ax.axhline(0, color='black')
ax.axvline(0, color='black')
ax.grid()
axcolor = 'lightgoldenrodyellow'
axI = plt.axes([0.3, 0.18, 0.6, 0.03], facecolor=axcolor)
axV = plt.axes([0.3, 0.23, 0.6, 0.03], facecolor=axcolor)
axC1 = plt.axes([0.3, 0.13, 0.6, 0.03], facecolor=axcolor)
axR4 = plt.axes([0.3, 0.08, 0.6, 0.03], facecolor=axcolor)
axC2 = plt.axes([0.3, 0.03, 0.6, 0.03], facecolor=axcolor)
axB = plt.axes([0.05, 0.05, 0.14, 0.1], facecolor=axcolor)
#now plot bode
ax1.grid();
ax2.grid();
freqs = [];
freqs += range(1000,10000,100);
freqs += range(10000,100000,1000);
freqs += range(100000,1000000,10000);
freqs += range(1000000,10000000,100000);
mag, phase = freqResp((A,B,C,D),freqs);

n, = ax1.semilogx(np.array(freqs),np.array(mag).flatten());
o, = ax2.semilogx(np.array(freqs),np.array(phase).flatten());
vert1 = ax1.add_line(lines.Line2D([],[], color='red'));
vert2 = ax2.add_line(lines.Line2D([],[], color='red'));
ax1.set_title("PM: " + "Crossover Freq: ");
#add interactive sliders
sI = Slider(axI, 'Iout', 0.1, 10.0, valinit=orig_Iout,valfmt='%1.1fA')
sV = Slider(axV, 'Vout', 0.1, 2.5, valinit=orig_Vout,valfmt='%1.2fV')
sC1 = Slider(axC1, 'C1', 1.0, 200.0, valinit=1.0,valfmt='%1.0fpF')
sR4 = Slider(axR4, 'R4', 100.0, 5000, valinit=100.0,valfmt='%1.0fOhms')
sC2 = Slider(axC2, 'C2', 0.1, 20.0, valinit=20.0,valfmt='%1.1fnF')
sB = RadioButtons(axB, ('Zoom Fit', 'Fix Zoom'))

zoom==1
def update(val):
  global Iout;
  global Vset;
  global Vout;
  global Vcfly;
  global C1;
  global R4;
  global C2;

  slider_I = sI.val;
  Iout = float(slider_I);
  slider_V = sV.val;
  Vset = float(slider_V);
  Vout = float(slider_V);
  Vcfly = float(slider_V);
  slider_C1 = sC1.val;
  C1 = float(slider_C1*10**-12);
  slider_R4 = sR4.val;
  R4 = float(slider_R4);
  slider_C2 = sC2.val;
```

```
    C2 = float(slider_C2*10**-9);

    #Vout = float(orig_Vout);
    #Vcfly = float(orig_Vcfly);
    updateFB3();
    (A,B) = addFeedback(A = getStateSpace([Vout,Vcfly,outFB3],
        ↪ advancePeriod));
    C = [1.0,0.0,0.0,0.0,0.0];
    D = 0.0;
    sys = control.ss(A,B,C,D);

    #update bode plot
    #mag, phase, omega = control.freqresp(sys, freqs);
    #B = B_Iout;
    #sys = control.ss(A,B,C,D);
    mag, phase = freqResp((A,B,C,D),freqs);
    n.set_ydata(np.array(mag).flatten());
    o.set_ydata(np.array(phase).flatten());
    ax1.set_ylim(np.min(mag),np.max(mag));
    ax2.set_ylim(np.min(phase),np.max(phase));

    """
    gm, pm, Omega_cg, Omega_cp = control.margin(sys);
    if((Omega_cg/2.0/3.14159)>np.min(freqs) and pm!=None):
       print("Phase Margin: " + str(pm));
       print("Crossover Freq :" + str(Omega_cg/2.0/3.14159));
       #print((Iout,Vset,Vout,Vcfly,C1,R4,C2))
       print("");
       ax1.set_title("PM: "+"{:3.1f}".format(pm) + "deg  Crossover Freq: " +
           ↪  "{:6.0f}".format(Omega_cg/2.0/3.14159/1000.0)+"kHz");
    elif(pm==None):
       print ".",;
    """
    #get phase magin from closed loop
    pm = np.min(np.array(phase)[np.array(mag)>=1.0]) - 90.0;
    #pm = 0.0;
    Omega_cg = 0.0;
    ax1.set_title("PM: "+"{:3.1f}".format(pm) + "deg  Crossover Freq: " + "
        ↪ {:6.0f}".format(Omega_cg/2.0/3.14159/1000.0)+"kHz");
    #ax1.add_line(lines.Line2D([Omega_cg/2.0/3.14159,Omega_cg
        ↪ /2.0/3.14159],[9999,-9999], color='red'))
    vert1.set_data([Omega_cg/2.0/3.14159,Omega_cg
        ↪ /2.0/3.14159],[9999,-9999]);
    vert2.set_data([Omega_cg/2.0/3.14159,Omega_cg
        ↪ /2.0/3.14159],[9999,-9999]);

    transfunc = control.tf(sys);
    poles = control.pole(transfunc);
    zeros = control.zero(transfunc);
    #poles2 = np.linalg.eig(A)[0];
    l.set_offsets(np.column_stack((np.real(poles), np.imag(poles))));
    m.set_offsets(np.column_stack((np.real(zeros), np.imag(zeros))));
    #q.set_offsets(np.column_stack((np.real(poles2), np.imag(poles2))));
    #auto reset axis limits:
    if(zoom==1):
       xmin=max(min(np.min(np.append(np.real(poles),np.real(zeros))),0,ax.
           ↪ get_xlim()[0]), -1e12);
```

```
        xmax=min(max(np.max(np.append(np.real(poles),np.real(zeros))),0,ax.
            ↪ get_xlim()[1]), 1e12);
        ymin=min(np.min(np.append(np.imag(poles),np.imag(zeros))),-0.001,ax.
            ↪ get_ylim()[0]);
        ymax=max(np.max(np.append(np.imag(poles),np.imag(zeros))),0.001,ax.
            ↪ get_ylim()[1]);
        ax.set_xlim(xmin,xmax)
        ax.set_ylim(ymin,ymax)
      plt.draw();
  sI.on_changed(update);
  sV.on_changed(update);
  sC1.on_changed(update);
  sR4.on_changed(update);
  sC2.on_changed(update);


  def button(val):
    global zoom;
    if(val=='Zoom Fit'):
      zoom = 1;
      xmin=min(np.min(np.append(np.real(poles),np.real(zeros))),0,ax.
          ↪ get_xlim()[0]);
      xmax=max(np.max(np.append(np.real(poles),np.real(zeros))),0,ax.
          ↪ get_xlim()[1]);
      ymin=min(np.min(np.append(np.imag(poles),np.imag(zeros))),-0.001,ax.
          ↪ get_ylim()[0]);
      ymax=max(np.max(np.append(np.imag(poles),np.imag(zeros))),0.001,ax.
          ↪ get_ylim()[1]);
      ax.set_xlim(xmin,xmax)
      ax.set_ylim(ymin,ymax)
      plt.draw()
    else:
      zoom = 0;
  sB.on_clicked(button);

  plt.show()
  #return state variables to what they were before
  Iout = float(orig_Iout);
  Vset = float(orig_Vset);
  Vout = float(orig_Vout);
  Vcfly = float(orig_Vcfly);
  updateFB3();

plotClosedLoop();


Vout2 = 0*float(Vout);
Vcfly2 = 0*float(Vcfly);
D2 = 0*float(outFB3);
bestR4 = float(R4);
bestC1 = float(C1);
bestC2 = float(C2);
bestCrossOverFreq = 0.0;
bestPhaseMargin = 0.0;

Vout = 1.0
print("Vout,Iout,C1,C2,R4");
```

```
listCurr = np.arange(0.1,10.0,0.1);
for Iout in listCurr:
  bestLoss = 999999999999.0;
  updateFB3();
  #find best controller
  for C1 in np.arange(1e-12, 2e-12, 1e-12):
    for R4 in np.arange(100, 5000, 10):
      for C2 in np.arange(2e-9, 5e-9, 0.1e-9):
        A = getStateSpace([Vout,Vcfly,outFB3],advancePeriod);
        (A,B) = addFeedback(A);

        #check if all eigenvalues have real part less than zero
        eigenValues = np.linalg.eig(A)[0];
        allNeg = np.all(np.real(eigenValues)<0);
        if(allNeg):

          #let's get the phase margin
          #B = B + B_Vin*Vin + B_Iout*Iout + B_Vset*Vset;
          #C = [1.0,0.0,0.0,0.0,0.0];
          #D = 0;
          #sys = control.ss(A,B,C,D);
          #gm, pm, Omega_cg, Omega_cp = control.margin(sys);
          #if((pm>=60.0) and (pm<=70.0) and Omega_cg!=None and Omega_cp!=
              ↪ None):
          # print((gm, pm, Omega_cg/2.0/3.14159, Omega_cp/2.0/3.14159));
          # if((Omega_cg/2.0/3.14159) > bestCrossOverFreq):
          #   bestCrossOverFreq = float(Omega_cg/2.0/3.14159);
          #   bestPhaseMargin = float(pm);
          #   bestR4 = float(R4);
          #   bestC1 = float(C1);
          #   bestC2 = float(C2);
          #pick out eigenValue and eigenVector the correspond most to
              ↪ output
          eigenVectors = np.absolute(np.linalg.eig(A)[1]);
          eigenValues = np.linalg.eig(A)[0];
          slowestIndex = np.argmin(np.absolute(np.real(eigenValues)));
          #print(eigenValues);
          #print(slowestIndex);
          #exit();
          #eigenVectorsMag = np.sum(eigenVectors, axis=0);
          #eigenVectorsPercent = (eigenVectors/eigenVectorsMag)[0];
          #outputVectorIndexMost = np.argmax(eigenVectorsPercent);

          #get metric to rate controller, smaller is better
          #eigenVectors = eigenVectors[eigenVectorsMag<1e9]; #only keep
              ↪ those that affect things above 1us-ish
          #loss = np.max(abs(180.0-np.angle(eigenValues[
              ↪ outputVectorIndexMost], deg=True)))/180.0;
          loss = min(abs(150.0-np.angle(eigenValues[slowestIndex], deg=True
              ↪ )), abs(210.0-np.angle(eigenValues[slowestIndex], deg=True)
              ↪ ));
          #wantedSpeed = 1e9; #50e3;
          #loss += abs(wantedSpeed-np.sum(np.absolute(eigenValues[
              ↪ outputVectorIndexMost])))/wantedSpeed;
          if(loss<bestLoss):
            bestLoss = float(loss);
            bestR4 = float(R4);
```

```
                bestC1 = float(C1);
                bestC2 = float(C2);
    #print("bestPhaseMargin: " + str(bestPhaseMargin));
    #print("bestCrossOverFreq: " + str(bestCrossOverFreq));

     print(str(Vout) + "," + str(Iout) + "," + str(bestC1) + "," + str(bestC2)
        ↪  + "," + str(bestR4));
#print("C1: " + str(bestC1));
#print("C2: " + str(bestC2));
#print("R4: " + str(bestR4));

exit();

R4 = float(bestR4);
C1 = float(bestC1);
C2 = float(bestC2);



A = getStateSpace([Vout,Vcfly,outFB3],advancePeriod);

eigenValues = np.linalg.eig(A)[0];
eigenVectors = np.linalg.eig(A)[1];
print("eigenValues: " + str(eigenValues));
print("eigenVectors: " + str(np.absolute(eigenVectors)));




(A,B) = addFeedback(A);

#print("A:" + str(A));
#print("B:" + str(B + B_Vin*Vin + B_Iout*Iout + B_Vset*Vset));
#print("B_Vin:" + str(B_Vin));
#print("B_Iout:" + str(B_Iout));
#print("B_Vset:" + str(B_Vset));



eigenValues = np.linalg.eig(A)[0];
eigenVectors = np.linalg.eig(A)[1];
print("eigenValues: " + str(eigenValues));
print("eigenValueAngles: " + str(np.angle(eigenValues, deg=True)));
print("eigenVectors: " + str(np.absolute(eigenVectors)));
#exit();

def matrixAdvancePeriod():
  global Vout2;
  global Vcfly2;
  input1 = np.array([[Vout2],[Vcfly2],[outFB3]]);
  output = input1 + 1.0/fsw*(np.dot(A,input1));

  Vout2 = float(output[0][0]);
  Vcfly2 = float(output[1][0]);


def matrixAdvancePeriodFeedback():
```

```
  global Vout2;
  global Vcfly2;
  global outFB3;
  global outFB2;
  global outFB1;
  global A,B;
  input1 = np.array([[Vout2],[Vcfly2],[outFB3],[outFB2],[outFB1]]);
  #print(input1);
  for p in range(1000):
    input1 = input1 + 1.0/fsw/1000.0*(np.dot(A,input1) + B*Vset);
  #print(input1)
  Vout2 = float(input1[0][0]);
  Vcfly2 = float(input1[1][0]);
  outFB3 = float(input1[2][0]);
  outFB2 = float(input1[3][0]);
  outFB1 = float(input1[4][0]);


  #(A,B,B_Vin,B_Iout) = getStateSpace();
  #(A,B,B_Vin,B_Iout,B_Vset) = addFeedback(A,B,B_Vin,B_Iout);


for p in range(200):

  #if((p*1.0/fsw)>100e-6 and (p*1.0/fsw)<200e-6):
  # D = 0.045;
  #else:
  # D = 0.040;

  #if((p*1.0/fsw)>300e-6 and (p*1.0/fsw)<400e-6):
  # Iout = 1.5*orig_Iout;
  #else:
  # Iout = orig_Iout;

  if((p*1.0/fsw)>50e-6 and (p*1.0/fsw)<400e-6):
    Vset = 0.1;
  else:
    Vset = 0.0;
  """
  try:
    [Vout,Vcfly,outFB3] = advancePeriod([Vout,Vcfly,outFB3]);
  except:
    print([Vout,Vcfly,outFB3]);
    exit();
  outFB3_t.append(outFB3);
  Vfly_t.append(Vcfly);
  Vout_t.append(Vout);
  """

  matrixAdvancePeriodFeedback();
  Vfly_t2.append(Vcfly2);
  Vout_t2.append(Vout2);
  outFB3_t2.append(outFB3);
  outFB2_t2.append(outFB2);
  outFB1_t2.append(outFB1);
```

```
    t.append((p+1)*1.0/fsw);



#plot voltages
#a = graph.grapher([t,t,t,t,t,t,t], [Vout_t,Vfly_t,Vout2_t,Vfly2_t,outFB3_t
    ↪ ,outFB2_t,outFB1_t], ["Vout","Vfly","Vout_matrix","Vfly_matrix","
    ↪ outFB3","outFB2","outFB1"], "time", "voltage");
a = graph.grapher([t,t,t,t,t], [Vout_t2,Vfly_t2,outFB3_t2,outFB2_t2,
    ↪ outFB1_t2], ["Vout_matrix","Vfly_matrix","outFB3","outFB2","outFB1"],
    ↪  "time", "voltage");
graph.plt.grid();
a.plot();


#now plot pole zero map
import control;
import matplotlib.pyplot as plt;
print(A);
A[2][0] = 0;
A[2][1] = 0;
A[2][2] = 0;
A[2][3] = 0;
A[2][4] = 0;
print(A);
B = B + B_Vin*Vin + B_Iout*Iout + B_Vset*Vset;
C = [1.0,0.0,0.0,0.0,0.0];
D = 0.0;
sys = control.ss(A,B,C,D);
transfunc = control.tf(sys);
control.matlab.pzmap(transfunc);
zeros = control.zero(transfunc);
plt.scatter(np.real(zeros), np.imag(zeros), s=50, marker='o',facecolors='
    ↪ none', edgecolors='g')
plt.show();

mag, phase, omega = control.matlab.bode(sys,range(1000,1000000,1000),dB=
    ↪ True,deg=True,Plot=False)
#print(mag)
plt.plot(np.divide(omega,(2*3.14159)),mag);
plt.plot(np.divide(omega,(2*3.14159)),phase);
plt.show();
```

C.5 Simplified Symbolic Efficiency Estimation

```
from __future__ import print_function
from sympy import *;
import itertools;
import random;
from si_prefix import si_format;
```

```
#names and units of inputs
names = [];
units = [];
names.append('fsw');units.append("Hz");
names.append('Vout');units.append("V");
names.append('Vin');units.append("V");
names.append('Iout');units.append("A");
names.append('L');units.append("H");
names.append('C');units.append("F");
names.append('Lres');units.append("H");
names.append('FOM');units.append("");
names.append('RoverL');units.append("");
names.append('CtimesR');units.append("");
names.append('RtimesW');units.append("");
names.append('CtimesW');units.append("");
names.append('W1');units.append("m");
names.append('W2');units.append("m");
names.append('W3');units.append("m");
names.append('W4');units.append("m");

#create the sympy variables that will be used as symbols in equations
fsw,Vout,Vin,Iout,L,C,Lres,FOM,RoverL,CtimesR,RtimesW,CoverW,W1,W2,W3,W4 =
    ↪ symbols(names[0:16]);


#ResSC
def loss_ResSC(returnAll=False):
  sw1_RMS = 2**0.75*3**-0.5 * (Vout*Iout/Vin)**0.75 * ((Vin-2*Vout)/L)
      ↪ **0.25 * (fsw)**-0.25;
  sw1_loss = sw1_RMS**2 * RtimesW/W1;
  sw2_RMS = L**-0.25 * (2.0**1.5/3.0*fsw**-0.5*(Vin-2*Vout)**1.5*(Iout/Vin)
      ↪ **1.5*Vout**0.5 + float(pi)/8.0*fsw**-1*(Iout/Vin)**2*(Vin-Vout)
      ↪ **2*C**-0.5)**0.5;
  sw2_loss = sw2_RMS**2 * RtimesW/W2;
  sw3_RMS = 2**0.75*3**-0.5 * (Vout*Iout/Vin)**0.75 * ((Vin-2*Vout)/L)
      ↪ **0.25 * ((Vin-Vout)/Vout)**0.5 * (fsw)**-0.25;
  sw3_loss = sw3_RMS**2 * RtimesW/W3;
  sw4_RMS = float(pi)**0.5*2.0**-1.5*(Vout*Iout/Vin)*((Vin-Vout)/Vout)*L
      ↪ **-0.25*C**-0.25*fsw**-0.5;
  sw4_loss = sw4_RMS**2 * RtimesW/W4;
  L_RMS = L**-0.25 * (2.0**1.5/3.0*fsw**-0.5*(Vin-2*Vout)**0.5*(Iout/Vin)
      ↪ **1.5*(Vin-Vout) + float(pi)/8.0*fsw**-1*(Iout/Vin)**2*(Vin-Vout)
      ↪ **2*C**-0.5)**0.5
  L_loss = L_RMS**2 * L * RoverL;
  C_RMS = L_RMS;
  C_loss = C_RMS**2 * CtimesR / C;
  switching_loss = CoverW*Vin**2*fsw*(W1+W2+W3+W4);

  #a = 1.0/(2.0*float(pi)*(L*C)**0.5)
  #valid_loss = Vin*Iout*(abs(a-fsw)-(a-fsw))
  t1 = (Iout*fsw**-1*(Vin/Vout)**-1*((Vin-2.0*Vout)/(2*L))**-1)**0.5;
  t2 = (Vin-2.0*Vout)/Vout*t1+t1;
  t3 = float(pi)*(L*C)**0.5+t2;
  valid_time_loss = 9999.0/t3*Vin*Iout*(abs(1.0/fsw-t3)-(1.0/fsw-t3));
  #c1 = Iout*(Vin-Vout)/(2.0*C*fsw*Vin*Vout);
  #c2 = Iout*(Vin-Vout)/(2.0*C*fsw*(Vin-2*Vout));
  c1 = Iout/(2.0*C*fsw);
```

```
  #valid_cap_loss = 999999.0*Vin*Iout*(abs(0.1-c1)-(0.1-c1)) + 999999.0*Vin
      ↪ *Iout*(abs(0.1-c2)-(0.1-c2));
  valid_cap_loss = 999999.0*Vin*Iout*(abs(0.5-c1)-(0.5-c1));

  if(returnAll):
    return [sw1_loss,sw2_loss,sw3_loss,sw4_loss,L_loss,C_loss,
        ↪ switching_loss,valid_time_loss,valid_cap_loss]
  else:
    return Vout*Iout/((sw1_loss+sw2_loss+sw3_loss+sw4_loss+L_loss+C_loss+
        ↪ switching_loss+valid_time_loss+valid_cap_loss) + Vout*Iout);
  #return sw1_loss+sw2_loss+sw3_loss+sw4_loss+L_loss+C_loss+switching_loss;

#3-Level Buck
def loss_3LevBuck(returnAll=False):
  sw1_RMS = 2**0.5*3**-0.5 * (Vout*Iout/Vin)**0.75 * ((Vin-2*Vout)/L)**0.25
      ↪  * (fsw)**-0.25;
  sw1_loss = sw1_RMS**2 * RtimesW/W1;
  sw2_RMS = sw1_RMS;
  sw2_loss = sw2_RMS**2 * RtimesW/W2;
  sw3_RMS = 2**0.5*3**-0.5 * (Vout*Iout/Vin)**0.75 * ((Vin-2*Vout)/L)**0.25
      ↪  * ((Vin-Vout)/Vout)**0.5 * (fsw)**-0.25;
  sw3_loss = sw3_RMS**2 * RtimesW/W3;
  sw4_RMS = sw3_RMS;
  sw4_loss = sw4_RMS**2 * RtimesW/W4;
  L_RMS = 2**0.5*3**-0.5 * (Iout)**0.75 * ((Vin-2*Vout)/L)**0.25 * (Vout/
      ↪ Vin)**0.25 * (fsw)**-0.25;
  L_loss = L_RMS**2 * L * RoverL;
  C_RMS = 2*3**-0.5 * (Vout*Iout/Vin)**0.75 * ((Vin-2*Vout)/L)**0.25 * (fsw
      ↪ )**-0.25;
  C_loss = C_RMS**2 * CtimesR / C;
  switching_loss = CoverW*Vin**2*fsw*(W1+W2+W3+W4);

  t1 = (Iout*fsw**-1*(Vin/Vout)**-1*((Vin-2.0*Vout)/(4*L))**-1)**0.5;
  t2 = (Vin-2.0*Vout)/(2.0*Vout)*t1+t1;
  valid_time_loss = 9999.0/t2*Vin*Iout*(abs(0.5/fsw-t2)-(0.5/fsw-t2));
  c1 = Iout*Vout/(C*fsw*Vin**2);
  c2 = Iout*Vout/(C*fsw*Vin*(Vin-2*Vout));
  valid_cap_loss = 999999.0*Vin*Iout*(abs(0.1-c1)-(0.1-c1)) + 999999.0*Vin*
      ↪ Iout*(abs(0.1-c2)-(0.1-c2));

  if(returnAll):
    return [sw1_loss,sw2_loss,sw3_loss,sw4_loss,L_loss,C_loss,
        ↪ switching_loss,valid_time_loss,valid_cap_loss]
  else:
    return Vout*Iout/((sw1_loss+sw2_loss+sw3_loss+sw4_loss+L_loss+C_loss+
        ↪ switching_loss+valid_time_loss+valid_cap_loss) + Vout*Iout);

#ResCuk
def loss_ResCuk(returnAll=False):
  sw1_RMS = 2**0.75*3**-0.5 * (Vout*Iout/Vin)**0.75 * ((Vin-2*Vout)/L)
      ↪ **0.25 * (fsw)**-0.25;
  sw1_loss = sw1_RMS**2 * RtimesW/W1;
  sw2_RMS = float(pi)**0.5*8.0**-0.5*(Vout*Iout/Vin)*(Lres*C)**-0.25*(fsw)
      ↪ **-0.5#TODO SOMETHING IS WRONG WITH THIS
  sw2_loss = sw2_RMS**2 * RtimesW/W2;
  sw3_RMS = 2**0.75*3**-0.5 * (Vout*Iout/Vin)**0.75 * ((Vin-2*Vout)/L)
      ↪ **0.25 * ((Vin-Vout)/Vout)**0.5 * (fsw)**-0.25;
```

```
    sw3_loss = sw3_RMS**2 * RtimesW/W3;
    sw4_RMS = (2.0**1.5*3.0**-1*((Vin-2*Vout)/L)**0.5*(Vout*Iout/Vin)**1.5*((
        ↪ Vin-2*Vout)/Vout)*(fsw)**-0.5 + float(pi)*8**-1*(Vout*Iout/Vin)
        ↪ **2*(Lres*C)**-0.5*(fsw)**-1)**0.5
    sw4_loss = sw4_RMS**2 * RtimesW/W4;
    L_RMS = sw3_RMS;
    L_loss = L_RMS**2 * L * RoverL;
    Lres_RMS = sw2_RMS;
    Lres_loss = Lres_RMS**2 * Lres * RoverL;
    C_RMS = (2.0**1.5*3.0**-1*((Vin-2*Vout)/L)**0.5*(Vout*Iout/Vin)**1.5*(fsw
        ↪ )**-0.5 + float(pi)*8**-1*(Vout*Iout/Vin)**2*(Lres*C)**-0.5*(fsw)
        ↪ **-1)**0.5
    C_loss = C_RMS**2 * CtimesR / C;
    switching_loss = CoverW*Vin**2*fsw*(W1+W2+W3+W4);

    t1 = (Iout*fsw**-1*(Vin/Vout)**-1*((Vin-2.0*Vout)/(2*L))**-1)**0.5;
    t2 = (Vin-2.0*Vout)/Vout*t1+t1;
    t3 = float(pi)*(Lres*C)**0.5+t1;
    valid_time_loss = 9999.0/t3*Vin*Iout*(abs(1.0/fsw-t3)-(1.0/fsw-t3));
    c1 = Iout/(2.0*C*fsw*Vin);
    c2 = Iout*Vout/(2.0*C*fsw*Vin*(Vin-2*Vout));
    valid_cap_loss = 999999.0*Vin*Iout*(abs(0.1-c1)-(0.1-c1)) + 999999.0*Vin*
        ↪ Iout*(abs(0.1-c2)-(0.1-c2));

    if(returnAll):
      return [sw1_loss,sw2_loss,sw3_loss,sw4_loss,L_loss,Lres_loss,C_loss,
          ↪ switching_loss,valid_time_loss,valid_cap_loss]
    else:
      return Vout*Iout/((sw1_loss+sw2_loss+sw3_loss+sw4_loss+L_loss+Lres_loss
          ↪ +C_loss+switching_loss+valid_time_loss+valid_cap_loss) + Vout*
          ↪ Iout);

#create loss functions in numpy with vector inputs to speed up
import numpy as np;
np_loss_ResSC = lambdify((fsw, Vout, Vin, Iout, L, C, Lres, FOM, RoverL,
    ↪ CtimesR, RtimesW, CoverW, W1, W2, W3, W4), loss_ResSC(), "numpy")
np_loss_3LevBuck = lambdify((fsw, Vout, Vin, Iout, L, C, Lres, FOM, RoverL,
    ↪ CtimesR, RtimesW, CoverW, W1, W2, W3, W4), loss_3LevBuck(), "numpy")
np_loss_ResCuk = lambdify((fsw, Vout, Vin, Iout, L, C, Lres, FOM, RoverL,
    ↪ CtimesR, RtimesW, CoverW, W1, W2, W3, W4), loss_ResCuk(), "numpy")
#print(np_loss_ResCuk(2e6,1,5,5,2e-9,2e-6,0.5e-9,0,4e-3/2e-9,2e-6*4e-3,4e
    ↪ -3*0.5,2e-9/0.5,1,1,1,1));
#exit();
#print(latex(loss_ResSC()))
#print("");
#print(latex(loss_ResSC().expand()))
#print("");
globalMax = [0,0,0];
inputsMax = [[None,None,None,None,None,None,None,None,None,None,None,None,
    ↪ None,None,None,None],[None,None,None,None,None,None,None,None,None,
    ↪ None,None,None,None,None,None,None],[None,None,None,None,None,None,
    ↪ None,None,None,None,None,None,None,None,None,None]];
while(True):
  inputs = [[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]];
  for combo in range(100000):
    inputs[0].append(random.uniform(1.0e6,10e6));#fsw
    inputs[1].append(1.0);#Vout
```

```
    inputs[2].append(5.0);#Vin
    inputs[3].append(5.0);#Iout
    inputs[4].append(random.uniform(2e-9,10e-9));#L
    inputs[5].append(7.04e-6);#C
    inputs[6].append(random.uniform(0.1e-9,10e-9));#Lres
    inputs[7].append(0);#FOM
    inputs[8].append(4e-3/2e-9); #RoverL
    inputs[9].append(7.04e-6*0.002513);#CtimesR
    inputs[10].append(4e-3*0.5); #RtimesW
    inputs[11].append(2e-9/0.5); #CoverW
    inputs[12].append(random.uniform(0.5,1.5));#W1
    inputs[13].append(random.uniform(0.5,1.5));#W2
    inputs[14].append(random.uniform(0.5,1.5));#W3
    inputs[15].append(random.uniform(0.5,1.5));#W4

  a0 = np_loss_ResSC(*inputs);
  a1 = np_loss_3LevBuck(*inputs);
  a2 = np_loss_ResCuk(*inputs);
  b0 = np.argmax(a0);
  b1 = np.argmax(a1);
  b2 = np.argmax(a2);

  #if something better has been found
  if(a0[b0]>globalMax[0] or a1[b1]>globalMax[1] or a2[b2]>globalMax[2]):
    print("\n");
    if(a0[b0]>globalMax[0]):
      print("ResSC*".ljust(25,' '),end="")
      globalMax[0] = a0[b0];
      for x in range(len(inputs)):
        inputsMax[0][x] = inputs[x][b0];
    else:
      print("ResSC".ljust(25,' '),end="")
    if(a1[b1]>globalMax[1]):
      print("3LevBuck*".ljust(25,' '),end="")
      globalMax[1] = a1[b1];
      for x in range(len(inputs)):
        inputsMax[1][x] = inputs[x][b1];
    else:
      print("3LevBuck".ljust(25,' '),end="")
    if(a2[b2]>globalMax[2]):
      print("ResCuk*")
      globalMax[2] = a2[b2];
      for x in range(len(inputs)):
        inputsMax[2][x] = inputs[x][b2];
    else:
      print("ResCuk")
    #print output to console

    #print("ResSC".ljust(25,' ')+"3LevBuck".ljust(25,' ')+"ResCuk")
    print(("Eff".ljust(10, '.') + "{0:.3f}".format(globalMax[0])).ljust(25,
        ↪ ' ') + ("Eff".ljust(10, '.') + "{0:.3f}".format(globalMax[1])).
        ↪ ljust(25,' ') + "Eff".ljust(10, '.') + "{0:.3f}".format(globalMax
        ↪ [2]));
    for x in range(len(names)):
      print((names[x].ljust(10,'.') + si_format(inputsMax[0][x],3)+units[x
          ↪ ]).ljust(25,' ') + (names[x].ljust(10,'.') + si_format(
          ↪ inputsMax[1][x],3)+units[x]).ljust(25,' ') + names[x].ljust(10,
```

```
        ↪ '.') + si_format(inputsMax[2][x],3)+units[x]);
  else:
    print(".",end='');

"""
  if(a[b]>globalMax):
    globalMax = a[b];
    print("\n");
    print("Eff".ljust(10, '.') + "{0:.3f}".format(a[b]));
    for x in range(len(names)):
      print(names[x].ljust(10,'.') + si_format(inputs[x][b],3)+units[x]);
  """

import graph;
b = graph.grapher([range(len(a))], [a], [""]);
#b.plot();
```

C.6 PCB uC Code

```
#include "project.h"
#include <stdio.h>

void configTri(int mVpeak)
{
    Control_Reg_OutputEnable_Write(0);
    AMux_1_FastSelect(1);

    //16 bit adc is 65536
    //so 0V is 32768 and 6.144V is 65536
    //block until input voltge is above 4.6V
    ADC_DelSig_1_StartConvert();
    CyDelay(100);
    int mVinput = (ADC_DelSig_1_Read16()-32768)*6144/32768;
    while(mVinput<4600)
    {
        ADC_DelSig_1_StartConvert();
        CyDelay(100);
        mVinput = ADC_DelSig_1_CountsTo_mVolts(ADC_DelSig_1_GetResult32());
        //Led_pin_Write(1);
    }
    //Led_pin_Write(0);

    //start triangle wave current sources
    IDAC8_1_Start();
    IDAC8_2_Start();

    int maxValue1 = 255;
    int minValue1 = 0;
    int maxValue2 = 255;
    int minValue2 = 0;

    //set the reference at the peak that we want, max of 4.080V
```

```
    if(mVpeak<0)
    {
        mVpeak = 0;
    }
    if(mVpeak>4080)
    {
        mVpeak = 4080;
    }
    VDAC8_1_SetValue(mVpeak/16);

    //figure out what the current values should be to have a peak of mVpeak
    //triangle wave 1
    int currentVal = (minValue1+maxValue1)/2;
    while((minValue1!=currentVal) & (maxValue1!=currentVal))
    {
        IDAC8_1_SetValue(currentVal);
        //clear the sample bit
        CyDelayUs(1000);
        Status_Reg_1_Read();
        CyDelayUs(1000);
        //see if the bit has been set
        if(Status_Reg_1_Read()==0)
        {
            maxValue1 = currentVal;
        }
        else
        {
            minValue1 = currentVal;
        }
        currentVal = (minValue1+maxValue1)/2;
    }
    //triangle wave 2
    currentVal = (minValue1+maxValue1)/2;
    while((minValue2!=currentVal) & (maxValue2!=currentVal))
    {
        IDAC8_2_SetValue(currentVal);
        //clear the sample bit
        CyDelayUs(1000);
        Status_Reg_2_Read();
        CyDelayUs(1000);
        //see if the bit has been set
        if(Status_Reg_2_Read()==0)
        {
            maxValue2 = currentVal;
        }
        else
        {
            minValue2 = currentVal;
        }
        currentVal = (minValue1+maxValue1)/2;
    }

    AMux_1_FastSelect(0);
    Control_Reg_OutputEnable_Write(1);
    return;
}
```

```
uint8 DAC_setOut;
uint8 DAC_setFly = 2500/16/2;
void setOutput(int mVout)
{
    //Control_Reg_OutputEnable_Write(0);

    //set the output to what we want, max of 4.080V
    if(mVout<0)
    {
        mVout = 0;
    }
    if(mVout>4080)
    {
        mVout = 4080;
    }
    //VDAC8_3_SetValue(mVout/16/2);
    DAC_setOut = mVout/16/2;
    //CyDelay(100);


    //Control_Reg_OutputEnable_Write(1);
}

void clearScreen()
{
    while(!USBUART_1_CDCIsReady());
    USBUART_1_PutChar(27);
    while(!USBUART_1_CDCIsReady());
    USBUART_1_PutString("[2J");
    while(!USBUART_1_CDCIsReady());
    USBUART_1_PutChar(27);
    while(!USBUART_1_CDCIsReady());
    USBUART_1_PutString("[H");
}

unsigned int currentR = 0;
void setResistorI2C(unsigned int Ohms)
{
    //full range is 20kOhms with 1024 steps
    if(Ohms>20000)
    {
        Ohms = 20000;
    }
    unsigned int steps = Ohms*1023;
    steps = steps/20000;
    currentR = steps*20000;
    currentR = currentR/1023;
    //send new command to digital potentiometer
    uint8 address = 0b00101111;
    //data bytes are 0-0-C3-C2-C1-C0-D9-D8 then D7-D6-D5-D4-D3-D2-D1-D0
    //enable updating of RDAC register
    uint8 toWrite[2] = {0b00011100,0b00000010};
    I2C_1_MasterWriteBuf(address, toWrite, 2, I2C_1_MODE_COMPLETE_XFER);
    CyDelay(1);
    //update the RDAC register
    toWrite[0] = 0b00000100 + ((steps&(1<<9))>>8) + ((steps&(1<<8))>>8);
    toWrite[1] = 0b00000000 + (steps&(0b11111111));
```

```
    I2C_1_MasterWriteBuf(address, toWrite, 2, I2C_1_MODE_COMPLETE_XFER);
    CyDelay(1);
}


//list of soldered capacitor values in pF
//unsigned int attachedC[8] = {12800,6400,3200,1600,800,400,200,100};
unsigned int attachedC[8] = {100000,10000,4700,2200,1000,470,220,100};
unsigned int currentC = 0;
void setCapacitorI2C(unsigned int pFset) //not done yet, not tested
{
    unsigned char toConnect[1] = {0b00000000};
    //see what capacitors should be connected
    //attachedC[i] must have decreasing values with decreasing i
    currentC = 0;
    for(int i=0; i<8; i++)
    {
        if(attachedC[i]<pFset)
        {
            toConnect[0] = toConnect[0] | (1<<i);
            pFset = pFset - attachedC[i];
            currentC += attachedC[i];
        }
    }
    //toConnect[0] = 0b11111111;

    //send the result to the analog switch
    uint8 address = 0b1001000;
    I2C_1_MasterWriteBuf(address, toConnect, 1, I2C_1_MODE_COMPLETE_XFER);
    CyDelay(1);
}

char menuStrings[10][30] = {"Set Triangle Wave Height","Set Output Voltage"
    ↪ ,"Set I2C Resistor","Set I2C capacitor","Set Cfly Voltage","testC","
    ↪ testD","testE","testF","testG"};
//char menu[10][24] = {"TestA1","TestA2","TestA3","testA","testB","testC","
    ↪ testD","testE","testF","testG"};
char dts[10] = {'0','1','2','3','4','5','6','7','8','9'};
int menu = -1;
unsigned int inputValue = 0;
//int inputValuePlace = 0;
void drawMenu()
{
    clearScreen();
    while(!USBUART_1_CDCIsReady());
    USBUART_1_PutString("Press m to escape and return to\r\n");
    while(!USBUART_1_CDCIsReady());
    USBUART_1_PutString("the main menu at any time.\r\n");

    //output voltage
    CyDelayUs(100);
    int counts1 = ADC_SAR_1_GetResult16();
    CyDelayUs(100);
    int counts2 = ADC_SAR_1_GetResult16();
    CyDelayUs(100);
    int counts3 = ADC_SAR_1_GetResult16();
    CyDelayUs(100);
```

```
    int counts4 = ADC_SAR_1_GetResult16();
    CyDelayUs(100);
    int counts5 = ADC_SAR_1_GetResult16();
    CyDelayUs(100);
    int counts6 = ADC_SAR_1_GetResult16();
    CyDelayUs(100);
    int counts7 = ADC_SAR_1_GetResult16();
    CyDelayUs(100);
    int counts8 = ADC_SAR_1_GetResult16();
    int readV = 2*ADC_SAR_1_CountsTo_mVolts((counts1+counts2+counts3+
        ↪ counts4+counts5+counts6+counts7+counts8)/8);
    char toPrint[20];
    sprintf(toPrint,"%d", readV);
    while(!USBUART_1_CDCIsReady());
    USBUART_1_PutString("Output/set (mV): ");
    while(!USBUART_1_CDCIsReady());
    USBUART_1_PutString(toPrint);
    while(!USBUART_1_CDCIsReady());
    USBUART_1_PutString("/");
    sprintf(toPrint,"%d", DAC_setOut*32);
    while(!USBUART_1_CDCIsReady());
    USBUART_1_PutString(toPrint);
    //flying cap voltage
    CyDelayUs(100);
    counts1 = ADC_SAR_2_GetResult16();
    CyDelayUs(100);
    counts2 = ADC_SAR_2_GetResult16();
    CyDelayUs(100);
    counts3 = ADC_SAR_2_GetResult16();
    CyDelayUs(100);
    counts4 = ADC_SAR_2_GetResult16();
    CyDelayUs(100);
    counts5 = ADC_SAR_2_GetResult16();
    CyDelayUs(100);
    counts6 = ADC_SAR_2_GetResult16();
    CyDelayUs(100);
    counts7 = ADC_SAR_2_GetResult16();
    CyDelayUs(100);
    counts8 = ADC_SAR_2_GetResult16();
    readV = 2*ADC_SAR_2_CountsTo_mVolts((counts1+counts2+counts3+counts4+
        ↪ counts5+counts6+counts7+counts8)/8);
    sprintf(toPrint,"%d", readV);
    while(!USBUART_1_CDCIsReady());
    USBUART_1_PutString("\r\nFlying Cap (mV): ");
    while(!USBUART_1_CDCIsReady());
    USBUART_1_PutString(toPrint);
    //feedback resistor
    sprintf(toPrint,"%d", currentR);
    while(!USBUART_1_CDCIsReady());
    USBUART_1_PutString("\r\nFeedback Resistance (Ohms): ");
    while(!USBUART_1_CDCIsReady());
    USBUART_1_PutString(toPrint);
    //feedback capacitance
    sprintf(toPrint,"%d", currentC);
    while(!USBUART_1_CDCIsReady());
    USBUART_1_PutString("\r\nFeedback Capacitance (pF): ");
```

```
    while(!USBUART_1_CDCIsReady());
    USBUART_1_PutString(toPrint);
    //die temperature
    int16_t dieTemp;
    DieTemp_1_GetTemp(&dieTemp);
    sprintf(toPrint,"%d",dieTemp);
    while(!USBUART_1_CDCIsReady());
    USBUART_1_PutString("\r\nDie Temperature (degC): ");
    while(!USBUART_1_CDCIsReady());
    USBUART_1_PutString(toPrint);
    while(!USBUART_1_CDCIsReady());
    USBUART_1_PutString("\r\n\r\n\r\n");

    if(menu==-1)
    {
        while(!USBUART_1_CDCIsReady());
        USBUART_1_PutString("----------------------------------------\r\n");
        while(!USBUART_1_CDCIsReady());
        USBUART_1_PutString("---------- Main Menu ----------\r\n");
        while(!USBUART_1_CDCIsReady());
        USBUART_1_PutString("----------------------------------------\r\n");
        for(uint8 i=0; i<10; i++)
        {
            while(!USBUART_1_CDCIsReady());
            USBUART_1_PutChar(dts[i]);
            while(!USBUART_1_CDCIsReady());
            USBUART_1_PutString(" - ");
            while(!USBUART_1_CDCIsReady());
            USBUART_1_PutString(menuStrings[i]);
            while(!USBUART_1_CDCIsReady());
            USBUART_1_PutString("\r\n");
        }
        while(!USBUART_1_CDCIsReady());
        USBUART_1_PutString("----------------------------------------\r\n");
    }
    else if(menu==0)
    {
        while(!USBUART_1_CDCIsReady());
        USBUART_1_PutChar(dts[menu]);
        while(!USBUART_1_CDCIsReady());
        USBUART_1_PutString(" - ");
        while(!USBUART_1_CDCIsReady());
        USBUART_1_PutString(menuStrings[menu]);
        while(!USBUART_1_CDCIsReady());
        USBUART_1_PutString("\r\n----------------------------------------\r\n");
        while(!USBUART_1_CDCIsReady());
        USBUART_1_PutString("Maximum is 4080(mV)\n\r");
        while(!USBUART_1_CDCIsReady());
        USBUART_1_PutString("mVpeak: ");
    }
    else if(menu==1)
    {
        while(!USBUART_1_CDCIsReady());
        USBUART_1_PutChar(dts[menu]);
        while(!USBUART_1_CDCIsReady());
        USBUART_1_PutString(" - ");
```

```
            while(!USBUART_1_CDCIsReady());
            USBUART_1_PutString(menuStrings[menu]);
            while(!USBUART_1_CDCIsReady());
            USBUART_1_PutString("\r\n----------\r\n");
            while(!USBUART_1_CDCIsReady());
            USBUART_1_PutString("Output Voltage (mV): ");
        }
        //else if(menu==4)
        //{
        //    while(!USBUART_1_CDCIsReady());
        //    USBUART_1_PutChar(dts[menu]);
        //    while(!USBUART_1_CDCIsReady());
        //    USBUART_1_PutString(" - ");
        //    while(!USBUART_1_CDCIsReady());
        //    USBUART_1_PutString(menuStrings[menu]);
        //    while(!USBUART_1_CDCIsReady());
        //    USBUART_1_PutString("\r\n----------\r\n");


        //}
}



int timer = 0;
void checkUSBSerial()
{
    timer += 1;
    if(timer>100)
    {
        if((menu==-1) | (menu==4))
        {
            drawMenu();
        }
        timer = 0;
    }

    //see if there is any incoming usb data
    int count = USBUART_1_DataIsReady();
    if(count!=0)
    {
        uint8 buffer[64];
        int len = USBUART_1_GetAll(buffer);

        for(int i=0; i<len; i++)
        {
            while(!USBUART_1_CDCIsReady());
            USBUART_1_PutChar(buffer[i]);


            if(buffer[i]=='m')
            {
                menu = -1;
                //keyPressed = 0;
            }
            if(menu==-1)
            {
```

```
            if(buffer[i]=='0')
            {
                menu = 0;
            }
            else if(buffer[i]=='1')
            {
                menu = 1;
            }
            else if(buffer[i]=='2')
            {
                menu = 2;
            }
            else if(buffer[i]=='3')
            {
                menu = 3;
            }
            else if(buffer[i]=='4')
            {
                menu = 4;
            }
            else if(buffer[i]=='5')
            {
                menu = 5;
            }
            else if(buffer[i]=='6')
            {
                menu = 6;
            }
            else if(buffer[i]=='7')
            {
                menu = 7;
            }
            else if(buffer[i]=='8')
            {
                menu = 8;
            }
            else if(buffer[i]=='9')
            {
                menu = 9;
            }
            drawMenu();
            inputValue = 0;
        }
        else if((menu==0) | (menu==1) | (menu==2) | (menu==3) | (menu
            ↪ ==4))
        {
            if(buffer[i]=='\r')
            {
                if(menu==0)
                {
                    if(inputValue>5000)
                    {
                        inputValue = 5000;
                    }
                    configTri(inputValue);
                }
                else if(menu==1)
```

```
            {
                if(inputValue>5000)
                {
                    inputValue = 5000;
                }
                setOutput(inputValue);
            }
            else if(menu==2)
            {
                setResistorI2C(inputValue);
            }
            else if(menu==3)
            {
                setCapacitorI2C(inputValue);
            }

            inputValue = 0;
            menu = -1;
            drawMenu();
        }
        else if((menu==1) & (buffer[i]=='w'))
        {
            //VDAC8_3_SetValue(VDAC8_3_Data+1);
            DAC_setOut = DAC_setOut + 1;
            drawMenu();
            inputValue = 0;
        }
        else if((menu==1) & (buffer[i]=='s'))
        {
            //VDAC8_3_SetValue(VDAC8_3_Data-1);
            DAC_setOut = DAC_setOut - 1;
            drawMenu();
            inputValue = 0;
        }
        else if((menu==4) & (buffer[i]=='w'))
        {
            //VDAC8_3_SetValue(VDAC8_3_Data+1);
            if(DAC_setFly<255)
            {
                DAC_setFly = DAC_setFly + 1;
            }
            drawMenu();
            inputValue = 0;
        }
        else if((menu==4) & (buffer[i]=='s'))
        {
            //VDAC8_3_SetValue(VDAC8_3_Data-1);
            if(DAC_setFly>0)
            {
                DAC_setFly = DAC_setFly - 1;
            }
            drawMenu();
            inputValue = 0;
        }
        else if((buffer[i]!='0')&(buffer[i]!='1')&(buffer[i]!='2')
            ↪ &(buffer[i]!='3')&(buffer[i]!='4')&(buffer[i]!='5')&(
            ↪ buffer[i]!='6')&(buffer[i]!='7')&(buffer[i]!='8')&(
```

```
                    ↪ buffer[i]!='9'))
                {
                    drawMenu();
                    inputValue = 0;

                }
                else
                {
                    inputValue *= 10;
                    if(buffer[i]=='0')
                    {
                        inputValue += 0;
                    }
                    else if(buffer[i]=='1')
                    {
                        inputValue += 1;
                    }
                    else if(buffer[i]=='2')
                    {
                        inputValue += 2;
                    }
                    else if(buffer[i]=='3')
                    {
                        inputValue += 3;
                    }
                    else if(buffer[i]=='4')
                    {
                        inputValue += 4;
                    }
                    else if(buffer[i]=='5')
                    {
                        inputValue += 5;
                    }
                    else if(buffer[i]=='6')
                    {
                        inputValue += 6;
                    }
                    else if(buffer[i]=='7')
                    {
                        inputValue += 7;
                    }
                    else if(buffer[i]=='8')
                    {
                        inputValue += 8;
                    }
                    else if(buffer[i]=='9')
                    {
                        inputValue += 9;
                    }
                }
            }

        }
    }
    return;
}
```

```
int main(void)
{
    CyGlobalIntEnable; /* Enable global interrupts. */

    //CyWdtStart(CYWDT_1024_TICKS,CYWDT_LPMODE_NOCHANGE);

    CyDelay(100);

    DAC_setOut = VDAC8_3_Data;

    Control_Reg_OutputEnable_Write(0);

    //start freq dividers for triangle wave synchronization
    Control_Reg_1_Write(0);

    VDAC8_1_Start();
    Comp_1_Start();
    Comp_2_Start();

    //Timer_1_Start();
    //Timer_2_Start();
    //Timer_3_Start();
    //Timer_4_Start();

    VDAC8_3_Start();

    ADC_DelSig_1_Start();

    Opamp_1_Start();
    Opamp_2_Start();
    PGA_1_Start();
    Comp_3_Start();
    Sample_Hold_1_Start();
    Sample_Hold_2_Start();
    Sample_Hold_3_Start();
    //Opamp_vc2_Start();

    Opamp_3_Start();

    PWM_1_Start();
    //PWM_2_Start();

    //configure the triangle wave outputs for 0-4V, blocks until supply is
       ↪ above 4.6V
    configTri(4000);

    setOutput(3000);
    DAC_setOut = 31;


    //start the usb serial port
    USBUART_1_Start(0, USBUART_1_5V_OPERATION);
    while(!USBUART_1_bGetConfiguration());
```

```
    USBUART_1_CDC_Init();

    drawMenu();

    //uint8 writeBuffer[100];
    I2C_1_Start();
    setResistorI2C(1000);
    setCapacitorI2C(100000);
    //I2C_2_SlaveInitWriteBuf(writeBuffer, 100);
    //I2C_2_Start();



    Control_Reg_OutputEnable_Write(1);

    ADC_SAR_1_Start();
    ADC_SAR_1_StartConvert();
    ADC_SAR_2_Start();
    ADC_SAR_2_StartConvert();


    VDAC8_1_SetValue(1000/16);

    unsigned char flip = 0;
    for(;;)
    {
        //CyWdtClear();

        //t+h holds when high
        Control_Reg_2_Write(1);
        Control_Reg_3_Write(1);
        CyDelay(1);
        if(flip)
        {
            Control_Reg_2_Write(0);
            VDAC8_3_SetValue(DAC_setOut);
        }
        else
        {
            Control_Reg_3_Write(0);
            VDAC8_3_SetValue(DAC_setFly);
        }
        flip = 1 - flip;

        CyDelay(10);
        //I2C_2_SlaveClearWriteStatus();
        //I2C_2_SlaveClearWriteBuf();

        if(Button0_pin_Read()==0)
        {
            Bootloadable_1_Load();
            CyDelay(1000);
        }

        //uint8 toWrite[100] = "Hello";
        //I2C_1_MasterWriteBuf(73, toWrite, 23, I2C_1_MODE_COMPLETE_XFER);
        //I2C_1_MasterWriteByte(0b01010101);
```

```
        checkUSBSerial();
    }
}


/* [] END OF FILE */
```

Bibliography

[1] Linear Technology - Altera Arria 10 FPGA Development Kit. `http://www.linear.com/solutions/5844`. Accessed: 2018-01-24.

[2] Linear Technology - LTM4650 - Dual 25A or Single 50A DC/DC μModule Regulator. `http://www.linear.com/product/LTM4650`. Accessed: 2018-01-25.

[3] Linear Technology - Xilinx Virtex^{TM}-7 High-End Networking Card With Dual CXP Ports. `http://www.linear.com/solutions/5471`. Accessed: 2018-01-24.

[4] Prof. Slobodan Ćuk - New Topology Eliminates Magnetic Cores at 50kHz NOT 50MHz! `http://gecs.ieee.tn/wp-content/uploads/2017/03/GECS2017PaperEDITED.pdf`. Accessed: 2018-01-24.

[5] E. Abramov, A. Cervera, and M. M. Peretz. Optimal design of a voltage regulator based resonant switched-capacitor converter ic. In *2016 IEEE Applied Power Electronics Conference and Exposition (APEC)*, pages 692–699, March 2016.

[6] M. Biglarbegian, N. Shah, I. Mazhari, and B. Parkhideh. Design considerations for high power density/efficient pcb embedded inductor. In *2015 IEEE 3rd Workshop on Wide Bandgap Power Devices and Applications (WiPDA)*, pages 247–252, Nov 2015.

[7] Mehrdad Biglarbegian, Neel Shah, Iman Mazhari, Johan Enslin, and Babak Parkhideh. Design and Evaluation of High Current PCB Embedded Inductor for High Frequency Inverters. (1):2998–3003, 2016.

[8] A. Cervera, M. Evzelman, M. M. Peretz, and S. (. Ben-Yaakov. A high-efficiency resonant switched capacitor converter with continuous conversion ratio. *IEEE Transactions on Power Electronics*, 30(3):1373–1382, March 2015.

[9] S. Cuk. Step-down converter having a resonant inductor, a resonant capacitor and a hybrid transformer, March 29 2011. US Patent 7,915,874.

[10] Y. Nour, Z. Ouyang, A. Knott, and I. H. H. Jørgensen. Design and implementation of high frequency buck converter using multi-layer pcb inductor. In *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, pages 1313–1317, Oct 2016.

[11] K. Sano and H. Fujita. Performance of a high-efficiency switched-capacitor-based resonant converter with phase-shift control. *IEEE Transactions on Power Electronics*, 26(2):344–354, Feb 2011.

[12] Christopher Schaef and Jason T. Stauth. A 3-Phase Resonant Switched Capacitor Converter Delivering 7.7 W at 85% Efficiency Using 1.1 nH PCB Trace Inductors. *IEEE Journal of Solid-State Circuits*, 50(12):2861–2869, 2015.

[13] V. Yousefzadeh, E. Alarcon, and D. Maksimovic. Three-level buck converter for envelope tracking applications. *IEEE Transactions on Power Electronics*, 21(2):549–552, March 2006.