Current Shuttling Cell Voltage Balancers: Design, Evaluation, and Simulation

by

Mostafa H. Negm

S.B. Electrical Engineering and Computer Science,
S.B. Mechanical Engineering
Massachusetts Institute of Technology, 2019

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

September 2021

Author . . . .
Department of Electrical Engineering and Computer Science
August 13, 2021

Certified by . . . .
James L. Kirtley Jr.
Professor of Electrical Engineering
Thesis Supervisor

Certified by . . . .
William A. Lynch
Research Specialist
Thesis Supervisor

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

Current Shuttling Cell Voltage Balancers: Design, Evaluation, and Simulation

by

Mostafa H. Negm

Submitted to the Department of Electrical Engineering and Computer Science
on August 13, 2021, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Batteries are becoming increasingly important in a variety of applications, including electric vehicles and ships as well as load matching in electric grids. Cell voltage balancers are critical to extracting maximal performance out of batteries and to extending their lifespan. Charge pump balancers can quickly and efficiently shuttle charge across battery cells to equalize voltages. Component selection of MOSFETs and capacitors is vital in optimizing for performance, cost, and volume. This thesis presents experimental and PSpice simulation data from several capacitor-based charge pump configurations designed for cell voltage balancing. At 0.4 V cell differential, the peak balance current of the 2S balancer was over 9.9 A. At 0.8 V cell differential, the peak balance current of the 4S balancer was over 14.6 A. Ultimately, these charge pumps can be combined to construct a high-current and multilevel cell voltage balancer efficient across a wide range of voltages.

Thesis Supervisor: James L. Kirtley Jr.
Title: Professor of Electrical Engineering

Thesis Supervisor: William A. Lynch
Title: Research Specialist

Acknowledgments

First, thank you to Professor James Kirtley Jr. and Dr. William (Bill) Lynch for giving me the opportunity to do the research described in this thesis. Thank you to Bill for being a very hands-on research advisor (I talk to him at least 2-3 times a day). He is a talented engineer who in turn has made me a better engineer. Many of the ideas for the work described in this thesis were his. Thank you to the Office of Naval Research (ONR) for funding this research under grant numbers N00014-19-1-2359 and N00014-21-1-2497.

Thank you to Professor Steven Leeb for sparking my interest in power electronics. 6.131 Power Electronics Laboratory was one of my favorite classes at MIT. (6.334 Power Electronics by Professor Perreault was also one of my favorites.)

Thank you to Thomas (Tommy) Krause, whom I've had many interesting conversations with in the lab. I truly hope he becomes a professor one day. Thank you to Lukasz Huchel for being a helpful resource to everyone in the lab including me. Thank you to MENG buddy Emmanuel Havugimana. Shout out to Bonnie Jones!

Thank you to Ahmad Negm for his help in the PCB designs. Thank you to Mr. David Otten for his feedback on the PCB designs. Several changes were made due to his recommendations.

Thank you to Mrs. Linda and Mr. Chuck Johnson, and Mr. Pardha Gadiyaram for always believing in me. Thank you to many wonderful educators (teachers, professors, etc.) I've had the fortune of having; there are too many to name.

Last, but not least, thank you to my family, my parents Hussein and Samira, and my siblings Maggie, Ahmad, and Abdullah, for always supporting and believing in me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Batteries are useful for a wide range of applications such as in electric vehicles and ships as well as load matching in electric grids to increase efficiency and reduce emissions. For example, the Tesla Model S uses a total of 7616 lithium-ion cells [1]. In [2], a 3 MW energy storage system including five 600 kW lithium-ion battery modules can provide full ship backup power for 10 minutes with substantial fuel savings! Other applications include UPS and pulse power. The voltages of individual cells and supercapacitors are low, typically in the range 2–4 V [3]. Therefore, the cells need to be configured in series to reach the sufficiently high voltage required. When multiple cells are connected in series, the cell voltage is not always equal to the pack voltage divided by the number of cells [4]. This is due to inherent slight differences among cells in terms of capacity and internal resistance [1]. In short, since series connected cells have the same current flowing through them, due to the inherent differences, the voltage of individual cells will be different [3, 5].

Cell voltage balancing is important for several reasons. One of these is lifespan. A cell that exceeds its maximal recommended charging voltage will degrade prematurely, a process that is auto-accelerating [4]. A second reason is incomplete performance extraction. Due to protection circuitry, a battery pack will stop charging if even one of the cells reaches the maximal recommended charging voltage [6]. Similarly, a battery pack will stop discharging if even one of the cells reaches the minimal recommended charging voltage.

There are two main cell voltage balancing techniques: passive and active [3]. A passive balancer is a simple circuit to make; as a cell reaches the maximum voltage, a resistor is connected in parallel with the cell through a MOSFET to drain the excess energy [1]. However, this leads to energy dissipation and heat-related issues [1, 3, 7]. On the other hand, an active balancer, while more complicated and expensive, shuttles charge back and forth between cells. Active balancers come in three main categories: capacitive based, inductive based, and transformer based [1].

This thesis is organized as follows: Chapter 2 presents the design of cell voltage balancers. It introduces the charge pump balancer concept and presents three different power circuits and the accompanying control circuit. It also discusses MOSFET, capacitor, and gate driver selection. Chapter 3 presents the evaluation of cell voltage balancers. It discusses the experimental setup, gate drive power, efficiency, power dissipation, and PCB design. Chapter 4 presents the simulation of cell voltage balancers in PSpice. Chapter 5 presents a cell tester circuit for cycling cells. Finally, Chapter 6 gives conclusions and discusses potential future work.

Chapter 2

Cell Voltage Balancer: Design

2.1 Charge Pump Balancer Concept

Figure 2-1 shows the charge pump based cell voltage balancer concept. The basic idea is that in state 1, switches 1 and 3 are closed, and the flying capacitor is connected in parallel with bypass capacitor 1 and they exchange charge. Likewise, in state 2, switches 2 and 4 are closed, and the flying capacitor is connected in parallel with bypass capacitor 2 and they exchange charge. Over time, charge will be transferred from the higher voltage cell/battery (group of cells) to the lower voltage cell/battery, and the voltages will equalize. Note, battery B1 and bypass capacitor 1 are constantly exchanging charge. Similarly, battery B2 and bypass capacitor 2 are also constantly exchanging charge. It goes without saying that the capacitance of the cells is much, much greater than the capacitance of the bypass capacitors. The bypass capacitors filter the battery current and decouple any connection inductance [8].

There are several commercial products on the market available in this topology; they usually go under the name of switched capacitor/charge pump inverter. These include the Texas Instruments LM266x, Maxim Integrated MAX889, and Linear Technology LT1054 [9, 10, 11], just to name three. There even exist on AliExpress, Amazon, Ebay, etc. modules that conveniently incorporate on a breakout board the LM2662 IC, the flying and two bypass capacitors, and header pins. [12] gives some additional insight into the charge pump inverter.

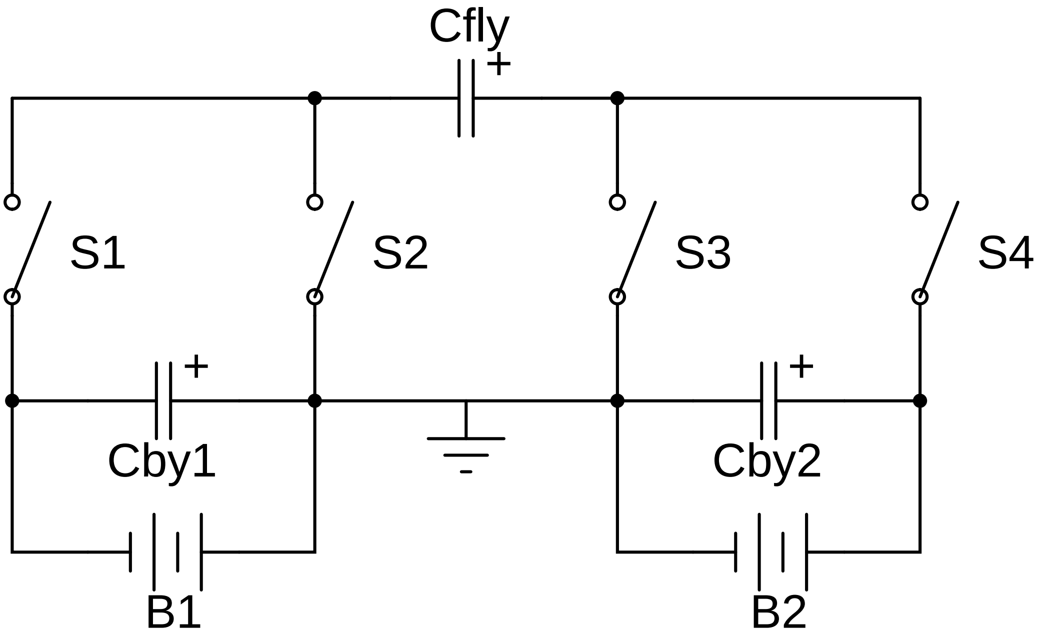

Figure 2-1: Charge pump based cell voltage balancer concept.

2.2 Power Circuit 1: 2S Balancer

Power Circuit 1, shown in Figure 2-2, is a 2S charge pump based cell voltage balancer consisting of four MOSFETs and three capacitors. 2S stands for 2 cells in series; in general, NS stands for N cells in series. (As an aside, NP stands for N cells in parallel.) MOSFETs M1 and M3 are N-channel, while M2 and M4 are P-channel. Enhancement mode (as opposed to depletion mode) MOSFETs are normally 'off' (i.e., at zero gate-source voltage V_{GS}) and require a gate-source voltage above the threshold (in magnitude) to turn 'on'. N-channel MOSFETs are turned on with a positive gate-source voltage, while P-channel MOSFETs are turned on with a negative gate-source voltage. Due to this complementary logic of N- and P-channel MOSFETs, a common gate signal can be used to drive all four MOSFETs in Power Circuit 1. This greatly simplifies the control circuit. When the gate signal is high, N-channels M1 and M3 conduct, and the flying capacitor is connected in parallel with bypass capacitor 1 and they exchange charge. Similarly, when the gate signal is low, P-channels M2 and M4 conduct, and the flying capacitor is connected in parallel with bypass capacitor 2.

MOSFETs M1 and M4 are high-threshold, while M2 and M3 are low-threshold.

Low-threshold MOSFETs are MOSFETs that can be turned on with low (in magnitude) gate-source voltage. Similarly, high-threshold MOSFETs are MOSFETs that can be turned on with high gate-source voltage. In the datasheets, manufacturers will typically give specs for important parameters such as the on-state resistance $R_{DS(on)}$ and gate charge Q_g at a typical V_{GS} value(s). For low-threshold MOSFETs, a common typical V_{GS} value used by manufacturers is 4.5 V (sometimes 3 V and/or 2.5 V are used). For high-threshold MOSFETs, a common typical V_{GS} value used by manufacturers is 10 V (sometimes 12 V and/or 20 V are used). As a general note, in the following figures, low-threshold MOSFETs are represented with green gates and high-threshold MOSFETs with red gates.

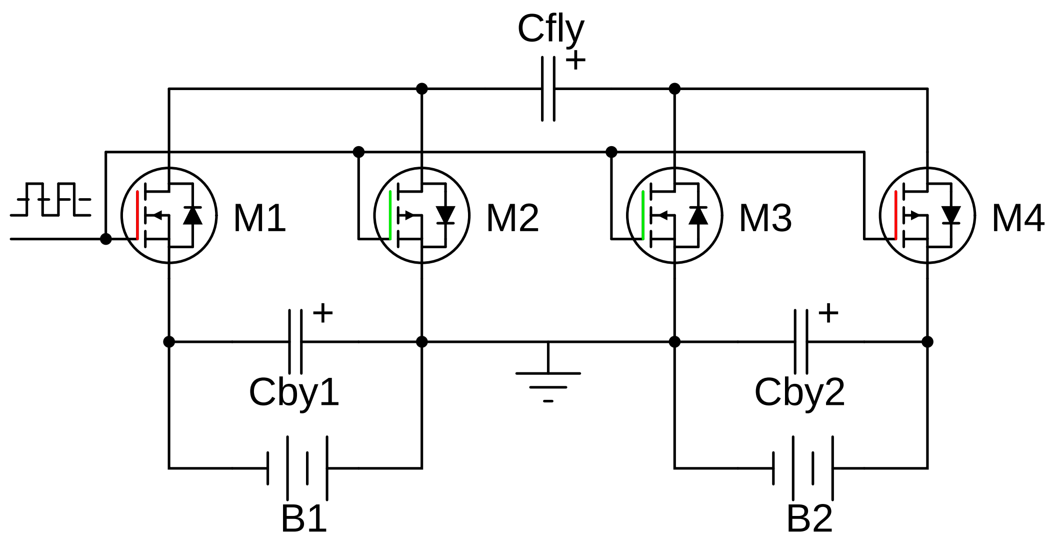

Figure 2-2: Power Circuit 1: 2S balancer. MOSFETs M1 and M3 are N-channel, while M2 and M4 are P-channel. M1 and M4 are high-threshold, while M2 and M3 are low-threshold.

The common source MOSFETs M2 and M3 can be low-threshold because the complementary logic of N- and P-channel MOSFETs prevents simultaneous conduction in these two switches. For instance, as the common gate signal swings through ground, both M2 and M3 are in the off state. Transient conduction of MOSFET pair M1 and M2, as well as pair M3 and M4, is also unlikely because neither M2 nor M3 can conduct until the common gate voltage swings past ground by more than the threshold voltage. The most likely incorrect transient conduction state in this power circuit is through MOSFETs

M1 and M4. When the gate voltage is close to the neutral ground, N-channel M1 has positive gate-source voltage and P-channel M4 has negative gate-source voltage. If the thresholds of M1 and M4 are not high enough, these two MOSFETs could transiently conduct, thus charging the flying capacitor to a higher voltage and thereby reducing the circuit efficiency. Therefore, M1 and M4 should be high-threshold.

2.3 Power Circuit 2: 4S Balancer

Power Circuit 2, shown in Figure 2-3, is a 4S charge pump based cell voltage balancer consisting of four MOSFETs and three capacitors. The positions of the N- and P-channel MOSFETs are switched compared with Power Circuit 1; thus, MOSFETs M1 and M3 are P-channel, while M2 and M4 are N-channel. This is because in Power Circuit 1, for a battery voltage of 8 V, M1 and M4 would simultaneously conduct as the gate signal swings through ground, even with the use of high-threshold MOSFETs. The sources and drains are also flipped compared with Power Circuit 1; this is necessary, else there would be an unwanted conduction path through the reverse p-n junction body diodes and the cells would just short themselves, even with no gate signal present.

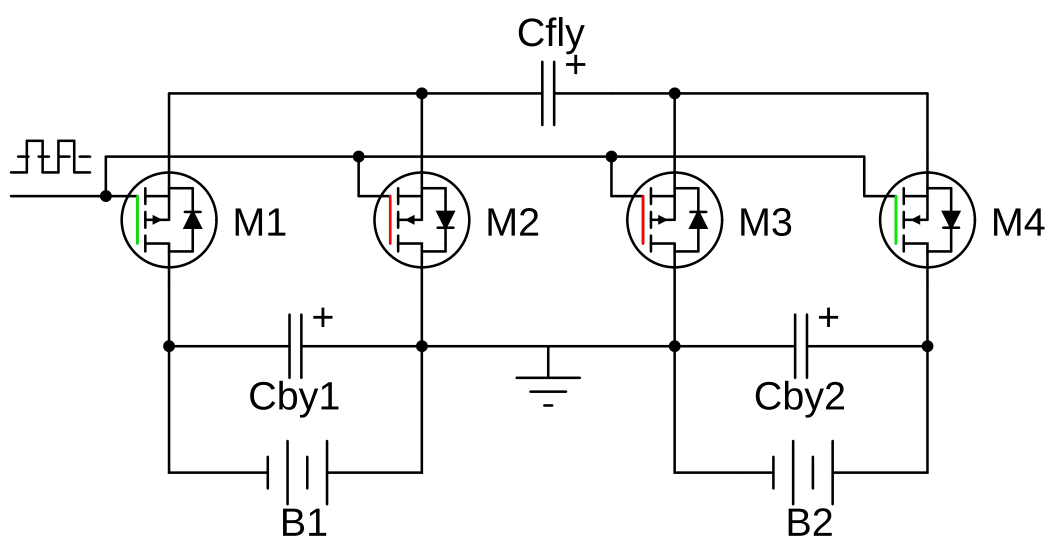

Figure 2-3: Power Circuit 2: 4S balancer. MOSFETs M1 and M3 are P-channel, while M2 and M4 are N-channel. M1 and M4 are low-threshold, while M2 and M3 are high-threshold.

In this power circuit, transient conduction of MOSFET pair M1 and M2, as well as pair M3 and M4, is impossible because of their common source connections. Gate voltages beyond the power circuit supply rails are needed to apply sufficient negative voltage to the gate of P-channel M1 and sufficient positive voltage to the gate of N-channel M4. These should be low-threshold to minimize the gate voltage needed beyond the power rails. Note, the MOSFETs need to have sufficient gate-source voltage rating for the relatively high voltage that the gate drive applies, both when the switch is on and off. Extra gate voltage used to drive MOSFETs M1 and M4 is available for M2 and M3, which have a neutral source voltage when on. It is possible for M2 and M3 to transiently cross-conduct if their threshold voltages are not high enough. For example, when the gate signal is some intermediate positive voltage during the transition from high to low, N-channel M2 can remain on as P-channel M3 turns on prematurely. A similar thing can happen when the gate signal is transitioning from low to high. This could partially discharge the flying capacitor and thereby reduce the circuit efficiency. Therefore, M2 and M3 should be high-threshold.

2.4 Power Circuit 3: 8S Balancer

Power Circuit 3, shown in Figure 2-4, is an 8S charge pump based cell voltage balancer consisting of four MOSFETs, three capacitors, and one inductor. The inductor is added to lower the resonant frequency. In Power Circuit 2, for a battery voltage of 16 V, M2 and M3 would simultaneously conduct during the transitions from high to low and low to high, even with the use of high-threshold MOSFETs. To avoid this, for the 8S balancer, MOSFETs M1 and M2 are P-channel, while M3 and M4 are N-channel. Because the two P-channel MOSFETs (M1 and M2) are adjacent, as are the two N-channel MOSFETs (M3 and M4), two inverted gate signals are necessary to drive the MOSFETs. One gate signal drives the outer MOSFETs M1 and M4, and the other gate signal drives the inner MOSFETs M2 and M3. This is unlike Power Circuits 1 and 2 where one gate signal is sufficient. A bootstrap circuit is proposed to level shift the gate signal more negative for P-channel M1 and more positive for N-channel M4.

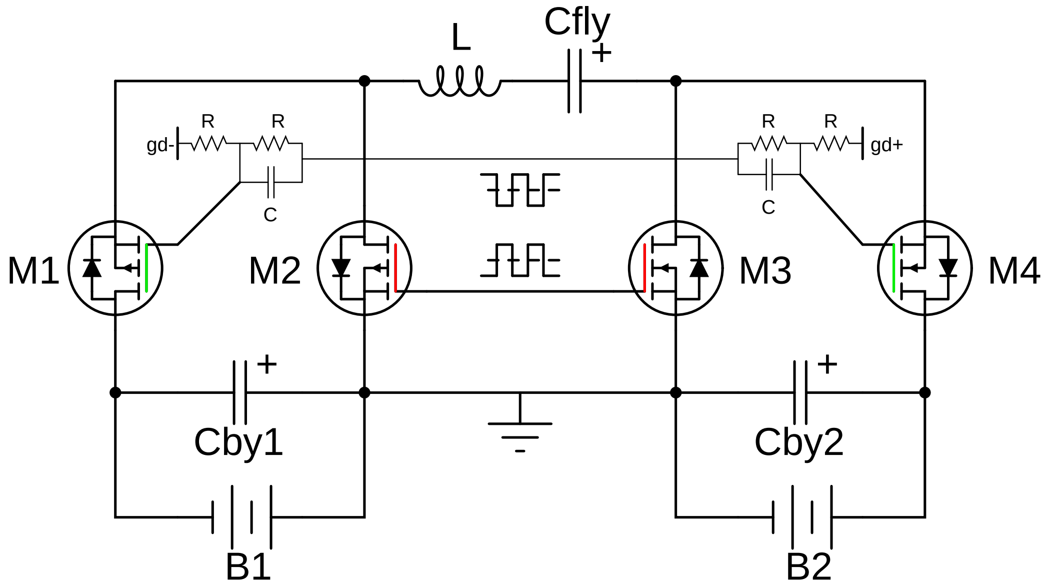

Figure 2-4: Power Circuit 3: 8S balancer (with bootstrap circuit shown). MOSFETs M1 and M2 are P-channel, while M3 and M4 are N-channel. M1 and M4 are low-threshold, while M2 and M3 are high-threshold.

2.5 MOSFET Selection

Table A.1 in Appendix A shows a subset of the MOSFET table used in the component selection process.

For an NS balancer, each of the four MOSFETs in the circuit has to block (N/2)S voltage in the static case (i.e., the circuit is either in state 1 or state 2) when off. However, this does not take into account the dynamics, and thus any potential voltage spikes during transitions between states, so a safety margin is necessary. For a 2S balancer, 10 V rated MOSFETs provide a 150 % safety margin over the 4 V static case. Because there exist many, many power MOSFETs rated for at least 10 V, the max V_{DS} rating is not much of a concern for a 2S balancer. For a 4S balancer, 20 V rated MOSFETs also provide a 150 % safety margin over the now 8 V static case. For an 8S balancer, 30 V rated MOSFETs provide a 87.5 % safety margin over the 16 V static case.

The drain-source on-state resistance $R_{DS(on)}$ of a MOSFET is both a function of the gate-source voltage V_{GS} and the junction temperature T_J. $R_{DS(on)}$ decreases as V_{GS} increases (the MOSFET is 'more on'), while $R_{DS(on)}$ increases as T_J increases. The gate charge Q_g is the total amount of charge needed to charge up the parasitic capacitances (e.g., C_{GS} and C_{GD}) to turn the MOSFET on [13, 14]. There is typically a trade-off relationship between the on-state resistance and the gate charge: the smaller the die size, the lower the gate charge but the higher the on-state resistance [13].

The turn-on time t_{on} is defined as the sum of the turn-on delay time $t_{d(on)}$ and the rise time t_r. Similarly, the turn-off time t_{off} is defined as the sum of the turn-off delay time $t_{d(off)}$ and the fall time t_f. Every switching period T_{sw}, each MOSFET turns on once and turns off once. Thus, the turn-on and turn-off times should be much faster than the switching period, i.e., $t_{on} + t_{off} \leq 0.1T_{sw}$. For example, if $T_{sw} = 10\,\mu s$ (i.e., $f_{sw} = 100\,\text{kHz}$), $t_{on} + t_{off}$ should at most be $1\,\mu s$. In addition, since the four MOSFETs share a common gate signal, it is especially important that no MOSFET turns on much quicker than another one turns off. Comparable turn-on and turn-off times help prevent shoot-through current.

The junction-to-ambient thermal resistance $R_{\theta JA}$ can be used as a first-order approximation to how much the device will heat up during operation while dissipating P watts of power: $T_J = T_A + R_{\theta JA}P \leq T_{J(\max)}$. One should never exceed the the maximum junction temperature $T_{J(\max)}$, usually 150 or 175 °C for power MOSFETs. Note, the $R_{\theta JA}$ values listed are typically measured on a $1\,\text{in}^2$ FR-4 material board with 2 oz copper weight and thus are optimistic; the thermal resistance will obviously be higher for a more minimal footprint size such as $1/4\,\text{in}^2$.

The gate location is the location of the gate relative to the source (left or right) if the drain is oriented at the top. A convenient gate location can make a board layout simpler.

Price is also an important consideration and is not shown in the table because it depends on the time of buy, quantity, supplier, etc.

2.6 Capacitor Selection

Section A.2 in Appendix A show subsets of the capacitor tables used in the component selection process.

Each bypass capacitor is directly across a cell/battery, so it needs to be able to withstand the maximum cell voltage (i.e., at the full state of charge), plus any voltage ripple on top of that. The flying capacitor in either state is connected to one of the bypass capacitors, so it also needs to withstand an equal voltage. For a lithium iron phosphate ($LiFePO_4$) cell with a nominal voltage of 3.2 V and max operating voltage of 3.6 V, 6.3 V rated bypass and flying capacitors would give a comfortable safety margin of 75 %. For a higher max voltage of 4.2 V, the safety margin would still be reasonable at 50 %. For a lower voltage cell such as lithium titanate oxide (LTO) with a nominal cell voltage of 2.3 V and max operating voltage of 3.0 V, 4 V rated capacitors would give a low safety margin of 33 %. For the 4S balancer, if 8 V is the maximum battery voltage, 10 V rated capacitors would give a low safety margin of 25 %, while 16 V rated capacitors would give a comfortable safety margin of 100 %. For the 8S balancer, if 16 V is the maximum battery voltage, 20 V rated capacitors would give a low safety margin of 25 %, while 16 V rated capacitors would give a reasonable safety margin of 56.25 %.

Among the specs listed in the tables are capacitance, ESR (equivalent series resistance), and volume. The amount of charge that can be transferred in each switching cycle is proportional to the capacitance (see $\Delta Q = C\Delta V$), so larger capacitances for the flying and bypass capacitors are preferable [15]. Larger capacitances also have the benefit of lowering the control power; this is because the gate drive power is proportional to the switching frequency (see Section 3.2) and the resonant frequency of the circuit is $f_0 = \frac{1}{2\pi\sqrt{LC}}$ [1, 15, 16, 17, 5]. Note, L is the total parasitic inductance and ESL of the flying and bypass capacitors if no discrete inductor is present in the circuit. Using low ESR capacitors lowers the overall circuit resistance and increases the balance current. For power dissipation, the use of low ESR capacitors is also important to avoid the dissipation being concentrated in the capacitors rather than the power MOSFETs (see Section 3.4). Paralleling capacitors is one easy way to increase the capacitance and lower the ESR.

Some important specs not listed in the tables are capacitance tolerance ($\pm 20\,\%$ is common), voltage and temperature coefficients (especially important for ceramics), and ripple current ratings (important for aluminum capacitors). Price is also an important consideration that is not shown.

2.7 Control Circuit

One large advantage of the power circuits described is the simplicity of drive. For Power Circuits 1 and 2, the same gate signal can be used to drive all 4 MOSFET gates. For Power Circuit 3, two gate signals that are inverse of each other (in sign, not necessarily in magnitude) are required to drive the gates. One can simply put the LEDs of the two optocoupler gate drivers antiparallel to ensure that the outputs cannot be high simultaneously.

An astable (not to be confused with bistable or monostable) multivibrator is a circuit that is not stable in either of the two possible output states and thus acts as an oscillator [18]. An astable multivibrator can be used to generate a square wave.

Figure 2-5 shows an op-amp based astable multivibrator [18, 19]. As mentioned, there are two states: output high (i.e., $V_O = V_{OH}$) and output low ($V_O = V_{OL}$). For analysis, let us assume the initial state is output high. Let us also assume the capacitor voltage is low (this will soon become apparent). Let us also make the reasonable assumption that $V_{OH} = -V_{OL} \equiv V_{SAT}$ (true for the Texas Instruments OPA2192 op-amp that was used in many of the experiments).

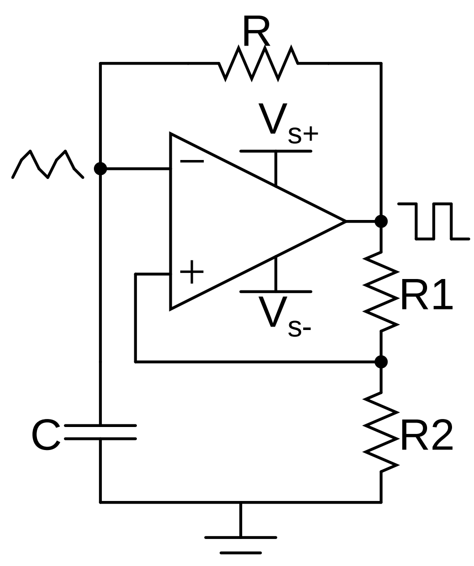

Figure 2-5: Op-amp astable multivibrator.

The resistor R and capacitor C make a series RC circuit. C will charge through

R and the capacitor voltage V_C will exponentially approach V_{SAT} with time constant $\tau = RC$. When V_C surpasses βV_{SAT}, where $\beta \equiv \dfrac{R_2}{R_1 + R_2}$ is defined as the feedback fraction, the inverting input V_- is now greater than non-inverting input V_+ and thus the output saturates to -V_{SAT}. In this second state, C will now *discharge* through R and exponentially approach $-V_{SAT}$. When V_C falls below $-\beta V_{SAT}$, V_- is now less than V_+ and the output saturates back to V_{SAT}, the first state. The cycle repeats with frequency $f = 1/\left[2RC \ln\left(\dfrac{1+\beta}{1-\beta}\right)\right]$ (a derivation is given in Appendix B).

The RC time constant multiplied by $2\ln\left(\dfrac{1+\beta}{1-\beta}\right)$ sets the oscillation period. Capacitors with $\pm 5\,\%$ tolerance are common, and are fine for this application; meanwhile, resistors with $\pm 1\,\%$ tolerance are common. To minimize unnecessary power dissipation, for a constant RC, R should be large and thus C is small. For a frequency range of 10–100 kHz, and at a feedback fraction $\beta = 0.75$, $C = 1\,\text{nF}$ gives a reasonable range of 2.57–25.7 kΩ for R. The values of R_1 and R_2 should also be large for similar reasons. For $R_1 = 10\,\text{k}\Omega$, and the same $\beta = 0.75$, $R_2 = 30\,\text{k}\Omega$.

Only 1 op-amp is needed to generate a square wave. If a pulse wave (a generalization of the square wave where the duty cycle does not have to be 50 %) is instead desired, a second op-amp is needed. Figure 2-6 shows a schematic of an op-amp based pulse wave generator. The capacitor waveform from the astable multibrator is fed into the inverting input of the second op-amp. An additional DC voltage is fed into the non-inverting input. The second op-amp functions as a comparator; the proportion of time that the DC voltage is higher than the sawtooth wave is the resulting duty cycle. The pull-down resistor ensures roughly 50 % duty cycle if no explicit DC voltage is applied to the non-inverting terminal.

2.8 Gate Driver Selection

The selection of the gate driver is not too critical, at least in comparison to MOSFETs and capacitors. Regardless, several gate drivers were tested, including the Isocom IS480P, IXYS IX3180G, Texas Instruments UCC27321, and Toshiba TLP5774. The Texas Instruments LM7322, a dual op-amp with the ability to drive a high capacitive

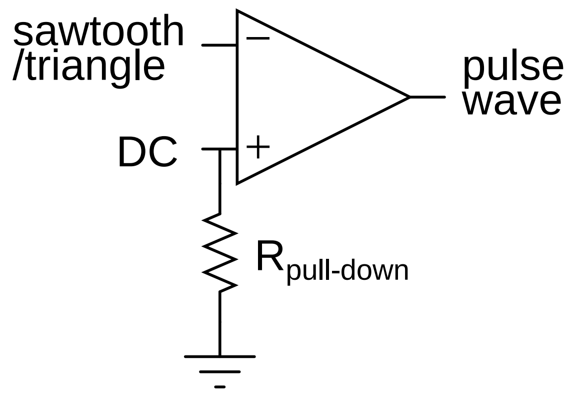

Figure 2-6: Op-amp pulse wave generator.

load, was also tested as a driver. The TLP5774 was the best driver we tested in terms of peak output current while consuming low quiescent supply and LED current, and the IX3180G performed similarly.

The IS480P, IX3180G, and TLP5774 are optocouplers and thus have the advantage of electrical isolation. The UCC27321 has an enable pin, which may be useful in stopping balancer operation when the voltage differential of the cells is below a small threshold. One of the two op-amps in the LM7322 can be used as an astable multivibrator, so one chip could potentially double as both the clock and the driver.

The UCC27321 and IS480P require a minimum supply voltage of 4 V and 4.5 V, respectively, compared to the 10 V of the TLP5774 and IX3180G. The minimum supply voltage for the LM7322 is 2.5 V. The LM7322, and possibly the UCC27321 and IS480P, have a low enough minimum supply voltage requirement that they might be able to be directly powered from the cells without the use of a DC-to-DC converter. The UCC27321 has a low *maximum* supply voltage rating of 15 V, practically eliminating it from consideration based on that spec alone. In fact, when the UCC27321 driver was tested at ±7 V, the driver failed to drive four MOSFET gates at around 50 kHz. It worked fine though for lower voltages. The IX3180G has a more moderate, but still low, maximum supply voltage rating of 20 V. The IS480P, TLP5774, and LM7322 have high maximum supply voltage ratings of at least 30 V.

Chapter 3

Cell Voltage Balancer: Evaluation

3.1 Experimental Setup

To test the cell voltage balancers, the combination of a DC power supply and DC electronic load was used to simulate a cell/battery (with infinitely large capacity). Both are needed as typical power supplies are one quadrant; this means they can only source current at a positive voltage. An electronic load, on the other hand, is designed to sink current at a positive voltage, so it complements a power supply. Power supplies (electronic loads too) with a voltage and current resolution of 1 mV and 1 mA or better are common and should be used.

Figure 3-1 shows a diagram of the experimental setup used in many of the tests. The power supplies are in constant voltage mode; the electronic loads are in constant current mode. Note, the current limit of the power supply that is emulating the higher voltage cell must be sufficiently high. Similarly, the current setting of the electronic load that is emulating the lower voltage cell must also be sufficiently high. For this specific example, the higher voltage 'cell' is at 3.400 V and is sourcing $6.716\,\mathrm{A} - 1.999\,\mathrm{A} = 4.717\,\mathrm{A}$. Similarly, the lower voltage 'cell' is at 3.000 V, but instead sinking $5.999\,\mathrm{A} - 1.279\,\mathrm{A} = 4.720\,\mathrm{A}$. The sourcing and sinking currents should be the same or almost the same for a well-designed circuit with properly selected components. An imbalance would suggest shoot-through current during switching. This could be due to several reasons; to name a few: inappropriate MOSFET thresholds (e.g., using a low-threshold MOSFET in place

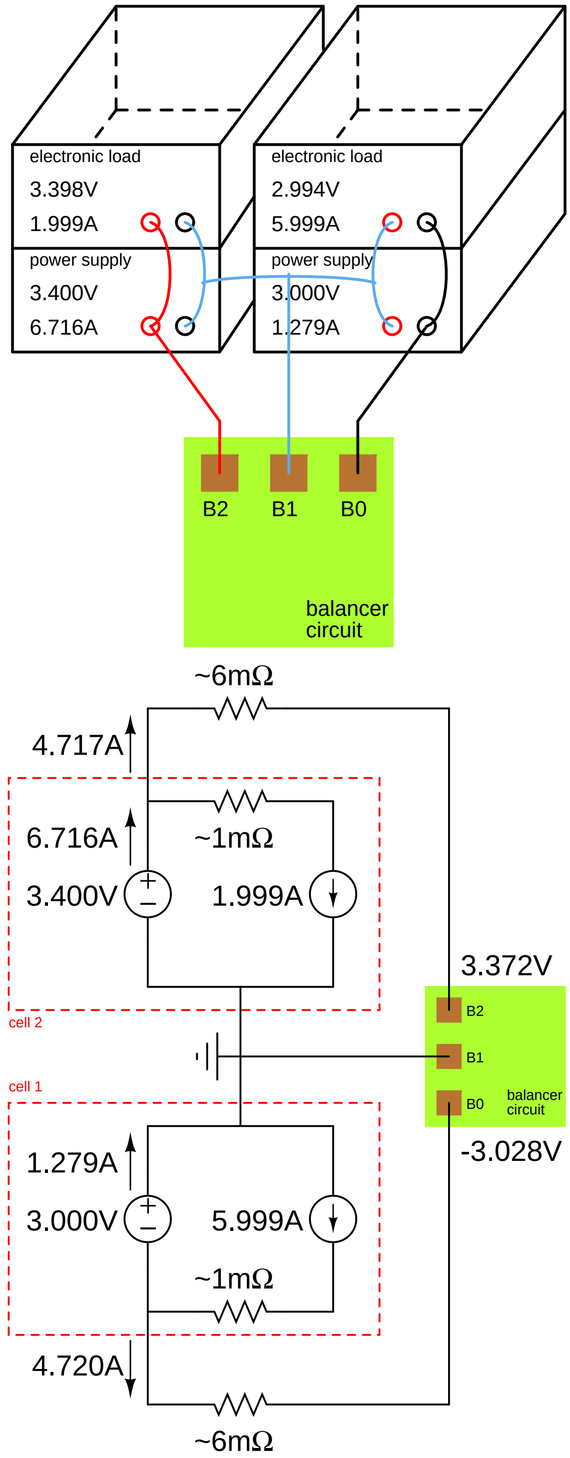

Figure 3-1: Experimental setup diagram (top) and simplified circuit model (bottom).

of a high-threshold MOSFET), incompatible MOSFET timings (e.g., using a MOSFET that is comparatively slow to turn off), or incorrect cell voltages (e.g., using total of 4S cell voltage for a 2S balancer). The average of the source and sink currents is taken to be the balance current.

Figure 3-1 also shows a simplified circuit model of the experimental setup with the important parasitic resistances shown. The resistance is due to a combination of wire and contact resistance. 10 AWG wire, which has a resistance of $\sim 1.0\,\text{m}\Omega/\text{ft}$, was used for the experiments. There is little net current flowing through the ground terminal, so any parasitic resistances there are ignored. Note, the voltage differential seen at the balancer circuit will be less than the 0.4 V differential set at the supplies because of the parasitic resistances from long power supply leads. In this example, the cell differential as seen by the balancer circuit is $3.372\,\text{V} - 3.028\,\text{V} = 344\,\text{mV}$. The remote sense features of the power supplies may be used to compensate for the voltage drop in the wires.

One-quadrant power supplies and electronic loads are ubiquitous EE (electrical engineering) lab equipment. However, instead of a power supply and an electronic load to simulate a cell, a bidirectional (i.e., two-quadrant) power supply can be used. A bidirectional power supply can both source and sink current at a positive voltage. Being physically one device, a bidirectional power supply offers several advantages. For example, a power supply and electronic load when connected may interact with each other strangely when a feature such as remote sense is turned on. In short, remote sense may or may not work; this would not happen with a bidirectional power supply. Another benefit is that bidirectional power supplies often are regenerative when sinking current, and thus send the majority of their absorbed energy back to the grid [20]. This is better for the environment and less wasteful than an electronic load. In the above example, the electronic loads dissipate roughly $2\,\text{A} * 3.4\,\text{V} + 6\,\text{A} * 3\,\text{V} = 24.8\,\text{W}$ of power. One disadvantage of a bidirectional power supply is that it tends to be more expensive than the sum of a (one-quadrant) power supply and an electronic load.

3.2 Gate Drive Power

This section shows an example calculation of the gate drive power, henceforth referred to as gate power, for Power Circuit 1 and compares that calculation with experimental results.

For the calculation and experiment, the gate signal is a square wave with amplitude 7 V. The MOSFETs M1, M2, M3, and M4, respectively, are the ON Semiconductor NVMYS1D3N04C, Vishay SQD40031EL, Nexperia PSMNR70-30YLHX, and Infineon IPD90P03P404. The voltage of cell 1 is $V_{B1} = 3.0\,\text{V}$ and cell 2 is $V_{B2} = 3.4\,\text{V}$. The Toshiba TLP5774 gate driver was used with a $3.3\,\Omega$ gate resistor.

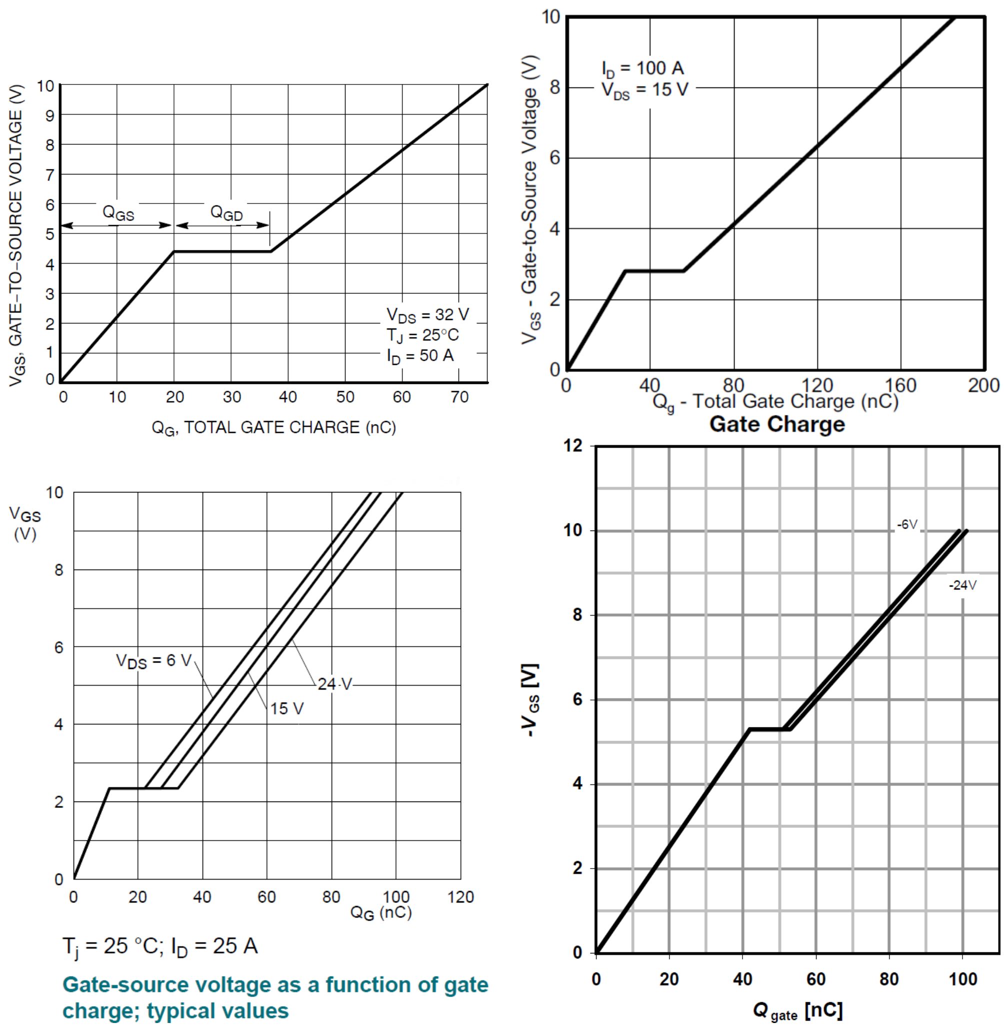

Figure 3-2: Gate charge curves of NVMYS1D3N04C (top left), SQD40031EL (top right), PSMNR70-30YLHX (bottom left), and IPD90P03P404 (bottom right) MOSFETs courtesy of the manufacturers' datasheets [21, 22, 23, 24].

The gate power (partial) is $P_{gate} = Q_g V_g f_{sw}$ [25]. Q_g is the gate charge, V_g is the gate voltage relative to the source voltage (i.e., V_{gs}), and f_{sw} is the switching frequency. As previously mentioned, Q_g is the total amount of charge needed to turn the MOSFET on [13, 14]. The $Q_g V_g$ product is the gate energy, and is represented by the area of the rectangle from the intersection point on the curve to the origin [26]. The Q_g values can be found from the gate charge curves on the manufacturer datasheets, reproduced in Figure 3-2. Remember that the sources of M1 and M4 are not at ground; the source of M1 is at $-V_{\text{B1}}$ while the source of M4 is at $+V_{\text{B2}}$.

The calculated gate powers of the four MOSFETs, and their total sum, as a function of f_{sw} are

$$P_{gate}|_{\text{NVMYS1D3N04C}} = [(75\,\text{nC})(10\,\text{V}) + (18\,\text{nC})(4\,\text{V})]f_{sw} = 8.22e - 7 * f_{sw}$$
$$P_{gate}|_{\text{SQD40031EL}} = [2(132\,\text{nC})(7\,\text{V})]f_{sw} = 1.85e - 6 * f_{sw}$$
$$P_{gate}|_{\text{PSMNR70-30YLHX}} = [2(75\,\text{nC})(7\,\text{V})]f_{sw} = 1.05e - 6 * f_{sw}$$
$$P_{gate}|_{\text{IPD90P03P404}} = [(104\,\text{nC})(10.4\,\text{V}) + (28\,\text{nC})(3.6\,\text{V})]f_{sw} = 1.18e - 6 * f_{sw}$$
$$\sum P_{gate} = 4.90e - 6 * f_{sw}$$

As astute reader may notice that the curves in Figure 3-2 only show the gate charge values for positive V_{GS} for the two N-channel MOSFETs, and negative V_{GS} for the two P-channel MOSFETs. Because the datasheets do not show the negative part of the curve for the N-channel MOSFETs nor the positive part of the curve for the P-channel MOSFETs, it was assumed, as an initial guess, that the gate charge curves *are* symmetric. However, note that gate charges from different gate voltage swings are generally not comparable; for example, there is no exact way to determine Q_g for a swing of $-10\,\text{V}$ to $+10\,\text{V}$ if Q_g is only given for $0\,\text{V}$ to $+10\,\text{V}$ [27].

Finally, adding in the quiescent power of the gate driver, the gate power is $P_{gate} = 4.90e - 6 * f_{sw} + (0.002\,\text{A})(14\,\text{V}) = 4.90e - 6 * f_{sw} + 0.028\,\text{W}$. The 2 mA is the power supply reading of the unloaded TLP5774 driver that was used in the experiment. The datasheet also confirms this value. Figure 3-3 shows the experimental data plotted along with the equation predicted from the calculation. The line of best fit of the experimental data is $P_{gate} = 8.27e - 6 * f_{sw} + 0.023\,\text{W}$. Compare the calculation with the experimental results. The calculation is an underestimate by ~41 %. Suffice to say, the gate power is proportional to the gate charge, gate voltage, switching frequency product. Additionally, a small offset is added to take into account the quiescent power.

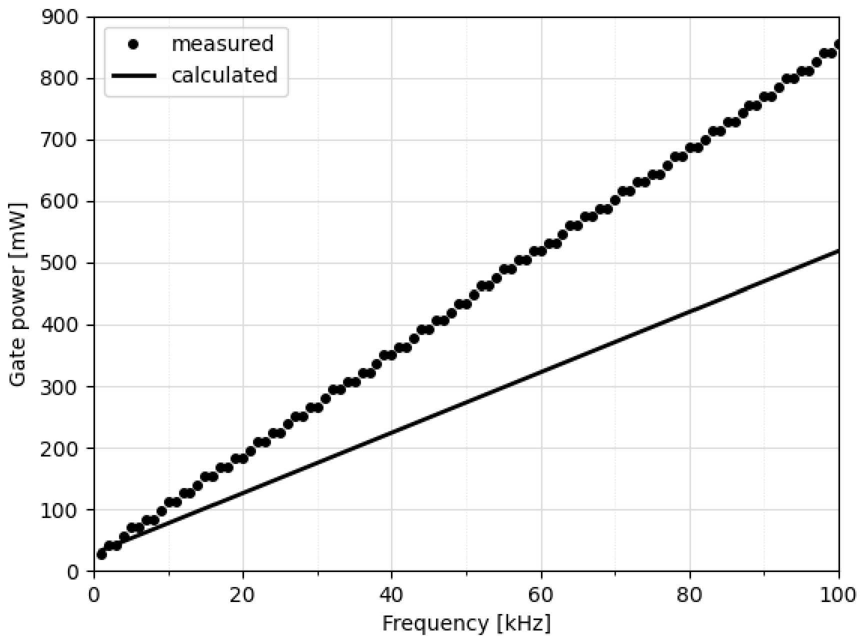

Figure 3-3: Gate power vs frequency for the NVMYS1D3N04C, SQD40031EL, PSMNR70-30YLHX, and IPD90P03P404 MOSFETs.

Note, the proportion of the gate power that is dissipated in the gate resistor is $\frac{R_g}{R_o + R_g}$, where R_g is the value of the gate resistor and R_o is the output impedance of the gate driver [28]. The remaining power is dissipated in the gate driver. In our case, the power dissipated in the gate resistor at 100 kHz switching frequency is $\frac{3.3\,\Omega}{7/6 + 3.3\,\Omega}(854\,\text{mW}) = 631\,\text{mW}$. This means that the gate resistor should be rated for at least 3/4 W power dissipation. The remaining power, 223 mW, is dissipated in the gate driver; this dissipation is less than the maximum allowable for the TLP5774: 500 mW.

3.3 Efficiency

The efficiency of the circuit can be calculated by $\eta = \frac{P_{\text{out}}}{P_{\text{in}}} = \frac{V_{\text{out}} I_{\text{out}}}{V_{\text{in}} I_{\text{in}} + P_{\text{control}}}$, where V_{out} is the voltage of the cell/battery at the lower voltage, i.e., $V_{out} < V_{in}$. The asymptotic efficiency, which ignores the power consumption of the control circuit (i.e., $P_{\text{control}} = 0$) and assumes zero current imbalance (i.e., $I_{out} = I_{in}$), is $\eta_{asymptotic} = \frac{V_{out}}{V_{in}}$. Let us define the average voltage of the two cells/batteries being balanced as $V_{avg} \equiv \frac{1}{2}(V_{in} + V_{out})$ and their voltage differential as $\Delta V \equiv V_{in} - V_{out}$. Rewriting the asymptotic efficiency formula in terms of these variables, $\eta_{asymptotic} = \frac{V_{\text{avg}} - 0.5\Delta V}{V_{\text{avg}} + 0.5\Delta V}$. This means that $\eta_{asymptotic} \uparrow$ as both $V_{avg} \uparrow$ and $\Delta V \downarrow$. Figure 3-4 shows the asymptotic efficiency vs cell voltage differential curves for the 2S, 4S, and 8S balancers. Figures 3-5 and 3-6 show the efficiency vs cell voltage differential as measured for the 2S and 4S PCBs, respectively, and compare that to the asymptotic efficiency.

The asymptotic efficiency does not take into account the power consumed by the control circuit. Using the asymptotic efficiency formula, $\lim_{\Delta V \to 0} \eta_{asymptotic} = 1$. In actuality, $\lim_{\Delta V \to 0} \eta = 0$. The control circuit is of course necessary to operate the power circuit, so its power consumption should be taken into account. Figure 3-7 shows the efficiency of a 10 mA/mV (corresponds to a 4 A balance current at 0.4 V cell differential) 2S balancer balancing cells that are at an average voltage of 3.2 V. Two scenarios are considered: one where the control circuit consumes a constant 0.1 W power and another where the control circuit consumes 1 W. For a high control power consumption of 1 W, the circuit efficiency is 24.2 % for 10 mV cell voltage differential and 38.9 % for 20 mV differential. For a low control power consumption of 0.1 W, the circuit efficiency is 76.0 % for 10 mV differential and 86.0 % for 20 mV differential. In practical use, hysteric control would be used to shut off balancer operation when the cell voltage differential is low, and resume operation when the differential is higher.

Large differences in the MOSFET timings may negatively affect the circuit efficiency by violating the zero current imbalance assumption that was used above. For example, the four MOSFETs in Table 3.1 were tested on the 2S balancer PCB. The current imbalance was over 0.9 A at the low switching frequency of 22 kHz! One tip-off that something was not quite right was that even in the large TO-220SM package, the Toshiba TJ200F04M3L was very hot, much hotter than the other MOSFETs in the circuit. This was confirmed with a thermal imaging camera. The TJ200F04M3L is a good MOSFET with lots of merit, but unsuitable for this particular switching application. It has a slow

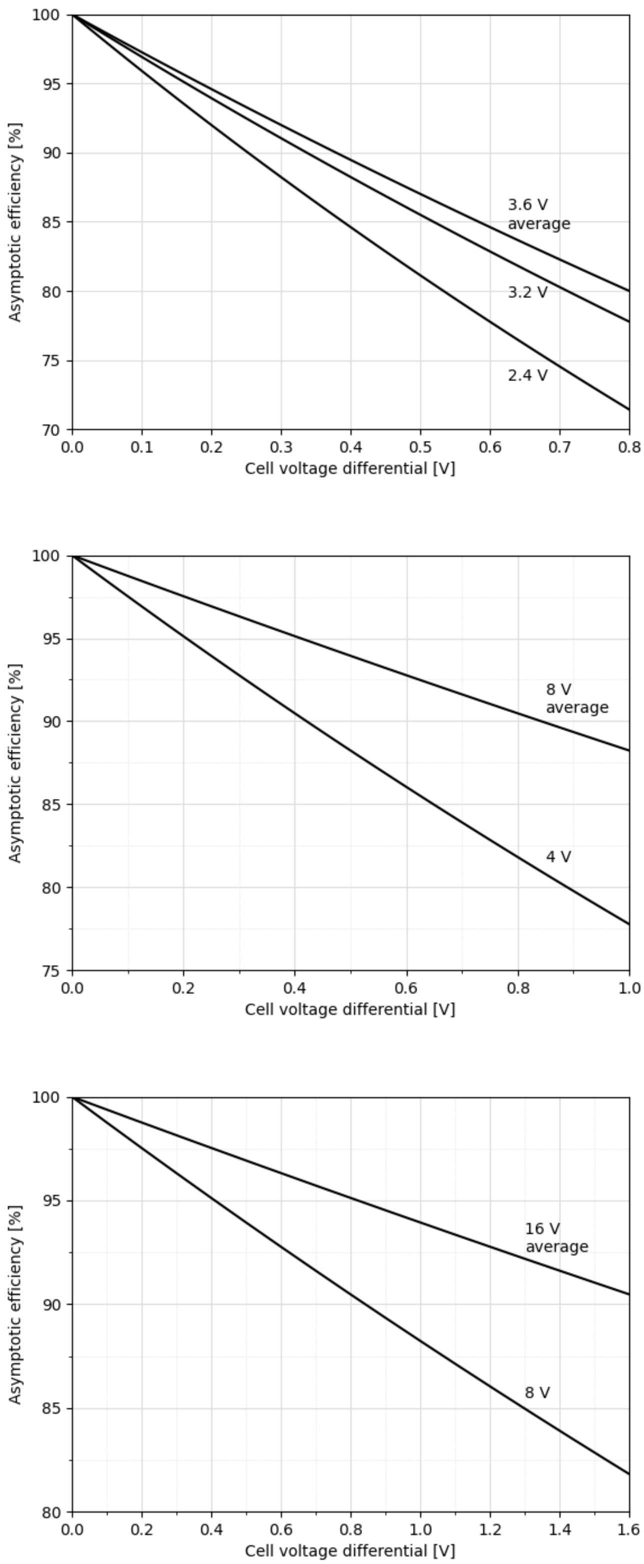

Figure 3-4: Asymptotic efficiency vs cell voltage differential for the 2S (top), 4S (middle), and 8S (bottom) balancers at various average cell voltages.

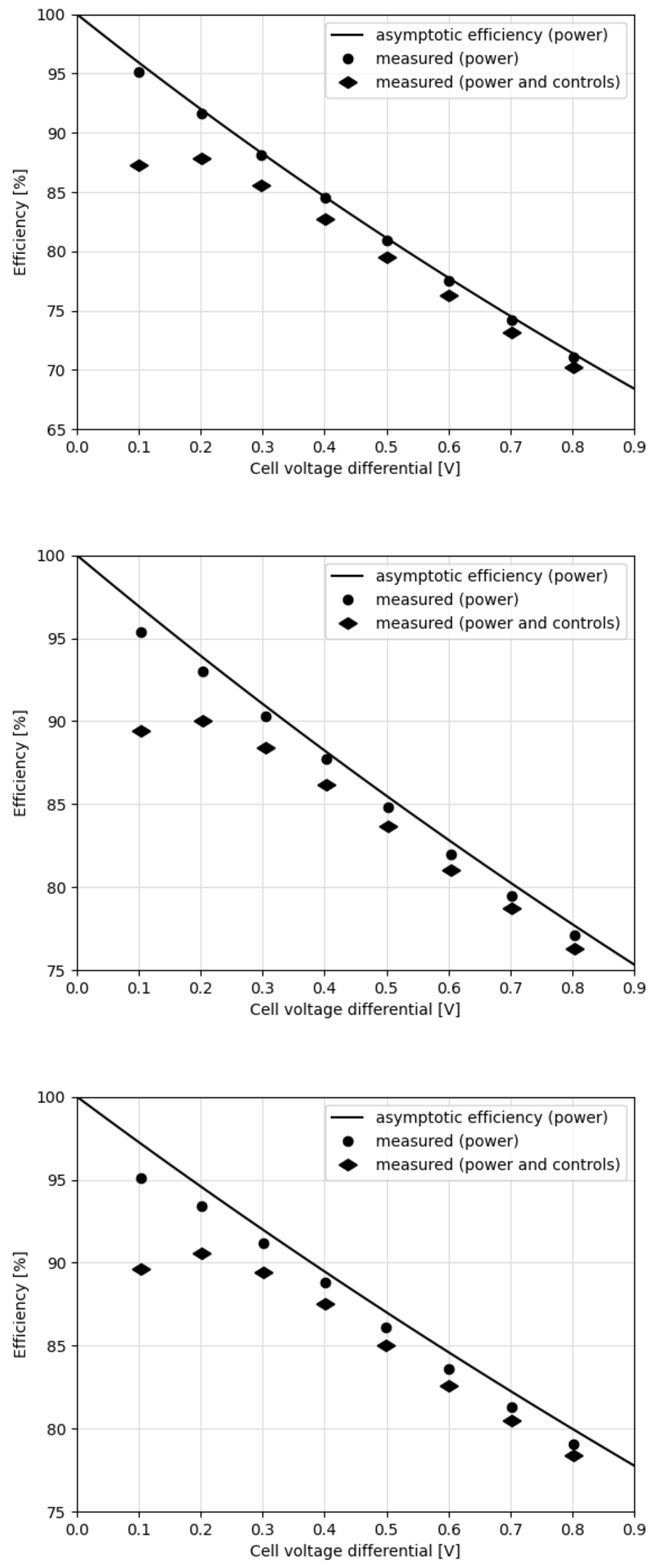

Figure 3-5: Efficiency vs cell voltage differential for the 2S PCB at average cell voltages of 2.4 V (top), 3.2 V (middle), and 3.6 V (bottom).

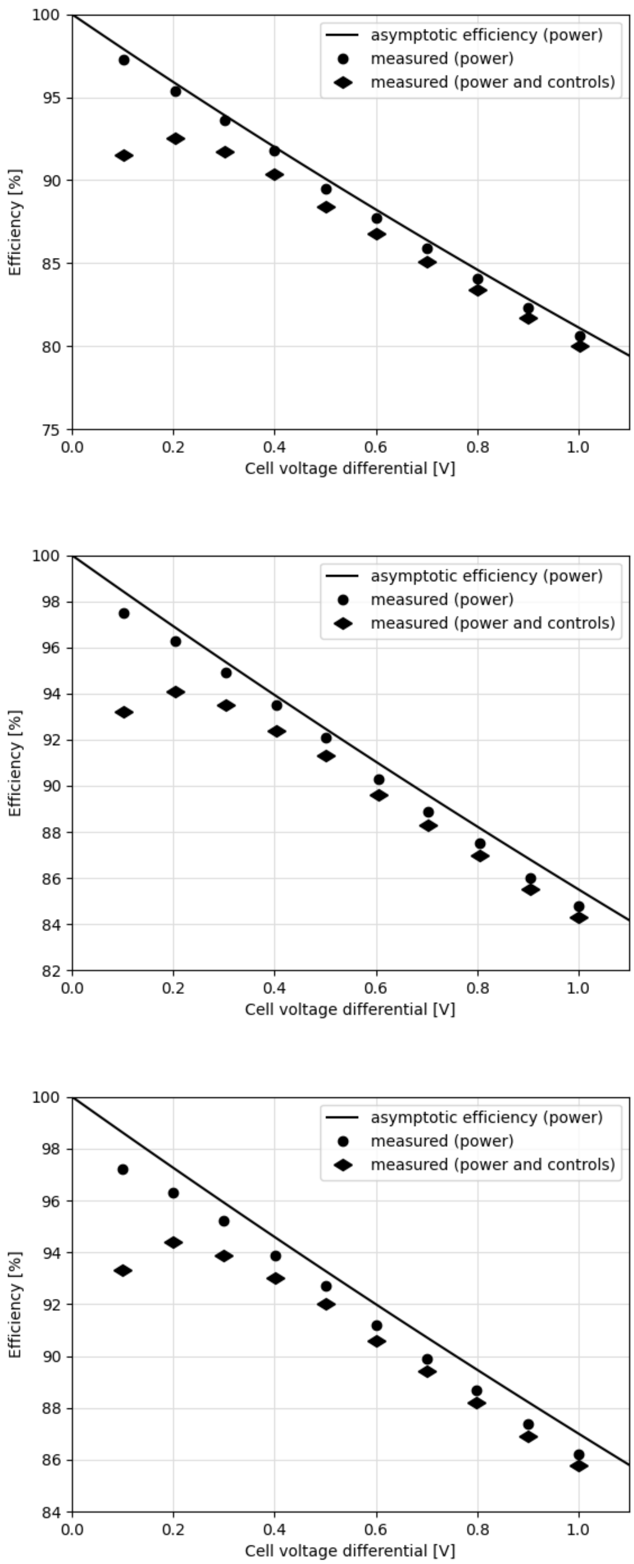

Figure 3-6: Efficiency vs cell voltage differential for the 4S PCB at average cell voltages of 4.8 V (top), 6.4 V (middle), and 7.2 V (bottom).

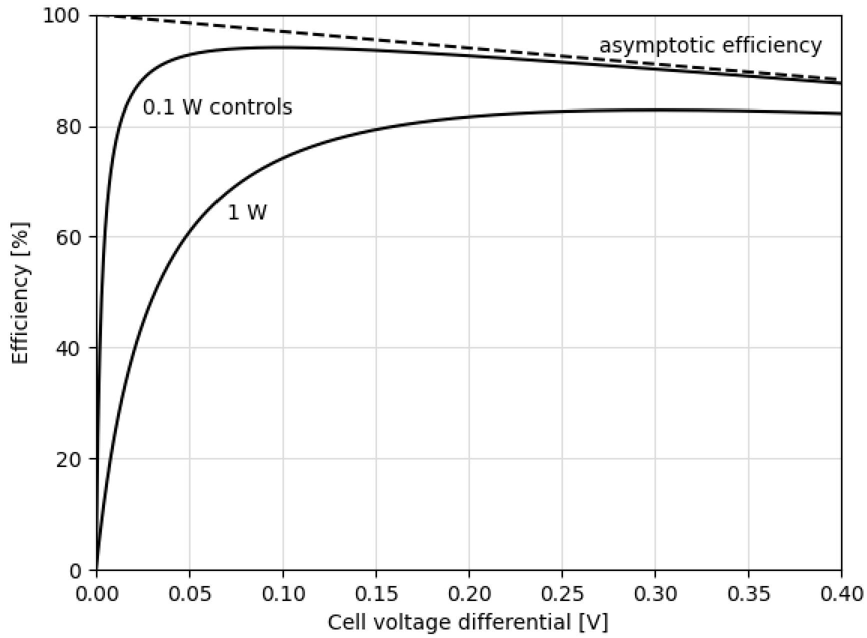

Figure 3-7: Efficiency vs cell voltage differential for a 10 mA/mV balancer balancing 3.2 V cells for different control powers.

turn-off time of over 2 µs; that is 1-2 orders of magnitude slower than the other turn-on and -off times of the other MOSFETs in that selection. In summary, it is important to choose MOSFETs with comparable turn-on and turn-off times.

Table 3.1: Switching characteristics of selected MOSFETs without comparable timings.

Manufacturer	Part number	$t_{d(on)}$ [ns] (typical)	t_r [ns] (typical)	$t_{d(off)}$ [ns] (typical)	t_f [ns] (typical)
ON Semiconductor	NVMYS1D3N04C	15	22	48	16
Toshiba	TJ200F04M3L	29	14	1750	515
Nexperia	PSMNR70-30YLHX	28	51	61	45
Infineon	IPB180P04P403	48	31	72	81

3.4 Power Dissipation

The power dissipated in the power circuit (four MOSFETs, flying capacitor, bypass capacitor 1, and bypass capacitor 2) is $P_{diss} = P_{in} - P_{out} = V_{in}I_{in} - V_{out}I_{out}$, where V_{in}

is the higher of the two cell voltages (i.e., $V_{in} > V_{out}$). Let us again assume zero current imbalance such that $I_{in} = I_{out} \equiv I$; $P_{diss} = (V_{in} - V_{out})I$.

Where is that power being dissipated? Let us ignore any power dissipation in the bypass capacitors because those are in parallel with a cell, which has a large capacitance in comparison to itself. In addition, for simplicity, let us assume that all four MOSFETs have the same $R_{DS(on)}$ value so that the current flowing in state 1 is equal to that in state 2. In state 1, MOSFETs M1 and M3 are on and connect the flying capacitor in parallel with bypass capacitor 1. In state 2, M2 and M4 are on and connect the flying capacitor in parallel with bypass capacitor 2. Because the flying capacitor conducts current in both states, and each MOSFET conducts current in only one state, the power dissipated in the flying capacitor is twice that of an individual MOSFET for *equivalent* resistance.

The power dissipated in the power circuit is highly dependent on the cell voltage differential $V_{in} - V_{out} \equiv \Delta V$. In fact, it is proportional to the square of the voltage differential: $\frac{\Delta V^2}{R}$, where R can be thought of as the overall circuit resistance. For the 2S balancer, let us choose (arbitrarily) 0.4 V as the typical worst-case cell differential. For 8 A balance current at 0.4 V cell differential, $P_{diss} = 3.2\,\text{W}$. That is 0.64 W per device on average. A quick rule of thumb for a power device such as a MOSFET is that a heat sink is probably not needed if the power dissipation is less than 1 W. This means heat sinking is not strictly necessary for the 2S PCB. For the 4S balancer, it is logical to use 0.8 V as the reference cell differential since 0.4 V was for the 2S. For 14 A balance current at 0.8 V cell differential, $P_{diss} = 11.2\,\text{W}$. That is 2.24 W per device on average. It is important that the ESR of the flying capacitor be roughly the same as $R_{DS(on)}$ of the MOSFETs to avoid the power dissipation being concentrated in the flying capacitor. If necessary, one can parallel capacitors to lower the equivalent ESR of the flying capacitor.

Figure 3-8 shows a thermal circuit model for a device connected to a heat sink [29]. Let us use the model to estimate the necessary heat sinking for the 4S balancer. Let us use the Vishay SQD40031EL, On Semiconductor NVMYS1D3N04C, Infineon IPB180P04P403, and Nexperia PSMNR70-30YLHX as MOSFETs M1, M2, M3, and

M4, respectively. Let us use the Aavid 7106DG heat sink on P-channel M1 and P-channel M3. The drains of M2 and M3 are connected, so they 'share' a heat sink. Let us use a case-to-sink thermal resistance of $R_{\theta CS} = 0.5\,^{\circ}\mathrm{C/W}$, even though the 7106DG heat sink attaches directly to a surface mount pad and the thermal conductivity of solders is negligible [30]. Finally, let us use 25 °C for the ambient temperature.

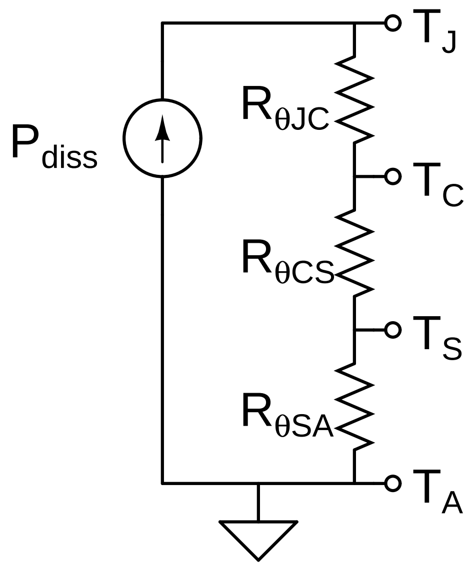

Figure 3-8: Thermal circuit model for a device connected to a heat sink.

The junction temperatures of M1 and M4 can be calculated as

$T_{J(M1)} = 25\,^{\circ}\mathrm{C} + (1.1 + 0.5 + 20\,^{\circ}\mathrm{C/W})2.24\,\mathrm{W} = 73.4\,^{\circ}\mathrm{C} \leq 175\,^{\circ}\mathrm{C}$

$T_{J(M4)} = 25\,^{\circ}\mathrm{C} + 1.5(42\,^{\circ}\mathrm{C/W})2.24\,\mathrm{W} = 166.1\,^{\circ}\mathrm{C} \leq 175\,^{\circ}\mathrm{C}$

The factor of 1.5 is a rough estimate to derate the T_{JA} specification for the copper pad not being $1\,\mathrm{in}^2$ [31]. To find the junction temperatures of M2 and M3, the following system of equations is solved

$$\frac{T_{J(M2)} - T_S}{1.12\,^{\circ}\mathrm{C/W} + 0.5\,^{\circ}\mathrm{C/W}} = 2.24\,\mathrm{W}$$
$$\frac{T_{J(M3)} - T_S}{1.0\,^{\circ}\mathrm{C/W} + 0.5\,^{\circ}\mathrm{C/W}} = 2.24\,\mathrm{W}$$
$$\frac{T_S - T_A}{16\,^{\circ}\mathrm{C/W}} = 4.48\,\mathrm{W}$$

$\Longrightarrow T_S = 94.9\,^{\circ}\mathrm{C}, \quad T_{J(M2)} = 98.5\,^{\circ}\mathrm{C} \leq 175\,^{\circ}\mathrm{C}, \quad T_{J(M3)} = 98.2\,^{\circ}\mathrm{C} \leq 175\,^{\circ}\mathrm{C}$

The above calculations show that two heat sinks for the 4S PCB can sufficiently dissipate the heat *provided* enough capacitors are paralleled for the flying capacitor so that the losses are not concentrated there. In general, for a given power dissipation and target junction temperature, Figure 3-9 can be used to find the maximum allowable junction-to-ambient thermal resistance. For the 8S balancer operating at a worst-case differential of 1.6 V, all four MOSFETs will need to be heat sinked.

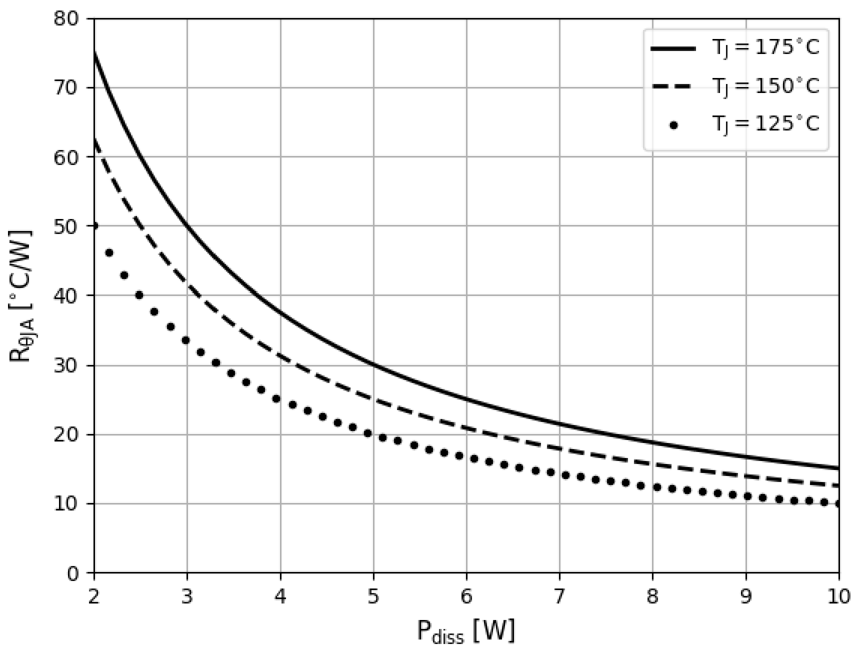

Figure 3-9: Junction-to-ambient thermal resistance vs power dissipation for junction temperatures of 125 °C, 150 °C, and 175 °C.

3.5 PCB Design

Figure 3-10 shows a photograph of the perfboard based 2S balancer prototype. Similar perfboard based prototypes were made for the 4S balancer and the tester circuit described in Chapter 5. For scale, the perfboard is 2" x 1.75". A handmade prototype is appropriate for rapid prototyping and allows for experimentation/testing. Screw terminal blocks were used for the balancer circuits to allow for easy testing of the capacitors.

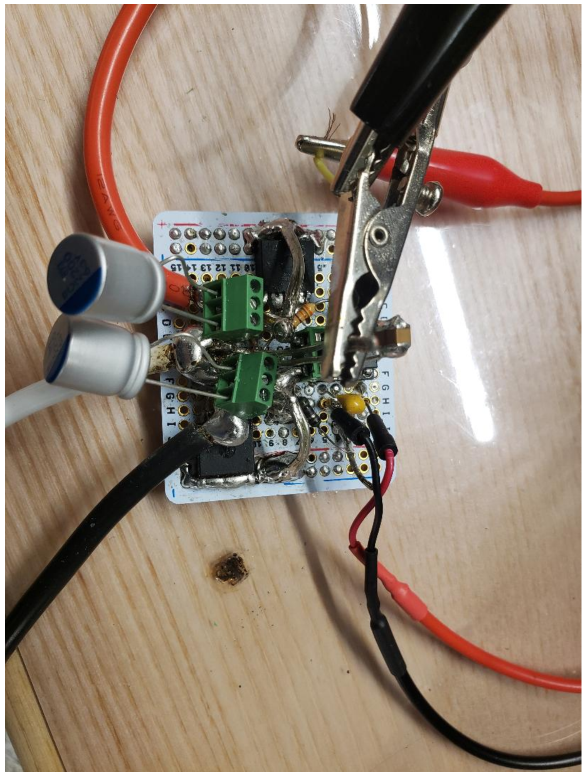

Figure 3-10: Photograph of the perboard based 2S prototype.

PCBs (printed circuit boards) for the 2S and 4S balancers were designed using an electronic design automation (EDA) software to verify the initial results of the perfboard-based prototypes. A PCB offers several advantages over a hand-made prototype. Copper can also be controlled to a few thousandths of an inch. This allows for precise control of parasitics and improved repeatability of results. Once the PCB is designed, the circuit can be mass produced quickly and cheaply.

Figures of the resulting 2S and 4S PCB designs are shown in Appendix C. The 2S PCB is 2" x 2" and the 4S PCB is 2.2" x 2". The overall PCB design is a 2-layer board with the control circuit on one side of the board and the power circuit on the other. There exist three vias: one via to connect ground on both sides of the board and the other two vias to connect the gate signal to the gates of the four MOSFETs. A copper weight of 2 oz (thickness 2.8 mils) was chosen instead of the more common 1 oz to allow for higher ampacity. The higher copper thickness should also be more durable and allow for multiple solder/desolder rework. In addition, there were several other design goals: to make the board both hand solder and testing friendly and to minimize parasitic resistances.

The PCB is intended to be hand soldered, so pads were made large. Tiny package sizes were also avoided. For example, on the control side, the smallest flat chip package size used is 0805 (imperial). 0805 size is 0.08” x 0.05” (i.e., 80 mil x 50 mil) nominally. For the 8-pin dual op-amp, the larger SOIC package (nominal body size of 4.9 mm x 3.9 mm) is used instead of the also common VSSOP (3.0 mm x 3.0 mm). On the power side, MOSFETs no smaller than 5 mm x 6 mm are used, though there exists many quality power MOSFETs in the 3 mm x 3 mm package size. At these larger sizes, there should not be a need for a microscope, although tweezers can be helpful.

To help make the board testing friendly, there are 4 testpoint pads included to help in debugging/verification of key signals. The testpoint pads are large enough to fit a surface mount testpoint attachment such as the Keystone 519xTR series. Testpoint 1 is located at the inverting input of the multivibrator op-amp. Testpoint 2 is located after the output resistor of the comparater op-amp. Testpoints 3 and 4 are located before and after the gate resistor, respectively. Testpoint 1 can be used to view the multivibrator timing capacitor waveform. Testpoint 2 can be used to view the pulse wave after the op-amp output resistor. Testpoint 2 can also be used to bypass the op-amp based oscillator signal and instead inject an external oscillator signal, such as from a signal generator (although make sure to desolder the resistor feeding back to the op-amp). Testpoints 3 and 4 can be used to view the gate signal before and after the gate resistor.

The flexibility of accepting components of slightly different package sizes is important for testing in the lab environment. A ‘flexible’ board helps avoid the wasteful production of a multitude of almost identical boards. For example, for the designed PCBs, the P-channel MOSFET pad can fit both the TO-263 and TO-252 packages. The flying capacitor pad can fit three 2917 size packages in addition to one 1210 size package, or six 1210 size packages. The bypass capacitor pad can fit two 2917 size packages in addition to one 1210 size package, or four 1210 size packages. The capacitor pads are surface mount, but of course through-hole capacitors can be soldered on as well. For example, aluminum polymer capacitors in a thin radial, can package can fit. The gate resistor pad can fit anything from 1206 size to the much larger 2512 size if a lot of power dissipation is needed. Other popular package sizes within that range include 1210 and 2010.

Other testing friendly features include the use of a dual op-amp to allow for the generation of the more generic pulse wave, even though a square wave would gate the MOSFETs fine. There exist independent supplies for the op-amp and gate driver, even though it is possible to run both off of the same supply. Additionally, there is a hole (M3 size) in each corner of the PCB that can be used for PCB standoffs.

Minimizing parasitic resistance was another major design goal, especially on the power side of the board. It would be pointless to use MOSFETs with sub $5\,\mathrm{m\Omega}$ on-state resistance if the resistance of the trace interconnects are equal or higher. For a target $1\,\mathrm{m\Omega}$ trace resistance, the required thickness for a 1" long, 2 oz copper trace is 252 mil at 35 °C (25 °C ambient temperature and 10 °C temperature rise) and 313 mil at 100 °C [32, 33]. A lot of copper on the power side also allows for better heat dissipation. On the control side, a ground pour was used as ground is by far the most common connection point. In addition to facilitating short connections, the ground pour can also help to provide shielding. In general, a high percentage of copper by area has the bonus of less etching chemicals used during the manufacturing process in addition to there being less copper as a waste product. The resulting PCBs are predominately copper by area (figures in Appendix C); the control side of the board is 77 % copper for the 2S PCB and 79 % for the 4S. The power side is 68 % copper for the 2S PCB and 75 % for the 4S.

A 'revision 2' for each was made after testing the first version of the PCB. The main improvement in revision 2 is the modification of the MOSFET footprints to eliminate the handlebar extensions on the drain pads. The handlebar shapes were initially chosen to increase the drain pad copper area to allow for improved heat dissipation. In reality, any additional heat sinking capability from them was small and they just got in the way of the layout. Note, the revision 2 PCBs have not yet been tested at the time of submission of this thesis.

Chapter 4

Cell Voltage Balancer: Simulation

4.1 PSpice Modeling

A PSpice model was developed to compare simulation results with measured experimental data from the Power Circuit 1 prototype. Figure 4-1 shows the resulting PSpice schematic. A single PSpice simulation with a frequency sweep of 10–50 kHz and a spacing of 1 kHz takes roughly 10–20 minutes on a 4th generation i7 quad-core processor, comparable to the time it takes to gather experimental data from the prototype.

4.1.1 MOSFETs

Infineon has sophisticated SPICE simulation models for many of their power MOSFETs. Their models include a temperature input, which we set to 40 °C. At low switching frequencies, as verified by an infrared thermometer, 40 °C is a satisfactory temperature.

4.1.2 Capacitors

Several manufacturers provide SPICE models on their website for their capacitors, although at varying levels of fidelity. Murata allows the user to specify the DC bias voltage and temperature. Similarly, Kemet allows the user to specify the bias voltage and temperature, as well as a center frequency. Panasonic, Rubycon, and Taiyo Yuden provide SPICE models with no adjustable DC bias or temperature. For ceramic capacitors, the

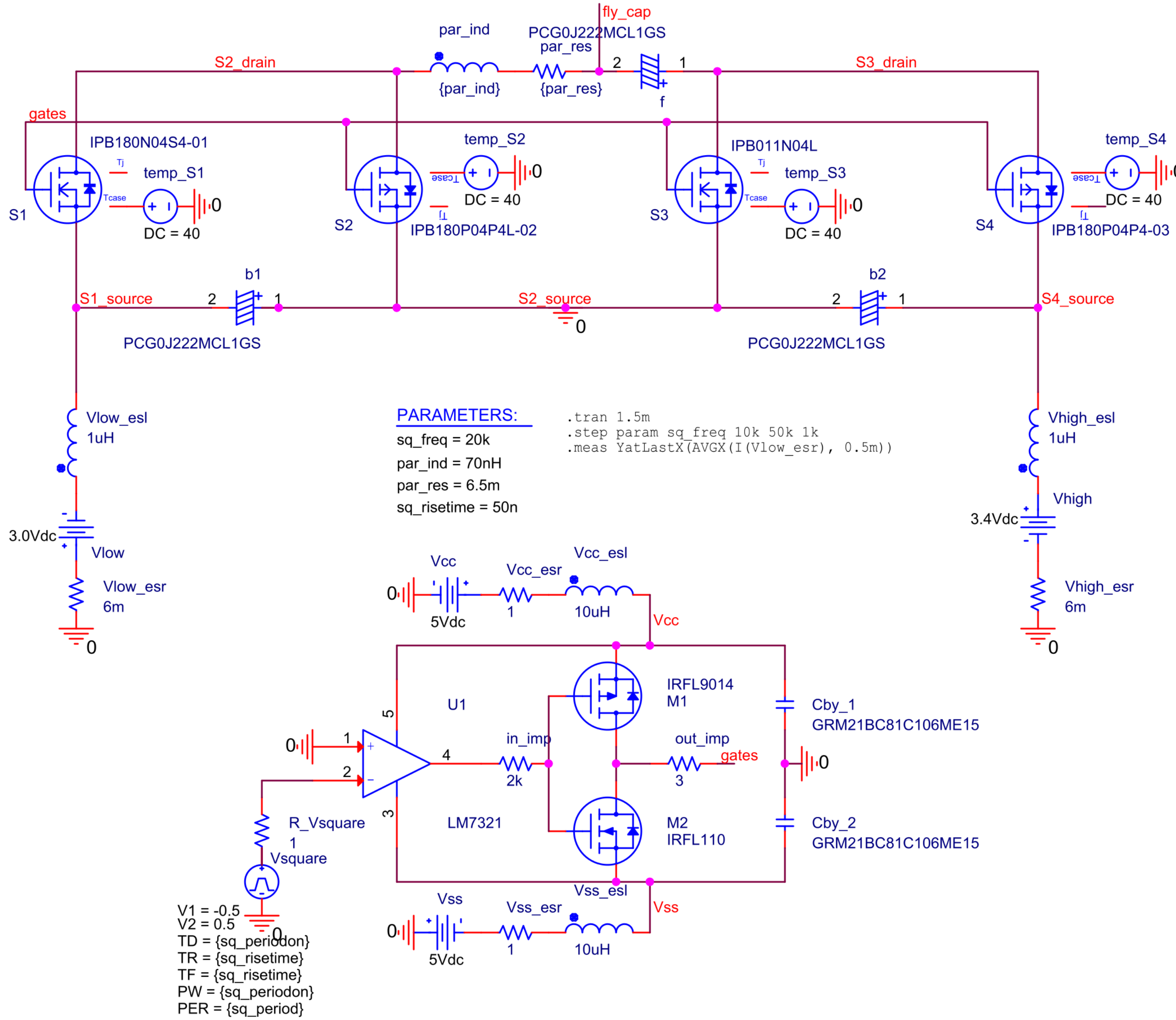

Figure 4-1: PSpice schematic of Power Circuit 1 with the aluminum polymer United Chemi-Con APSG160ELL222MJ20S as the flying and bypass capacitors.

bias voltage matters a lot, while for polymer and film capacitors, for example, it may not matter. AVX allows the user to specify a temperature from a set of discrete options: -55, 0, 25, 85, 105, and 125 (in °C). Some capacitors do not have a manufacturer-provided SPICE model. For simulation purposes, a SPICE model for a similar capacitor can be used instead. For example, United Chemi-Con does not provide a SPICE model for the APSG160ELL222MJ20S; the SPICE model for the Nichicon PCG0J222MCL1GS was used instead.

4.1.3 Gate Drivers

We used the IX3180G driver in many of our experiments, but ISYX does not have any SPICE models available on their website. Toshiba provides several PSpice models of their drivers, but not of the TLP5774. Toshiba has models for the TLP350H and TLP5702, potential substitutes for the TLP5774, but they are NMOS totem-pole output and not CMOS totem-pole output like the TLP5774. The Texas Instruments UCC27321 has a PSpice model which worked well when driving ideal capacitive loads; however, it failed to converge with the model of Power Circuit 1, even when relaxing the numerical constraints of the simulation.

PSpice has a model of the Microchip MIC4452, a CMOS totem-pole output driver, in their default library. The MIC4452 model worked great as long as the GND pin was connected to 0 V; thus, we had to add a DC offset to the input square wave voltage and subtract that same DC offset at the output of the driver. Numerically, this was fine, but to avoid this altogether, we made our own driver model. We used a TI LM7321 op-amp to drive a CMOS totem-pole output with $2\,\mathrm{k}\Omega$ resistance between the op-amp and the totem and $3\,\Omega$ output resistance representing approximately $1\,\Omega$ output resistance of the driver itself and an additional $2.2\,\Omega$ resistor on the board that was used in the experiments. We arrived at the input and output resistances by running simulations of Power Circuit 1 with two of our best capacitor models and making sure the peak balance current matched. Because switching losses are noticeable at frequencies above 40 kHz, we made sure the simulation results matched at these higher frequencies as well.

One thing we have observed is that a lot of the drivers we have simulated are numerically unstable. Instead, an almost ideal square wave can be used in place of a gate driver. The almost ideal square wave will tend to slightly overestimate the balance current at higher frequencies, but works well as the limiting case.

4.1.4 Parasitics

In addition to MOSFETs, capacitors, and gate drivers, the parasitic resistances and inductances greatly impact the performance of Power Circuit 1. We estimate roughly 1 µH of inductance from long power supply leads. We measured roughly 6 mΩ resistance from the power supplies to the terminal blocks of the bypass capacitors. The remaining circuit parasitic resistances and inductances were lumped together and put in series with the flying capacitor for simplicity of modeling. Lumped parasitic resistance and inductance were estimated and then tuned using the capacitor models for the Murata GRM32ER60J227ME05 and Kemet T530X477M006ATE004, as those were considered our best capacitor models.

4.2 Results

The PSpice simulation results matched the experimental data of the Power Circuit 1 prototype with varying levels of success. Figure 4-2 shows the results for select ceramic and aluminum polymer capacitors. Similarly, Figure 4-3 shows the results for select tantalum and tantalum polymer capacitors. In general, the shapes of the resulting curves matched. However, for some capacitors, the simulation results were off in terms of peak balance current and/or resonant frequency.

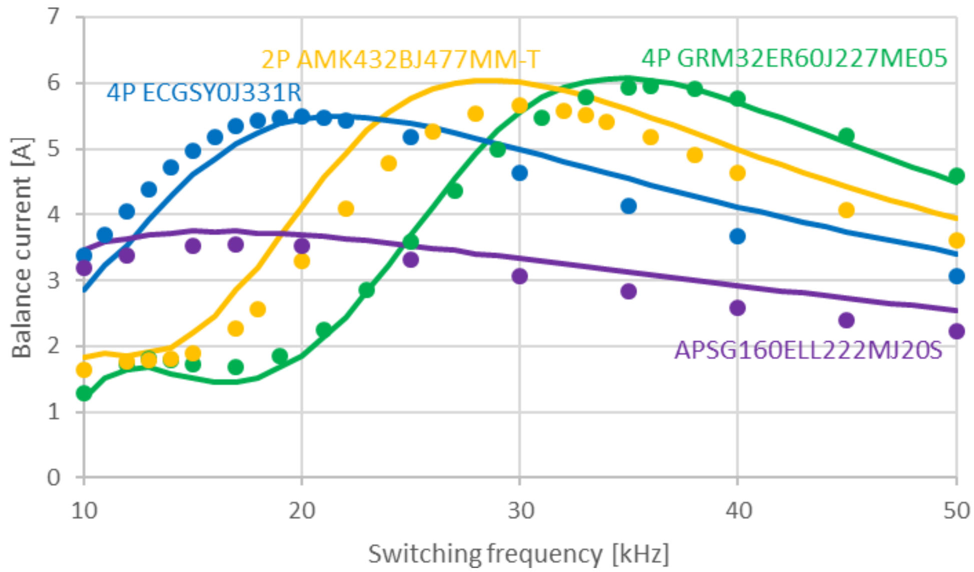

Figure 4-2: PSpice simulation results (solid line) compared with experimental data (dotted line) of Power Circuit 1 prototype for select ceramic and aluminum polymer capacitors.

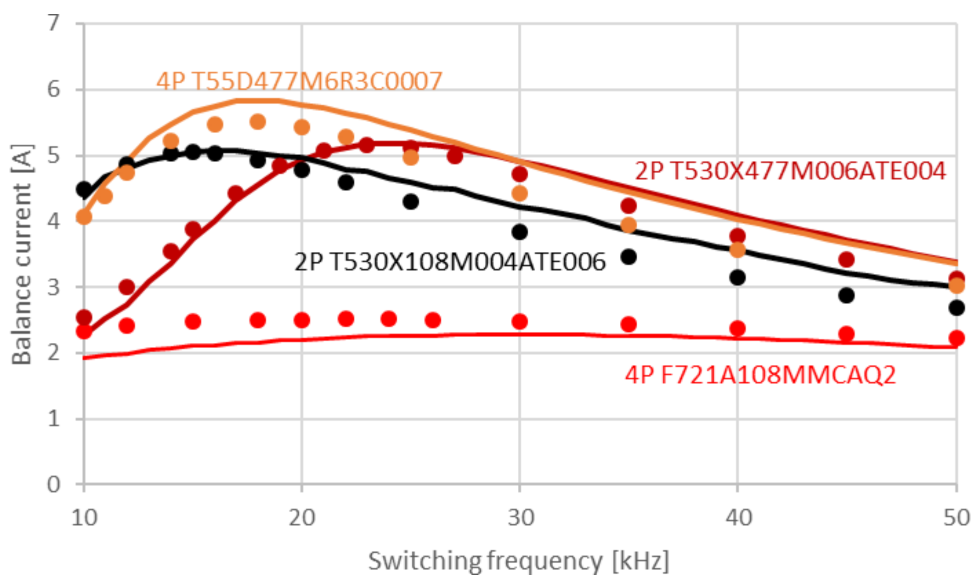

Figure 4-3: PSpice simulation results (solid line) compared with experimental data (dotted line) of Power Circuit 1 prototype for select tantalum and tantalum polymer capacitors.

Chapter 5

Cell Tester

Figure 5-1 shows the inductor based cell voltage balancer concept and an implementation for a 2S balancer using both N- and P-channel high-threshold MOSFETs. Compared to the charge pump based balancer, there are a few key differences. For example, the inductor based balancer uses an inductor as the intermediate storage element instead of a flying capacitor. Moreover, two switches are used instead of four. Like the charge pump based balancer, each cell/battery has a bypass capacitor across it. A failure mode that exists in the inductor based balancer that is not present in the charge pump based balancer is if the control circuit were to fail and the output got ‘stuck’ in one state (i.e., 0 % or 100 % duty cycle), a short circuit would occur. In [34], Freescale Semiconductor demonstrates a 12S balancer for a Nickel–metal hydride battery based on this inductor based topology.

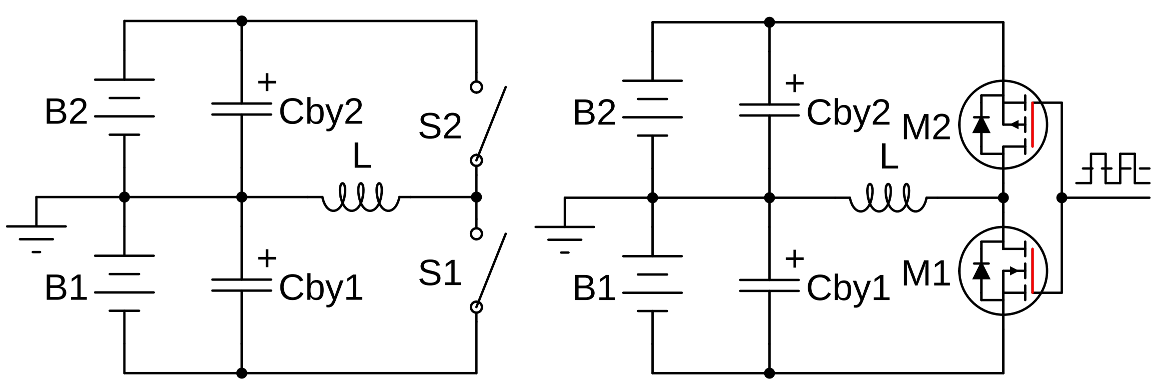

Figure 5-1: Inductor based balancer concept (left) and 2S balancer (right).

The inductor based balancer can be slightly modified to become a tester circuit as shown in Figure 5-2. The goal of the tester circuit is to be able to do constant current/voltage charge/discharge of the 'cell under test' B0. Cells B1 and B2 are 'buffer cells'. As the name implies, the buffer cells should have a larger capacity than the cell under test. In between the cell under test and the inductor is a current sensor. The DAQ takes in as analog input the current (the current sensor voltage) and the voltage of the cell under test. The DAQ also takes in the voltages of the buffer cells to ensure they are in the safe operating area. To attenuate any high frequency noise, each analog input should be passed through a low-pass RC filter before connection to the DAQ; this is a must! The DAQ implements feedback control and outputs an analog signal to change the duty cycle. It is important that the two MOSFETs be attached to a heat sink; as opposed to a balancer with natural negative feedback present to coerce the cell voltage differential, and thus power dissipation, low, a tester can be on for several hours or even days at high power. Table 5.1 lists the major components used for the cell tester.

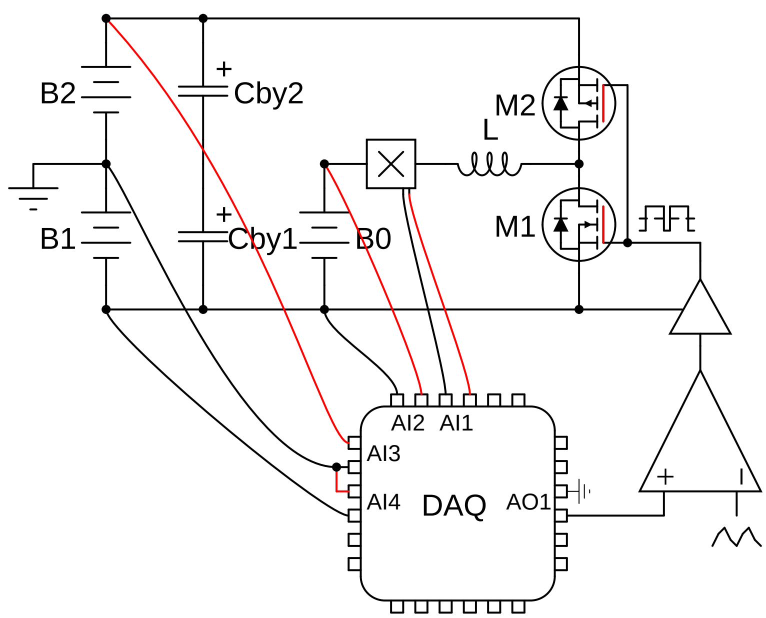

Figure 5-2: Inductor based cell tester.

The source code for the cell tester is displayed in Appendix E. The most important part of the code is the PID feedback control; much of the other code relates to the GUI, intricacies of the DAQ library, writing to CSV, logging, etc. For a useful reference, Brett

Table 5.1: Components cell tester.

Description	Manufacturer	Part number	Value
high threshold NMOS	Infineon	IPB180N04S401	1.3 mΩ
high threshold PMOS	Infineon	IPB180P04P403	2.8 mΩ
heat sink	Ohmite	DA-T268-301E	4.3 °C/W
inductor	Coilcraft	AGM2222-512ME	5.1 µH, 1.1 mΩ
bypass capacitor 1/2	Panasonic	ECG-SY0J331R	330 µF, 9 mΩ (3 in parallel used)
current sensor	LEM	LAH 25-NP	±0.3 % error
data acquisition system (DAQ)	Measurement Computing	USB-1608GX-2AO	8 differential analog inputs, 2 analog outputs
dual op-amp (multivibrator and comparator)	Texas Instruments	OPA2192	±5 µV offset
gate driver	Toshiba	TLP5774	optocoupler

Beauregard, the creator of the PID Library for Arduino, published a series of blog posts explaining his implementation of PID control [35].

Chapter 6

Conclusions and Future Work

This thesis has demonstrated the design, evaluation, and simulation of charge pump based cell voltage balancers. The designs of the 2S, 4S, and 8S balancer are described. Evaluation of the balancers are discussed and experimental results of the 2S and 4S perfboard prototype and PCBs are shown. Simulations in PSpice of the 2S prototype were developed and the simulation results closely matched with the experimental data. Additionally, an inductor based cell tester prototype circuit was discussed and the source code of the accompanying program is shown.

Future work includes making a complete 4S balancer module, with two 2S balancers and one 4S balancer on one PCB. This includes miniaturizing the existing circuits to make them more practical. Several changes can be made with this goal in mind. On the power side, the use of the large TO-263 MOSFET package for the P-channel MOSFETs can be eliminated in favor of the smaller TO-252 package. The flying capacitor pad can be shrunk to allow for two 2917 size packages in addition to one 1210 size package. The bypass capacitor pads can be shrunk to allow for one 2917 size packages in addition to one 1210 size package. On the control side, the dual op-amp can be replaced with a single, as a square wave is satisfactory to operate the circuit. The 0603 size package can be used in place of the 0805 size package where the power level is appropriate. For example, resistors $10\,\mathrm{k\Omega}$, and the bypass capacitors for the op-amp, can likely be 0603 size. The gate resistor can be shrunk from 2512 to 2010 package size. Last, but not least, the testpoint pads can be eliminated, or miniaturized to a size as small as 40 mil.

The ultimate goal of our work is to produce a complete 8S balancer module. In general, a complete NS balancer (consisting exclusively of charge pump based balancers) requires N-1 balancers and thus 4(N-1) switches and 3(N-1) capacitors. Figure 6-1 shows the power circuit of a complete 8S balancer. The 8S balancer consists of four units of Power Circuit 1 operating at 2S voltage on the first level, two units of Power Circuit 2 operating at 4S voltage on the second level, and one unit of Power Circuit 3 operating at 8S voltage on the third level. In other words, each balancer on the first level balances two 1S batteries, each balancer on the second level balances two 2S batteries, and the balancer on the third level balances two 4S batteries.

A 16S balancer PCB that exclusively uses N-channel MOSFETs would also be interesting to design. In addition to designing more circuits/PCBs, other work can be done. Tests on actual cells, instead of a power supply and electronic load combination (or a bidirectional power supply), should be run to characterize the dynamic performance of the balancers. Additional testing of the inductor based tester circuit is needed. Moreover, additional SPICE modeling and simulations can be carried out on the other power circuits.

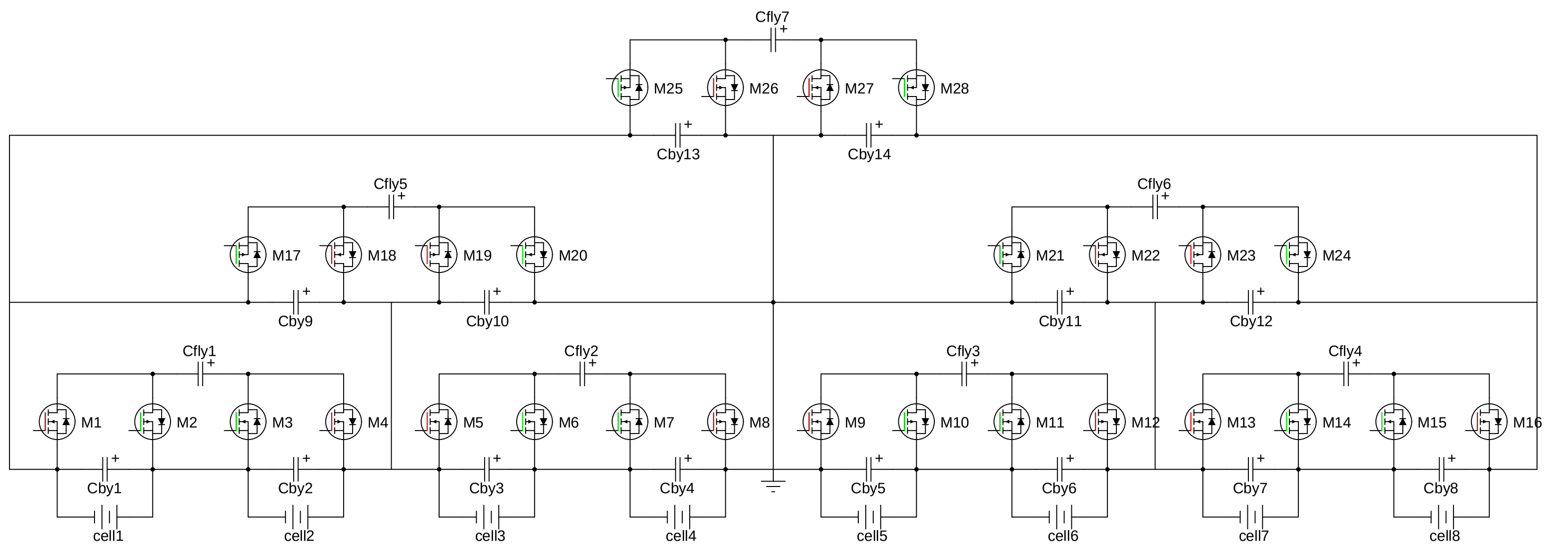

Figure 6-1: 8S multilevel balancer.

Appendix A

MOSFET and Capacitor Selection Tables

A.1 MOSFET Selection Table

Table A.1 shows a subset of the MOSFET table used in the component selection process. This is by no means an exhaustive list of power MOSFETs. Please confirm any specs before selecting/using!

Many of these specs are at $T_J = 25\,°\mathrm{C}$ unless otherwise specified. Note, a V_{drive} of 4.5 V is used for low-threshold MOSFETs, and 10 V is used for high-threshold.

Table A.1: MOSFET selection table.

Manufacturer	Part number	V_{DS} [V] (max)	V_{GS} [V] (max)	$R_{DS(on)}$ [mΩ] (max) @ $V_{GS} = V_{drive}$	Q_g [nC] (typical) @ $V_{GS} = V_{drive}$	V_{GS} [V] (typical) @ I_D = 5 A	t_{on} [ns] (typical)	t_{off} [ns] (typical)	$R_{\theta JC}$ [°C/W] (max)	$R_{\theta JA}$ [°C/W] (max)	Package	Approximate dimensions [mm x mm]	Gate location
low threshold N-channel													
Alpha & Omega	AOL1404	20	± 12	4.0	36	1.5	15	88	2.5	60	UltraSO-8	5 x 6	right
Alpha & Omega	AOUS66414	40	± 20	3.2	24	2.7	15.5	52.5	1.35	50	UltraSO-8	5 x 6	right
Infineon	IAUC120N04S6L008	40	± 16	1.1	47	2.1	12	70	1.0	50	TDSON-8	5 x 6	right
Infineon	IPD90N03S4L02	30	± 16	2.6	52	2.3	23	75	1.1	40	TO-252	7 x 11	left
Infineon	IRLR6225	20	± 12	4.0	48	1.5	46.7	115	2.0	50	TO-252	7 x 11	left
Nexperia	PH2925	25	± 10	3.0	92	1.1	110	372	2		LFPAK56	5 x 6	right
Nexperia	PSMNR58-30YLH	30	± 20	0.9	55	2.1	99	115	0.45	42	LFPAK56E	5 x 6	right
Nexperia	PSMNR70-30YLH	30	± 20	1.1	46	2.0	79	106	0.56	42	LFPAK56	5 x 6	right
Nexperia	PSMNR90-40YLH	40	± 20	1.2	54	2.3	91	92	0.45	42	LFPAK56E	5 x 6	right
On Semiconductor	FDB9403L-F085	40	± 20	1.6	89	2.2	156	399	0.45	43	TO-263	10 x 15	left
Vishay	SQJQ100EL	40	± 20	1.5	70	2.5	84	90	1	50	PowerPAK 8 x 8L	8 x 8	left
high threshold N-channel													
Alpha & Omega	AOL1440	25	± 30	5.2	33	5.8	31	23.5	2	55	UltraSO-8	5 x 6	right
Infineon	IPB180N04S401	40	± 20	1.3	135	4.2	59	79	0.8	40	TO-263	10 x 15	left
Infineon	IPD100N04S402	40	± 20	2.0	91	4.3	35	50	1.0	40	TO-252	7 x 11	left
On Semiconductor	FDD9409-F085	40	± 20	3.2	42	4.4	45	56	1	52	TO-252	7 x 11	left
On Semiconductor	NVMYS1D3N04C	40	± 20	1.15	75	4.1	37	64	1.12	39	LFPAK4	5 x 6	right
Toshiba	TPH2R608NH	75	± 20	2.6	72	4.2	41	71	0.88	50	SOP Advance	5 x 6	right
Vishay	SQJQ144AE	40	± 20	0.9	116	4.0	44	59	0.25	44	PowerPAK 8 x 8L	8 x 8	left
low threshold P-channel													
Infineon	IPD042P03L3G	30	± 20	6.8	60	2.5	188	111	1	50	TO-252	7 x 11	left
Renesas	2SJ687	20	± 12	7.0	57	1.9	256	580	3.47	125	TO-252	7 x 11	left
Vishay	SQD40031EL	30	± 20	5.20	88	2.9	220	198	1.1	50	TO-252	7 x 11	left
Vishay	SQJ411EP	12	± 8	5.8	99	1.3	68	273	2.2	68	PowerPAK SO-8L	5 x 6	right
high threshold P-channel													
Infineon	IPB180P04P403	40	± 20	2.8	190	3.9	79	153	1.0	40	TO-263	10 x 15	left
Infineon	IPD90P03P404	30	± 20	4.5	100	4.2	45	90	1.1	40	TO-252	7 x 11	left

V_{drive} = 4.5 V for low threshold, 10 V for high threshold

A.2 Capacitor Selection Tables

Tables A.2 and A.3 show a subset of the 4 V and 6.3 V capacitor tables used in the component selection process for the 2S balancer. Tables A.4 and A.5 show a subset of the 10 V and 16 V capacitor tables used in the component selection process for the 4S balancer. Similar tables can be made with 20 V and 25 V capacitors for the 8S balancer. This is by no means an exhaustive list of capacitors. Please confirm any specs before selecting/using!

Table A.2: 4 V capacitor selection table.

Manufacturer	Part number	Voltage rating [V]	Capacitance [µF]	ESR [mΩ]	Package	Area [100 * in^2]	Volume [1000 * in^3]
			ceramic				
Taiyo Yuden	AMK432BJ477MM-T	4	470		1812	2.23	2.36
			tantalum				
AVX	TPME158K004R0015	4	1500	15	2917	4.85	8.2
Vishay	597D158X9004R2T	4	1500	15	3024	7.06	11.1
			tantalum polymer				
Kemet	T530X687M004ATE004	4	680	4	2917	4.85	8.2
Vishay	T55D687M004C0007	4	680	7	2917	4.85	5.92
Kemet	T530X108M004ATE006	4	1000	6	2917	4.85	8.2
			aluminum polymer				
Panasonic	ECG-SY0G471R	4	470	9	2917	4.85	5.72
Nichicon	PCG0G332MCL1GS	4	3300	11	radial, can	12.2	61
			supercapacitor				
Maxwell	BCAP0010 P300 X11	3	10e6	25	radial, can	12.2	151
Tecate Group	TPLH-3R0/100SS22X46	3	100e6	12	radial, can	58.9	1110
Taiyo Yuden	LIC1840RS3R8107	3.8	100e6	60	radial, can	39.5	653
Illinois Capacitor	DGH505Q5R5	5.5	5e6	80	radial, can	52.4	578

Table A.3: 6.3 V capacitor selection table.

Manufacturer	Part number	Voltage rating [V]	Capacitance [µF]	ESR [mΩ]	Package	Area [100 * in^2]	Volume [1000 * in^3]
			ceramic				
AVX	12106D107MAT2A	6.3	100		1210	1.23	1.36
Kemet	C1210C107M9PACTU	6.3	100		1210	1.23	1.36
Murata	GRM32EC70J107ME15L	6.3	100		1210	1.23	1.31
Samsung	CL32A107MQVNNNE	6.3	100		1210	1.23	1.36
Taiyo Yuden	JMK325ABJ107MM-P	6.3	100		1210	1.23	1.36
TDK	C4532X5R0J107M280KA	6.3	100		1812	2.23	2.72
Murata	GRM32ER60J227ME05L	6.3	220		1210	1.23	1.31
Samsung	CL32A227MQVNNNE	6.3	220		1210	1.23	1.36
Taiyo Yuden	JMK325ABJ227MM-T	6.3	220		1210	1.23	1.36
Taiyo Yuden	JMK325ABJ337MM-P	6.3	330		1210	1.23	1.36
			tantalum				
Vishay	594D108X06R3R2T	6.3	1000	30	2824	6.68	9.88
Vishay	597D108X96R3R2T	6.3	1000	20	3024	7.06	11.1
Vishay	592D228X96R3X2T20H	6.3	2200	55	5829	16.6	13.1
			tantalum polymer				
Vishay	T55D337M6R3C0007	6.3	330	7	2917	4.85	5.92
Kemet	T520Y477M006ATE010	6.3	470	10	2917	4.85	7.61
Kemet	T530X477M006ATE004	6.3	470	4	2917	4.85	8.2
Kemet	T530X477M006ATE005	6.3	470	5	2917	4.85	8.2
Panasonic	6TPF470MAH	6.3	470	10	2917	4.85	7.61
Vishay	T55D477M6R3C0007	6.3	470	7	2917	4.85	5.92
Kemet	T530X687M006ATE010	6.3	680	10	2917	4.85	8.2
Kemet	T545H158M006ATE035	6.3	1500	35	2924	6.77	5.35
			aluminum polymer				
Kemet	A700X227M006ATE007	6.3	220	7	2917	4.85	8.2
Panasonic	ECG-SY0J331R	6.3	330	9	2917	4.85	5.72
Panasonic	ECG-CY0J331R	6.3	330	15	2917	4.85	5.72
United Chemi-Con	APSC6R3ELL222MJB5T	6.3	2200	10	radial, can	12.2	62.4
Nichicon	PCG0J272MCL1GS	6.3	2700	12	radial, can	12.2	61
			aluminum electrolytic				
Rubycon	6.3ZLQ10000MEFC12.5X35	6.3	10000	10	radial, can	19	277

Table A.4: 10 V capacitor selection table.

Manufacturer	Part number	Voltage rating [V]	Capacitance [μF]	ESR [mΩ]	Package	Area [100 * in^2]	Volume [1000 * in^3]
			ceramic				
Kemet	C1210C107M8PACTU	10	100		1210	1.23	1.36
Murata	GRM32ER61A107ME20L	10	100		1210	1.23	1.31
Samsung	CL32A107MPVNNNE	10	100		1210	1.23	1.36
Taiyo Yuden	LMK325ABJ107MM-P	10	100		1210	1.23	1.36
TDK	C5750X5R1A107M280KC	10	100		2220	4.41	5.38
			tantalum				
AVX	F721A108MMCAQ2	10	1000	140	2824	6.68	5.28
			aluminum polymer				
Kemet	A755MS158M1AAAE013	10	1500	13	radial, can	12.2	62.4
			aluminum electrolytic				
Nichicon	UHW1A103MHD	10	10000	11	radial, can	31.2	411

Table A.5: 16 V capacitor selection table.

Manufacturer	Part number	Voltage rating [V]	Capacitance [μF]	ESR [mΩ]	Package	Area [100 * in^2]	Volume [1000 * in^3]
			ceramic				
Kemet	C1210C107M4PAC7800	16	100		1210	0.882	0.935
Taiyo Yuden	EMK325ABJ107MM-T	16	100		1210	1.23	1.31
			tantalum polymer				
Vishay	T59EE477M016C0025	16	470	25	2917	4.85	8.2
			aluminum polymer				
Nichicon	RNL1C102MDS1	16	1000	8	radial, can	7.79	65.9
United Chemi-Con	APSG160ELL102MH20S	16	1000	8	radial, can	7.79	65.9
Nichicon	RNL1C222MDS1	16	2200	8	radial, can	12.2	103
United Chemi-Con	APSG160ELL222MJ20S	16	2200	8	radial, can	12.2	103
			aluminum electrolytic				
Kemet	ESY108M016AH4AA	16	1000	26	radial, can	12.2	101
United Chemi-Con	ESMH160VSN104MA80T	16	100000	9	radial, can	149	4700

Appendix B

Oscillation Frequency Derivation for Op-amp Astable Multivibrator

This appendix derives the oscillation frequency for the op-amp astable multivibrator introduced in Section 2.7.

The oscillation period is the sum of the time T_1 that the op-amp is in state 1 (in steady-state) and the time T_2 that the op-amp is in state 2. Note, under the symmetric saturation voltage (i.e., $V_{OH} = -V_{OL} \equiv V_{SAT}$) assumption, T_2 will equal T_1 (i.e., the duty cycle is 50 %). One can solve the following equation to find T_1

$$V_C(T_1) = -\beta V_{SAT} = (\beta V_{SAT} - -V_{SAT})e^{-T_1/\tau} - V_{SAT}$$

Rearranging for T_1 and plugging in RC for τ

$$T_1 = -RC \ln\left(\frac{-\beta V_{SAT} + V_{SAT}}{\beta V_{SAT} + V_{SAT}}\right) = RC \ln\left(\frac{1+\beta}{1-\beta}\right)$$

The period is then simply

$$T = 2T_1 = 2RC \ln\left(\frac{1+\beta}{1-\beta}\right)$$

and thus the oscillation frequency is

$$f = \frac{1}{T} = \frac{1}{2RC \ln\left(\frac{1+\beta}{1-\beta}\right)}$$

Appendix C

2S and 4S PCBs

C.1 2S PCB

Figure C-1 shows the 2S PCB schematic and Table C.1 lists the 2S PCB components. Figures C-2 and C-3 show two different views of the control side of the 2S board; both revisions 1 and 2 are shown. Figures C-4 and C-5 show two different views of the power side; likewise, both revisions 1 and 2 are shown.

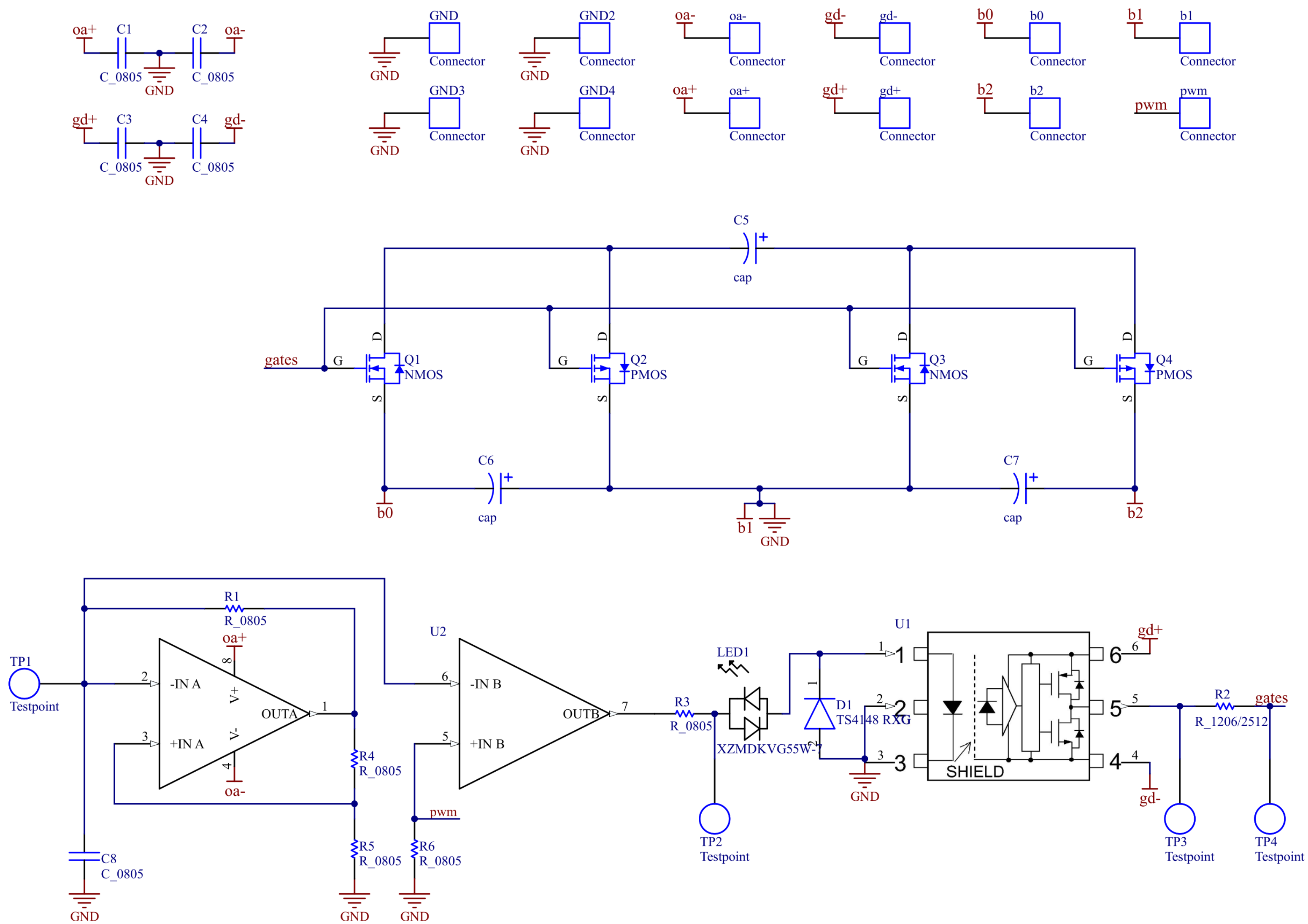

Figure C-1: Circuit diagram of 2S PCB. (Optocoupler gate driver image courtesy of TLP5774 datasheet.)

Table C.1: Components 2S PCB

Designator	Description
	Control side
C1	bypass positive supply op-amp
C2	bypass negative supply op-amp
C3	bypass positive supply gate driver
C4	bypass negative supply gate driver
C8	timing capacitor op-amp
D1	signal diode anti-parallel with optocoupler LED
LED1	indicator LED
R1	timing resistor op-amp
R2	gate resistor
R3	output resistor op-amp
R4	feedback divider resistor op-amp
R5	feedback divider resistor op-amp
R6	pull-down resistor pwm
U1	gate driver
U2	dual op-amp as multivibrator and comparator
	Power side
C5	flying capacitor
C6	bypass capacitor 1
C7	bypass capacitor 2
Q1	high threshold NMOS
Q2	low threshold PMOS
Q3	low threshold NMOS
Q4	high threshold PMOS

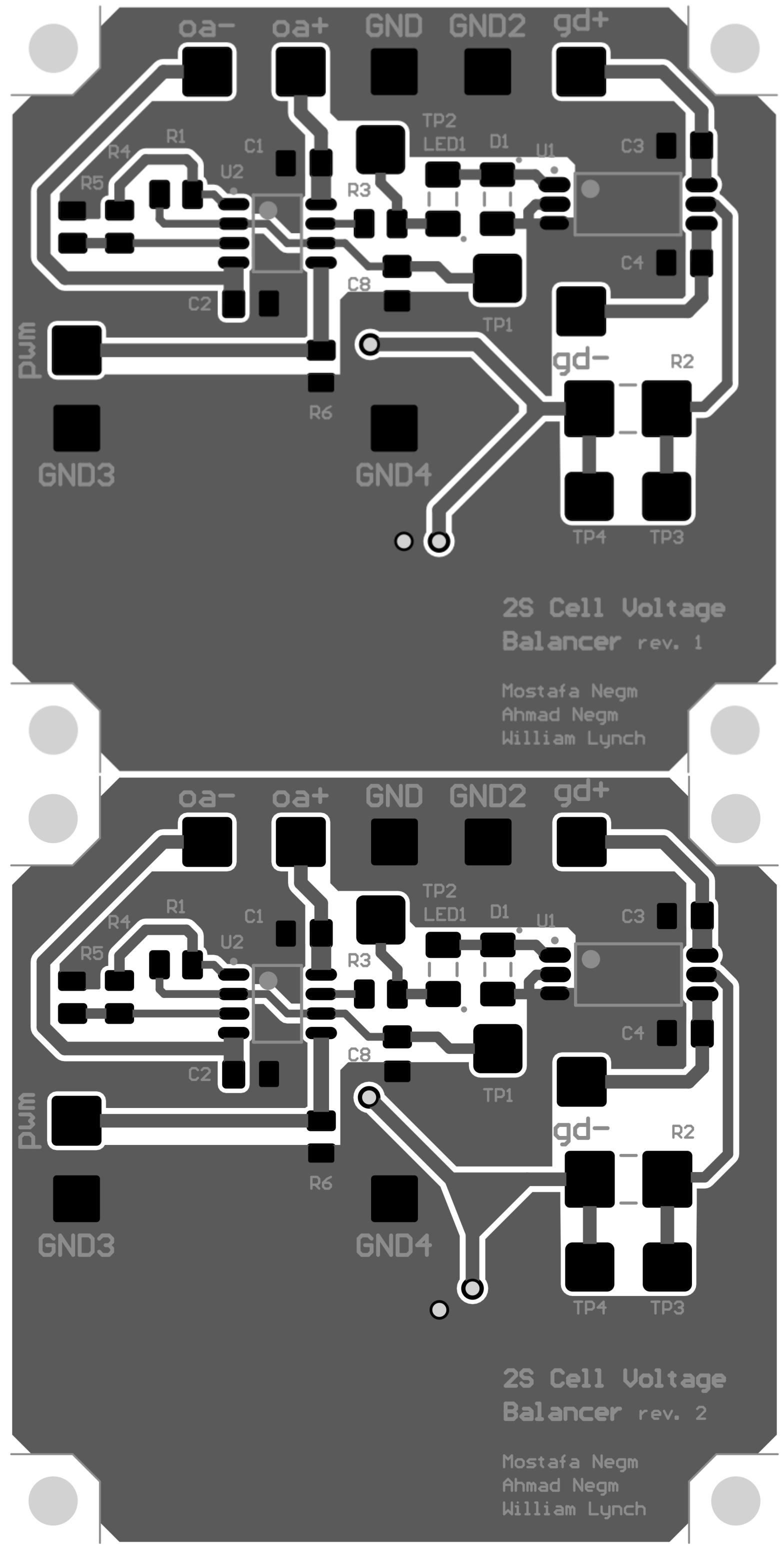

Figure C-2: 2D view of control side of 2S PCB: revisions 1 (top) and 2 (bottom).

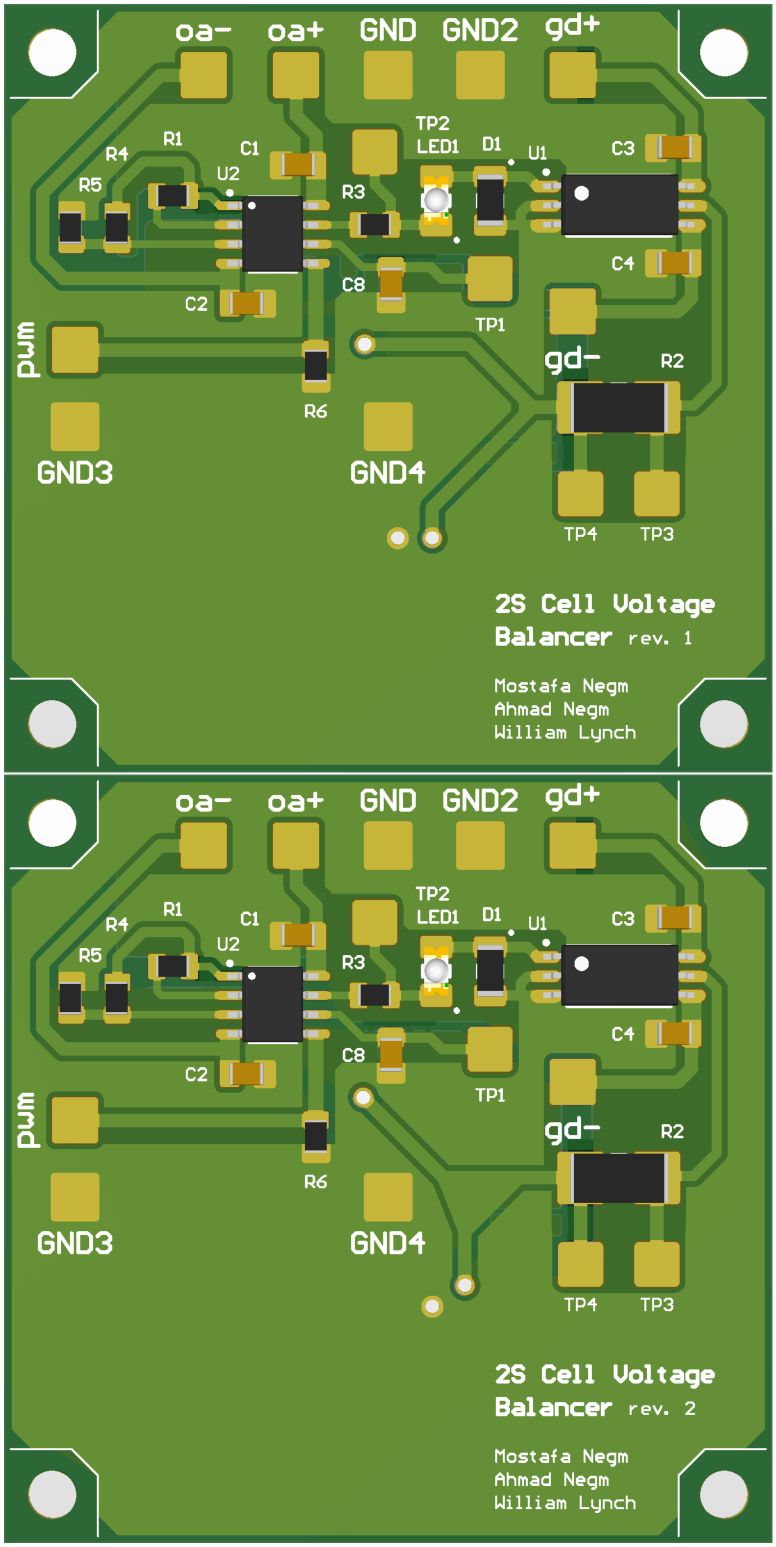

Figure C-3: 3D view of control side of 2S PCB: revisions 1 (top) and 2 (bottom).

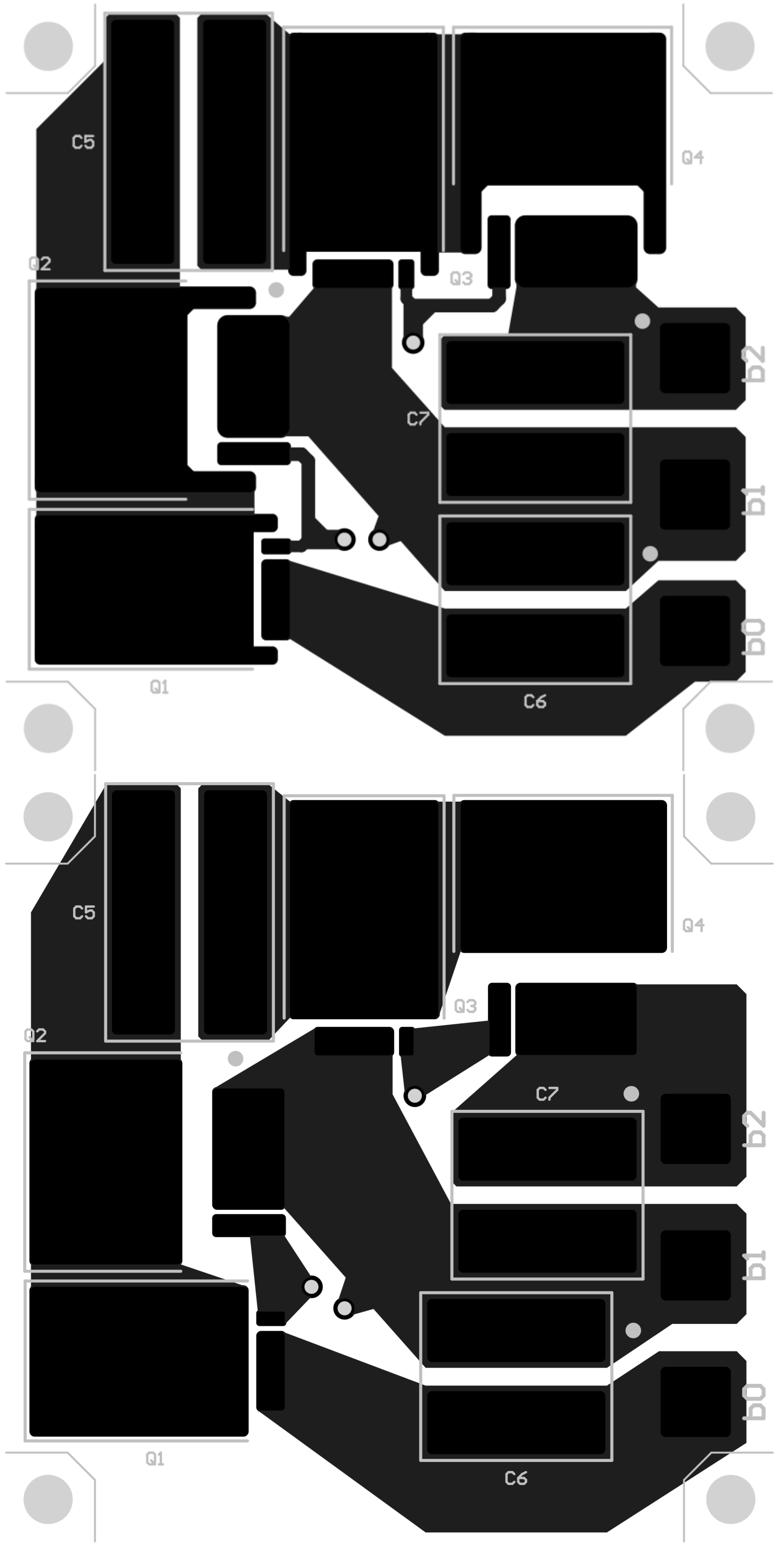

Figure C-4: 2D view of power side of 2S PCB: revisions 1 (top) and 2 (bottom).

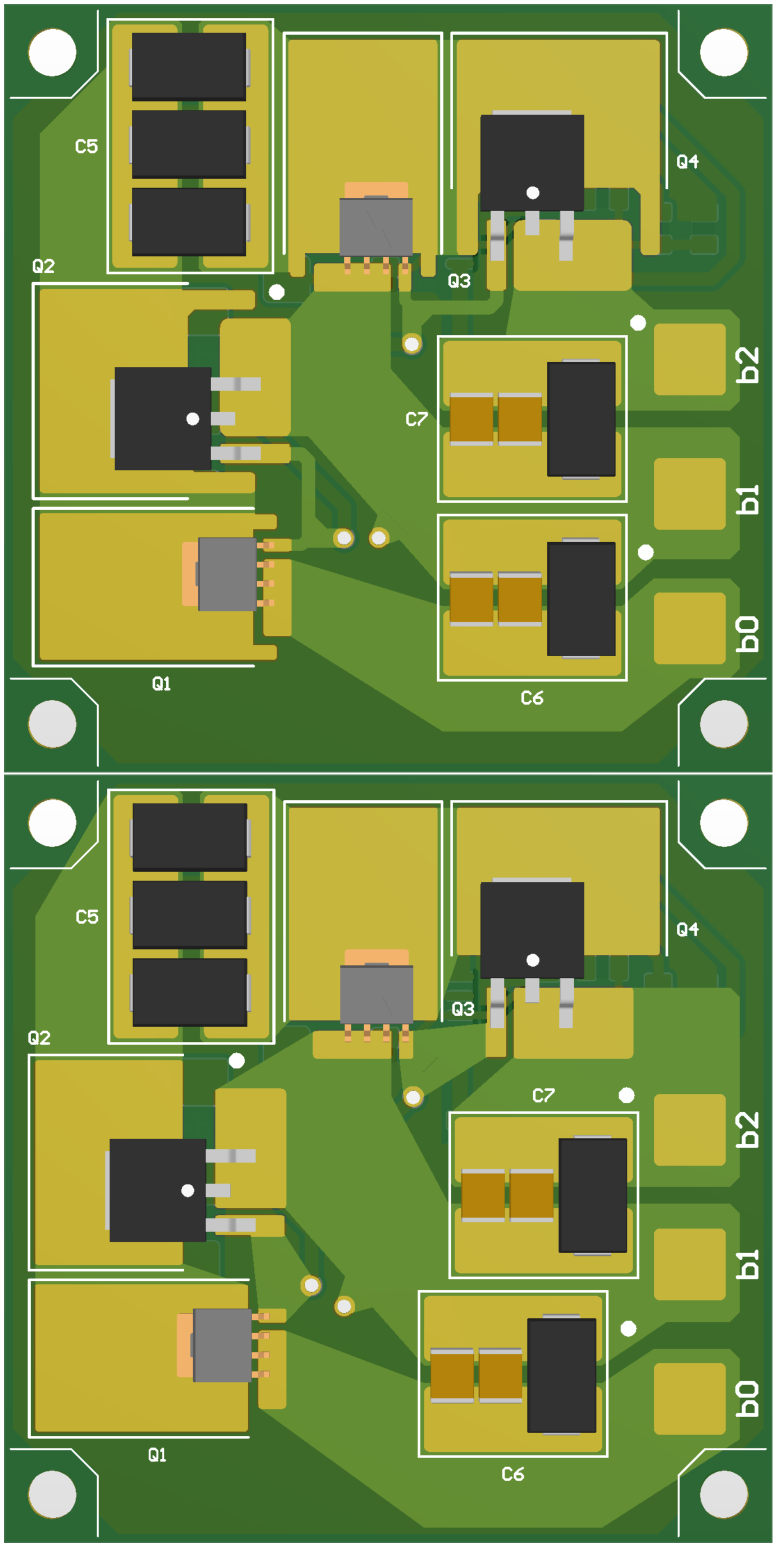

Figure C-5: 3D view of power side of 2S PCB: revisions 1 (top) and 2 (bottom).

C.2 4S PCB

Figure C-6 shows the 4S PCB schematic and Table C.2 lists the 4S PCB components. Figures C-7 and C-8 show two different views of the control side of the 4S board; both revisions 1 and 2 are shown. Figures C-9 and C-10 show two different views of the power side; likewise, both revisions 1 and 2 are shown.

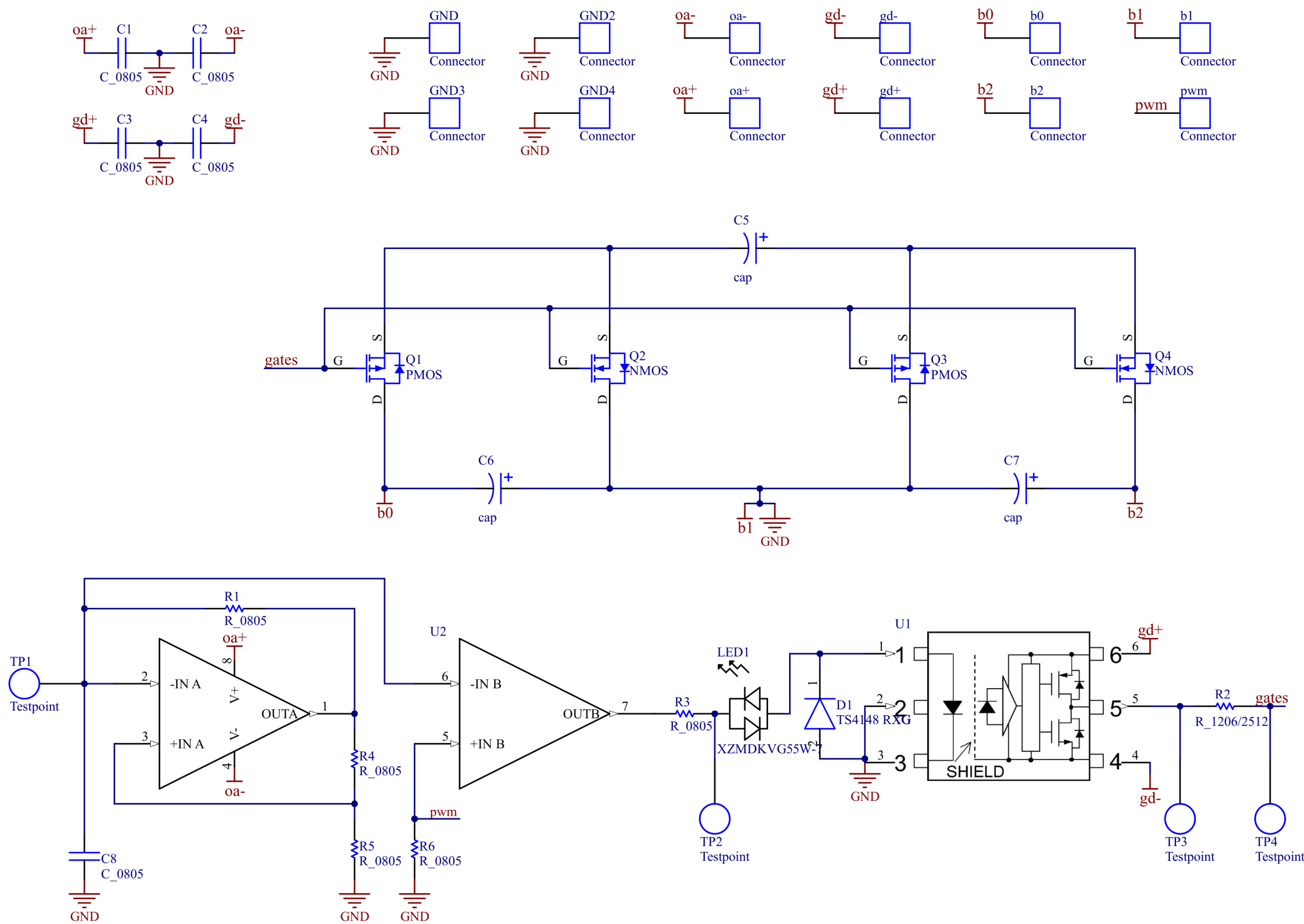

Figure C-6: Circuit diagram of 4S PCB. (Optocoupler gate driver image courtesy of TLP5774 datasheet.)

Table C.2: Components 4S PCB

Designator	Description
	Control side
C1	bypass positive supply op-amp
C2	bypass negative supply op-amp
C3	bypass positive supply gate driver
C4	bypass negative supply gate driver
C8	timing capacitor op-amp
D1	signal diode anti-parallel with optocoupler LED
LED1	indicator LED
R1	timing resistor op-amp
R2	gate resistor
R3	output resistor op-amp
R4	feedback divider resistor op-amp
R5	feedback divider resistor op-amp
R6	pull-down resistor pwm
U1	gate driver
U2	dual op-amp as multivibrator and comparator
	Power side
C5	flying capacitor
C6	bypass capacitor 1
C7	bypass capacitor 2
Q1	low threshold PMOS (and heat sink)
Q2	high threshold NMOS
Q3	high threshold PMOS (and heat sink)
Q4	low threshold NMOS

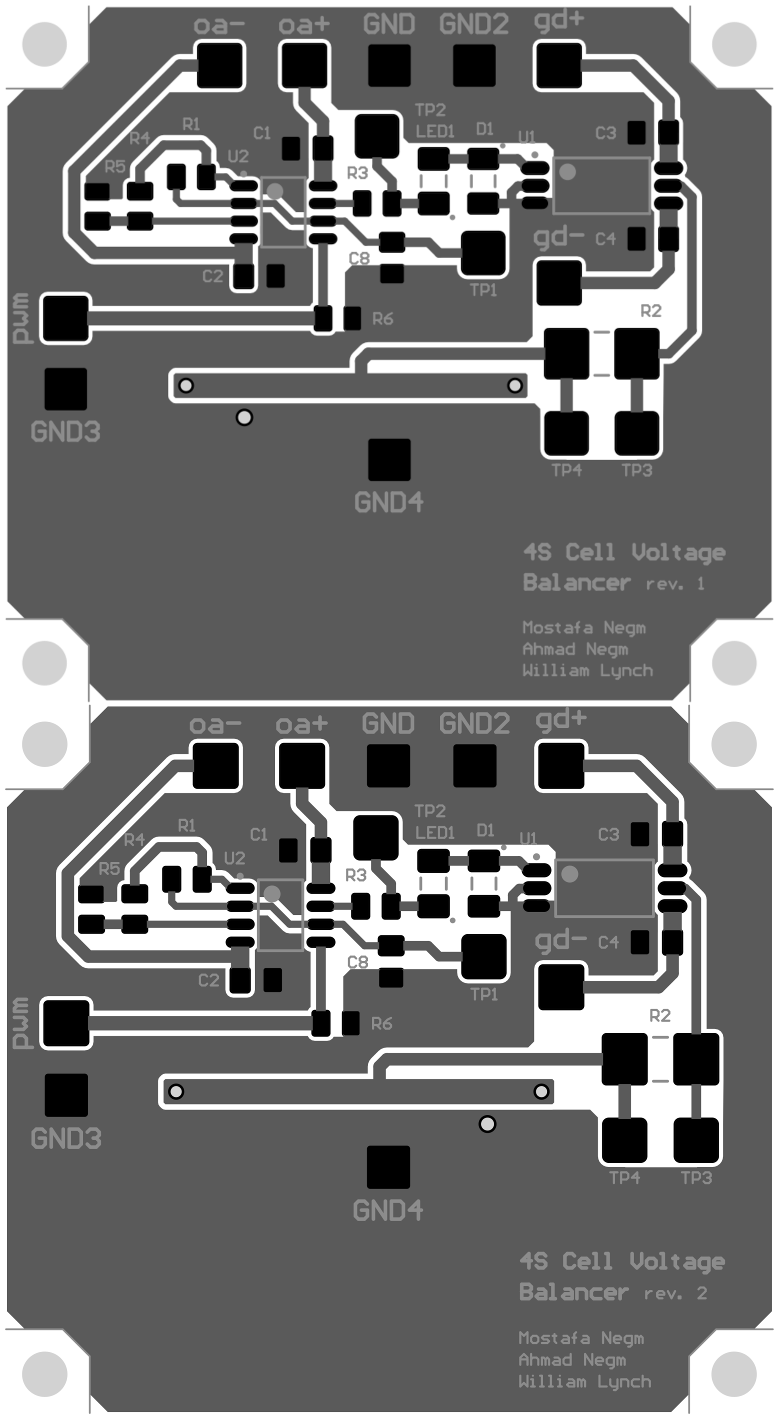

Figure C-7: 2D view of control side of 4S PCB: revisions 1 (top) and 2 (bottom).

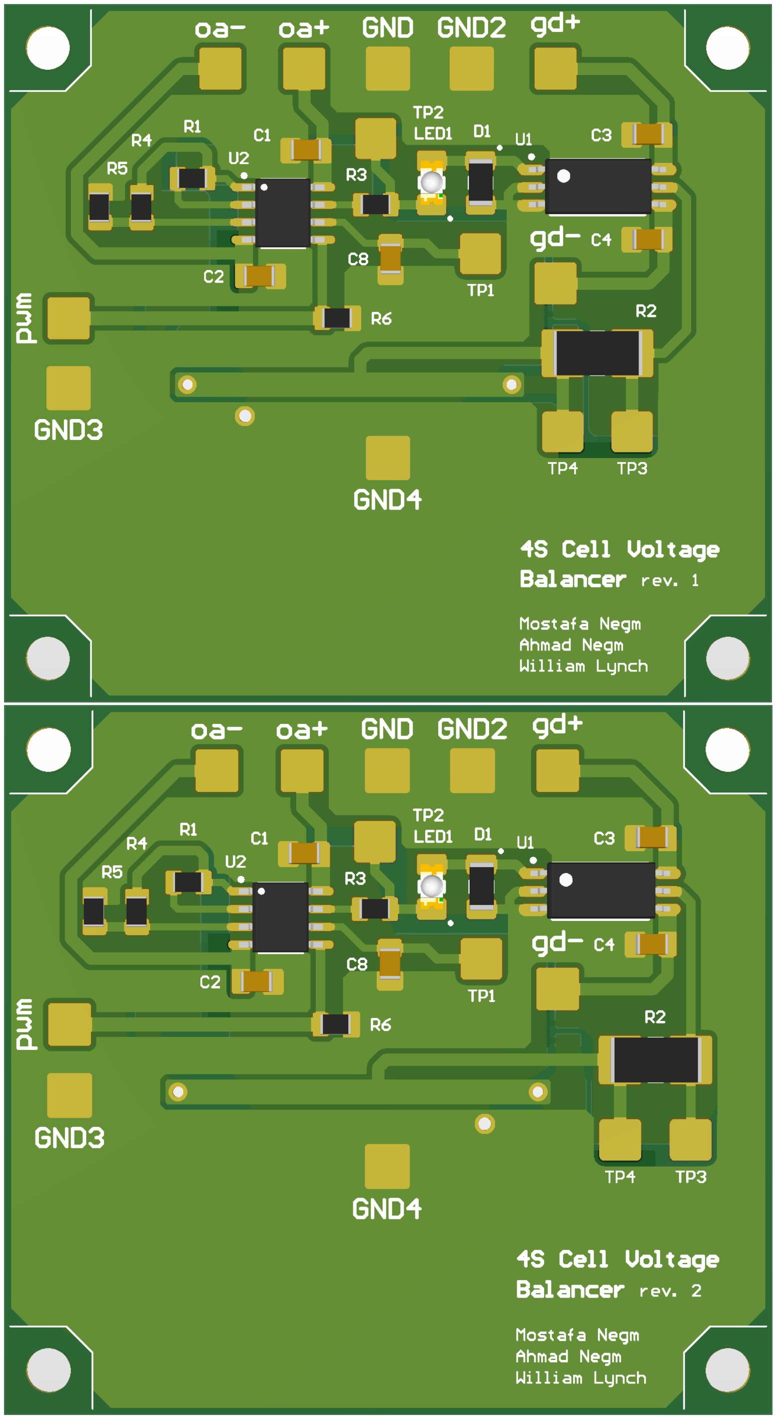

Figure C-8: 3D view of control side of 4S PCB: revisions 1 (top) and 2 (bottom).

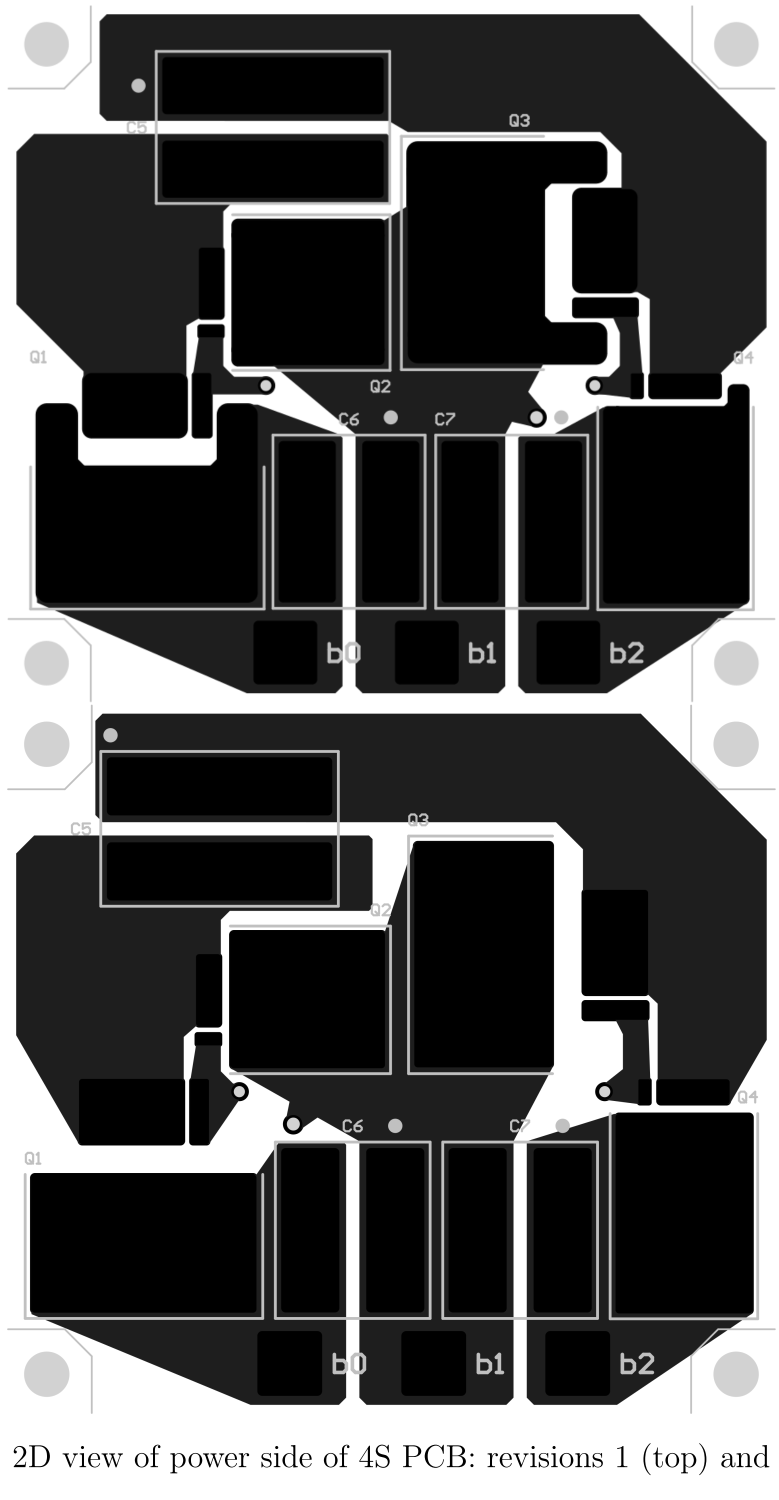

Figure C-9: 2D view of power side of 4S PCB: revisions 1 (top) and 2 (bottom).

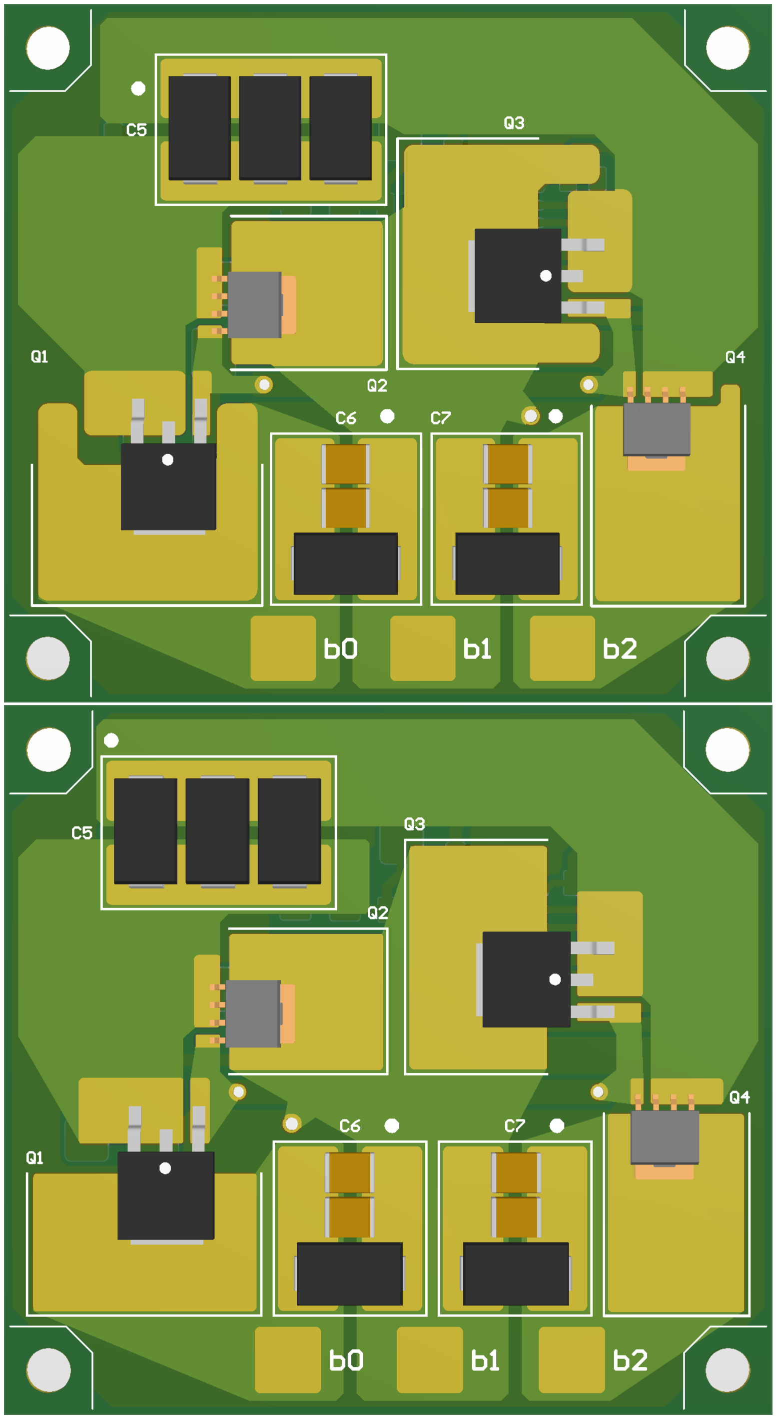

Figure C-10: 3D view of power side of 4S PCB: revisions 1 (top) and 2 (bottom).

Appendix D

Balancer and Tester Results

This appendix shows some of the experimental results from the 2S and 4S balancer (perfboard) prototypes and PCBs. The experiments with the 2S balancers used 3.0 V and 3.4 V as the 'cell' voltages, and the experiments with the 4S balancers used 6.0 V and 6.8 V. Moreover, the experiments on the prototypes did *not* use the remote sense feature of the power supplies. In contrast, the experiments on the PCBs did use the remote sense feature. The only exception to this is when we forgot to turn on remote sense for one of the experiments and is noted (the yellow curve in Figure D-9).

This appendix also shows some of the experimental results from the cell tester prototype.

D.1 2S Balancer Prototype

The following MOSFETs were used on the 2S balancer prototype: Infineon IPB180N04S401, Infineon IPB180P04P4L02, Infineon IPB011N04LG, and Infineon IPB180P04P403 as M1, M2, M3, and M4, respectively.

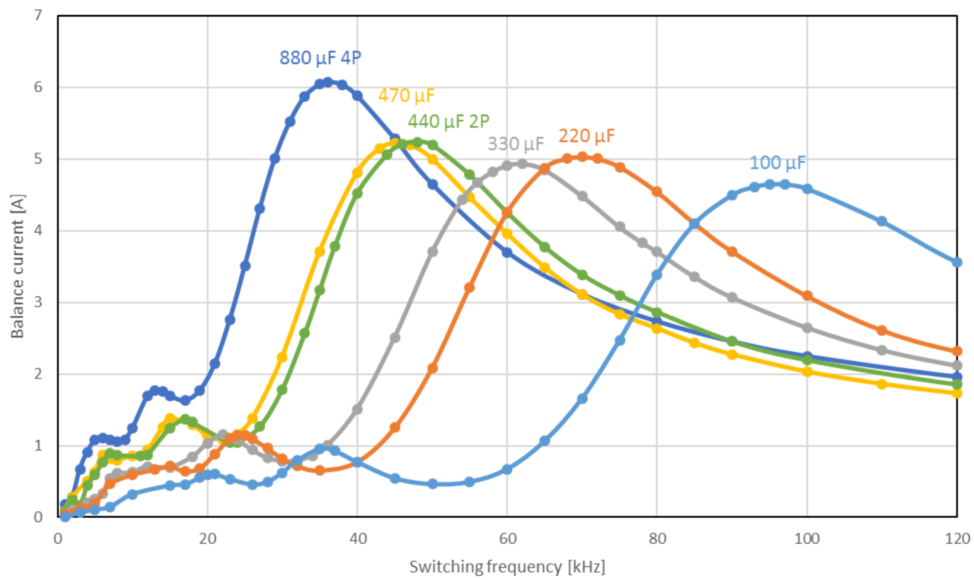

Figure D-1: Balance current vs switching frequency for ceramic capacitors on the 2S balancer prototype.

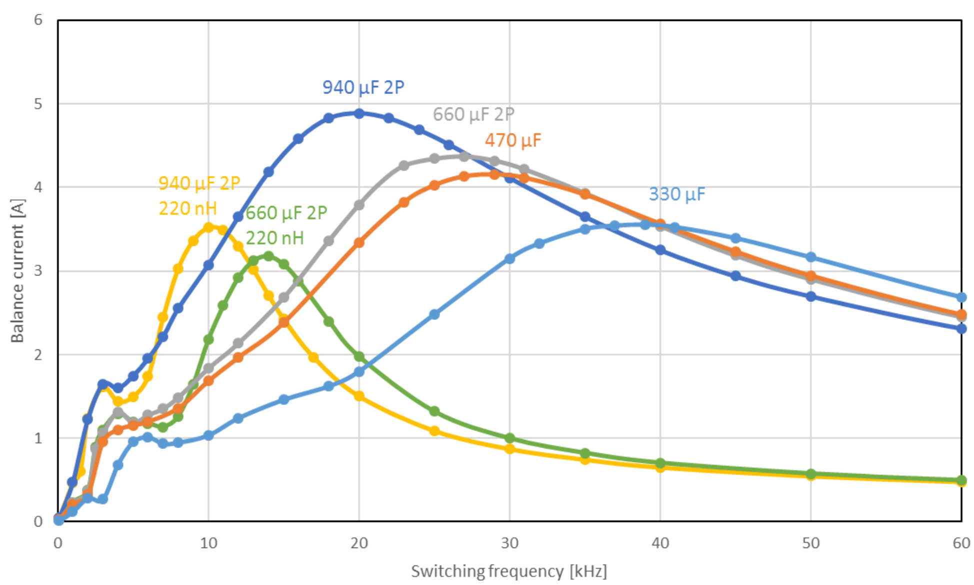

Figure D-2: Balance current vs switching frequency for aluminum polymer capacitors on the 2S balancer prototype.

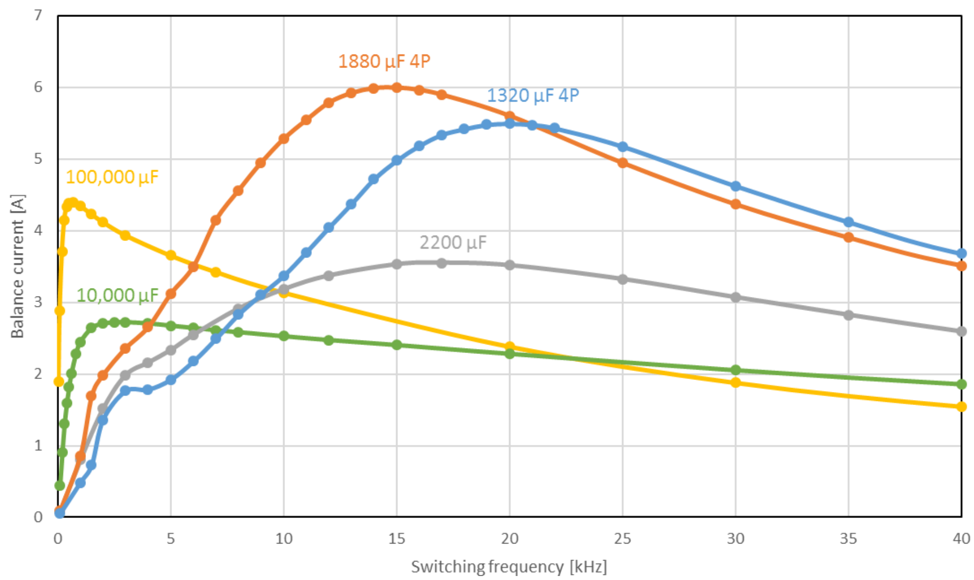

Figure D-3: Balance current vs switching frequency for aluminum polymer and electrolytic capacitors on the 2S balancer prototype.

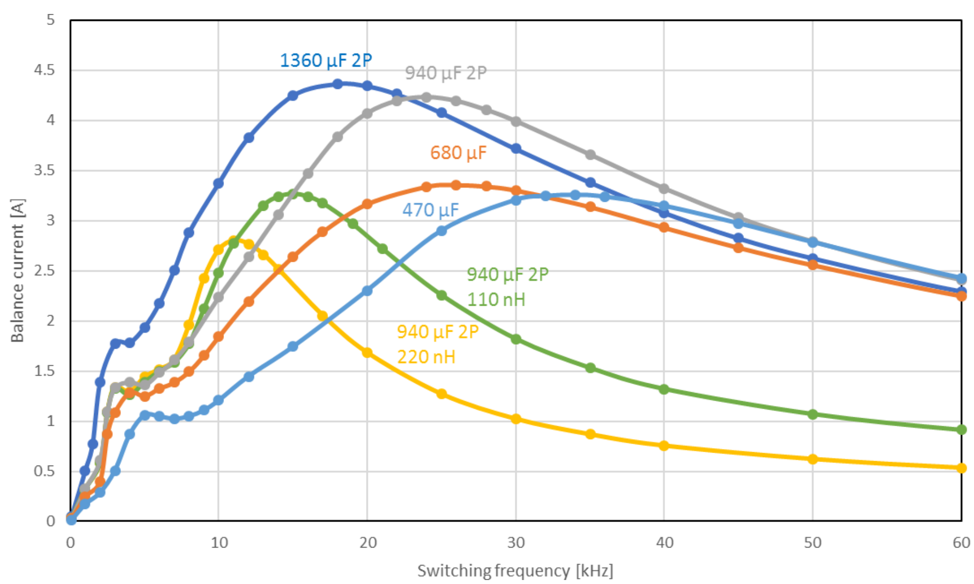

Figure D-4: Balance current vs switching frequency for tantalum polymer capacitors on the 2S balancer prototype.

D.2 4S Balancer Prototype

Various MOSFET combinations were tried on the 4S balancer prototype. For the curves shown, the following MOSFETs were used: Infineon IPD042P03L3G, ON Semicondutor FDD9409-F085, Infineon IPD90P03P404, and Vishay SQJQ100EL as M1, M2, M3, and M4, respectively.

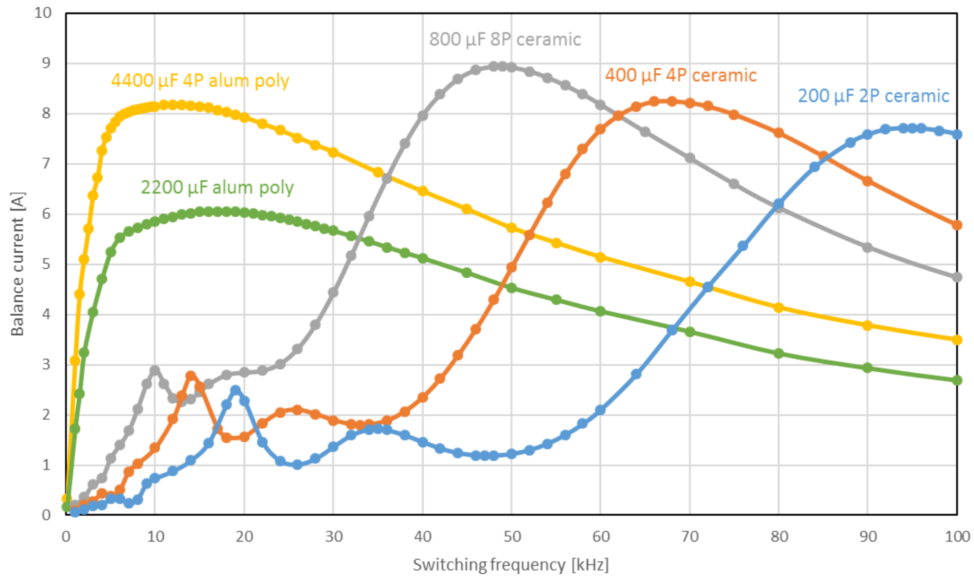

Figure D-5: Balance current vs switching frequency for ceramic and aluminum polymer capacitors on the 4S balancer prototype.

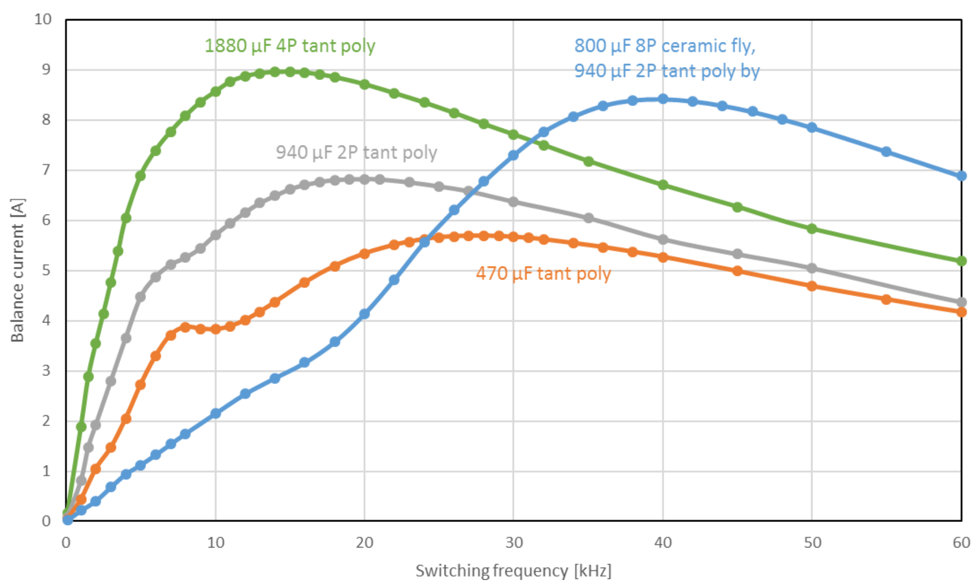

Figure D-6: Balance current vs switching frequency for tantalum polymer capacitors on the 4S balancer prototype.

D.3 2S Balancer PCB

The following MOSFETs were used on the 2S balancer PCB: ON Semiconductor NVMYS-1D3N04C, Vishay SQD40031EL, Nexperia PSMNR70-30YLHX, and Infineon IPD90P03-P404 as M1, M2, M3, and M4, respectively. However, for the gray curve in Figure D-7, P-channels in the larger TO-263 package were used in an attempt to push performance, Infineon IPB180P04P4L02 as M2 and Infineon IPB180P04P403 as M4.

Figure D-8 shows the balance current vs cell voltage differential for ceramic and tantalum polymer capacitors on the 2S balancer PCB. Note the linearity of balance current with cell voltage differential. This is an intrinsic property of the charge pump based balancer and extends beyond the 2S. At high voltage differential, the current may slightly deviate from linearity to a lower value (in magnitude) due to the increased power dissipation and thus increased circuit resistance (e.g., traces, MOSFETs, etc.).

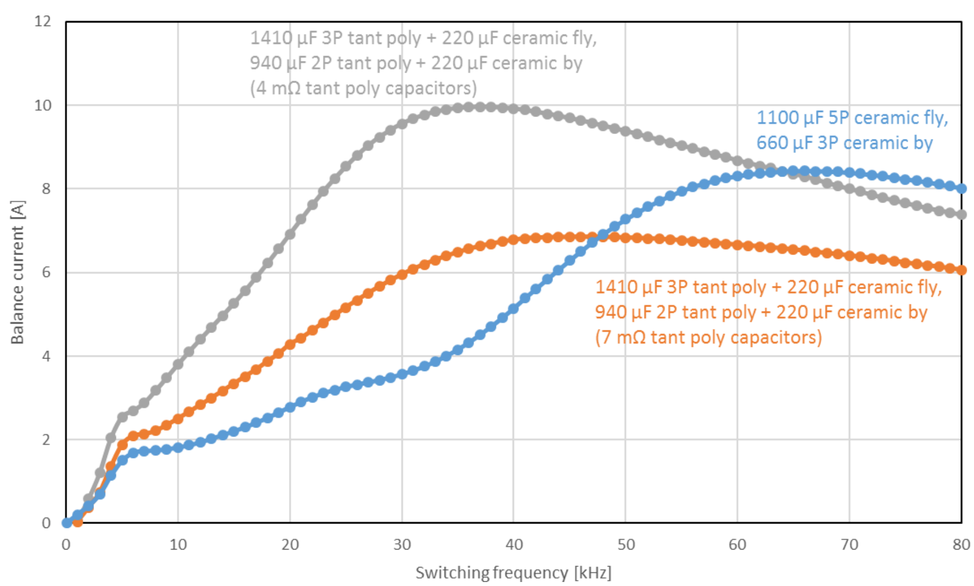

Figure D-7: Balance current vs switching frequency for ceramic and tantalum polymer capacitors on the 2S balancer PCB.

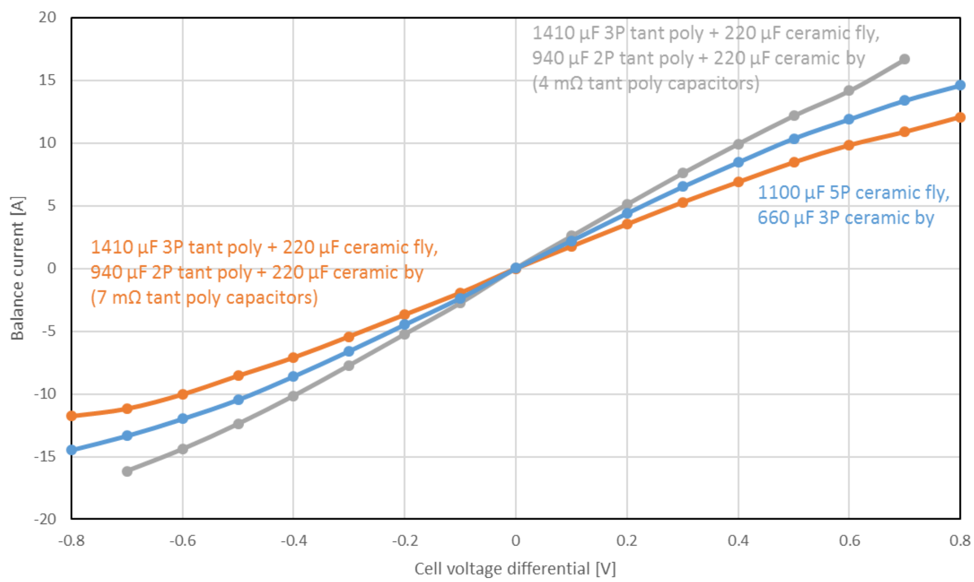

Figure D-8: Balance current vs cell voltage differential for ceramic and tantalum polymer capacitors on the 2S balancer PCB.

D.4 4S Balancer PCB

The following MOSFETs were used on the 4S balancer PCB: Vishay SQD40031EL, ON Semiconductor NVMYS1D3N04C, Infineon IPB180P04P403, and Nexperia PSMNR70-30YLHX as M1, M2, M3, and M4, respectively.

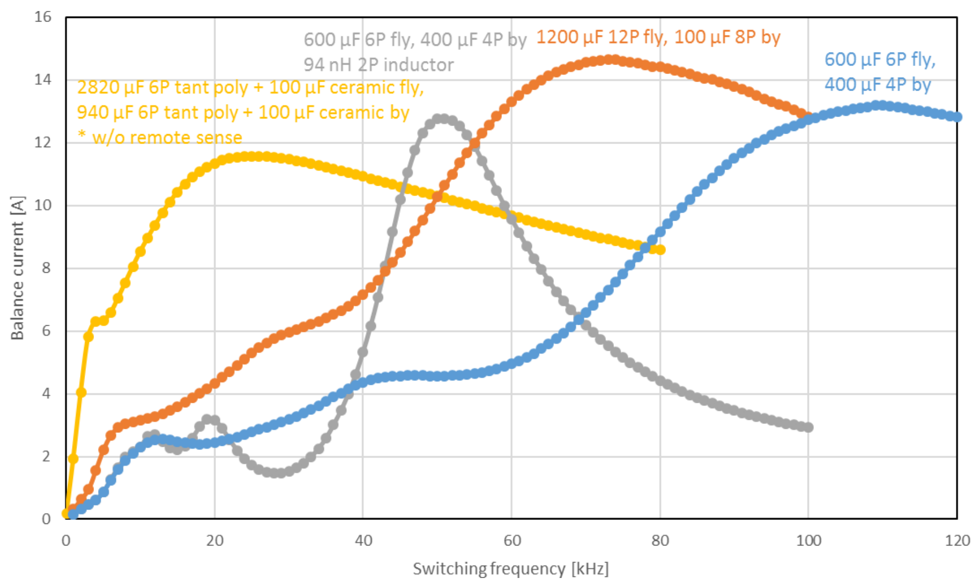

Figure D-9: Balance current vs switching frequency for ceramic and tantalum polymer capacitors on the 4S balancer PCB.

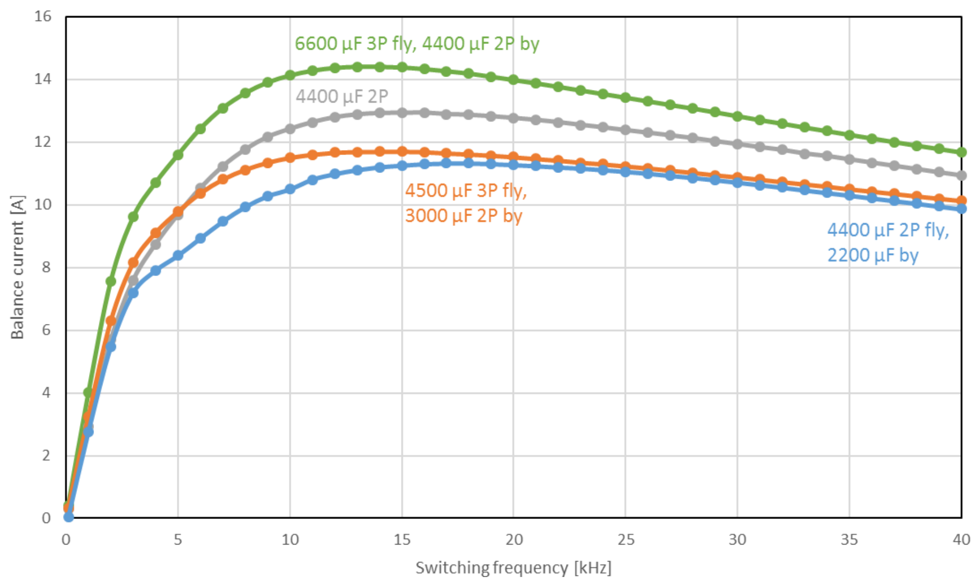

Figure D-10: Balance current vs switching frequency for aluminum polymer capacitors on the 4S balancer PCB.

D.5 Cell Tester Prototype

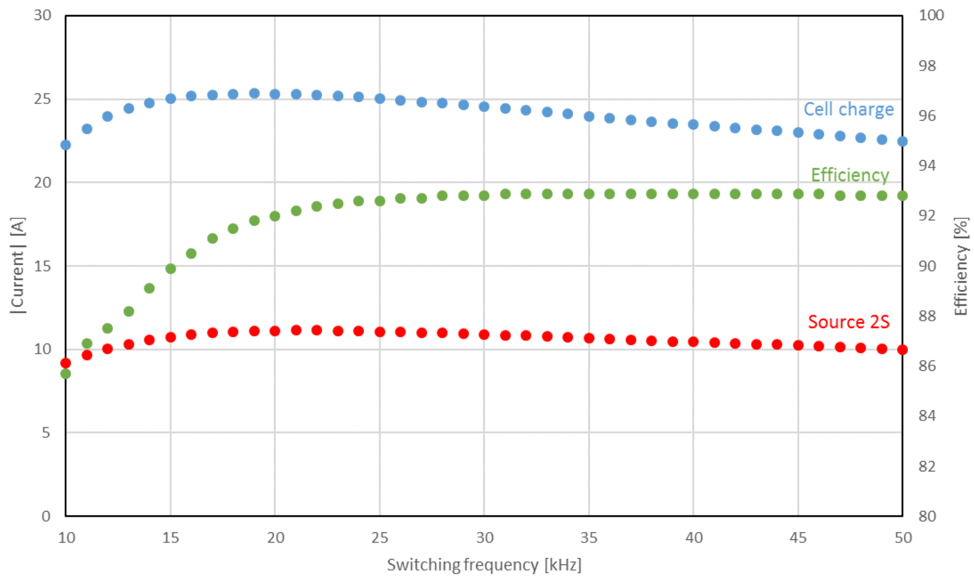

Figure D-11: Current vs switching frequency at 45 % duty cycle on the cell tester prototype.

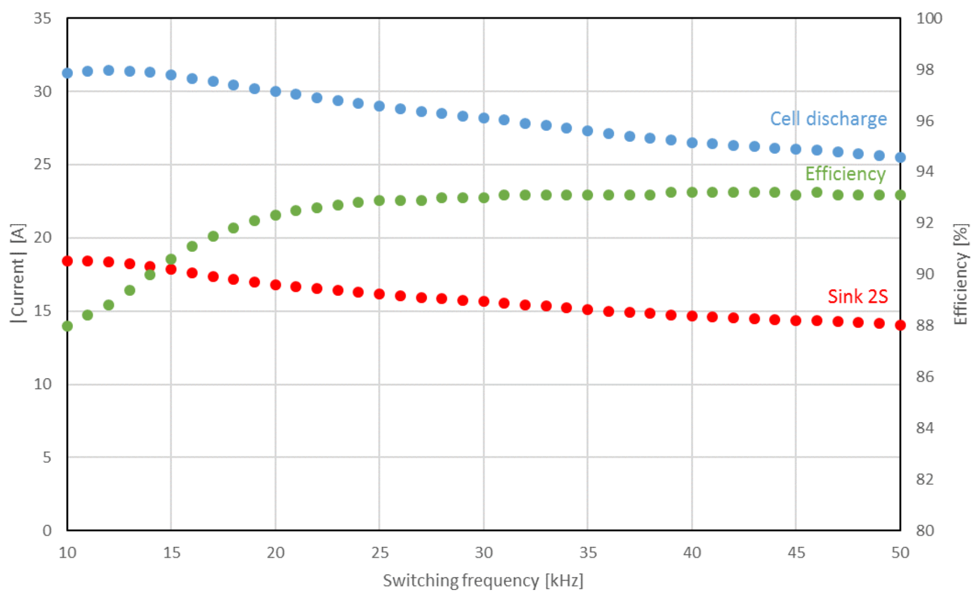

Figure D-12: Current vs switching frequency at 55 % duty cycle on the cell tester prototype.

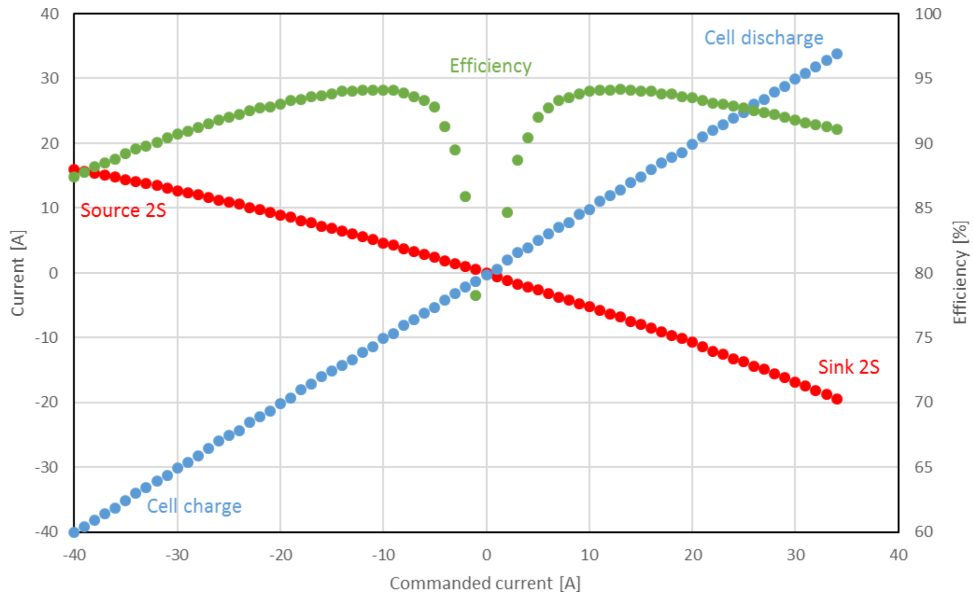

Figure D-13: Current vs commanded current at 30 kHz switching frequency on the cell tester prototype.

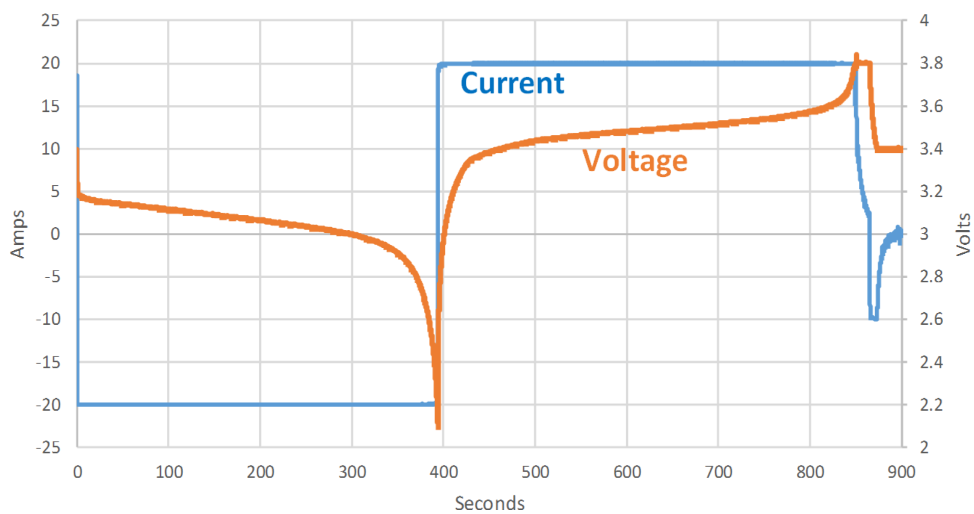

Figure D-14: Cycle test with a cell on the cell tester prototype.

Appendix E

Cell Tester Program

The following program was written for the cell tester. More testing of the program is needed.

```
"""
Program for battery tester circuit.

There are two variations of the program:
    1) manual program (runs by default)
    2) cycling program (runs with '-cycle' as command line argument)

ALWAYS MAKE SURE THERE IS A GATE SIGNAL BEFORE CONNECTING THE CELLS!!!
PLEASE REMEMBER TO DISCONNECT CELLS ON PROGRAM QUIT!!!
The biggest potential failure modes of this program:
    1) program fails to output pulse
    2) program outputs a pulse of fixed duty cycle and cell voltage
   drifts out of range

The program uses two threads, the second one for the controls, and the
   main one for the gui.

Python 3.8 or higher required. Tested on Python 3.8.
External packages required:
    matplotlib (https://matplotlib.org/)
    mcculw (https://github.com/mccdaq/mcculw/)
```

```

Tested on USB-1608GX-2AO.

Authors:
    Mostafa Negm
    William Lynch
"""

import abc
from collections import deque
import csv
import datetime
import enum
import logging
import operator
import queue
import sys
import threading
import time
import tkinter as tk
import tkinter.font as tkFont
import tkinter.messagebox

from matplotlib.animation import FuncAnimation
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import matplotlib.pyplot as plt

import mcculw
import mcculw.device_info

##########
# CONSTANTS
##########

DEBUG = True  # check_rep and logs controls contributions of
    proportional, derivative, and integral
```

```

# channels
CHANNEL_V_PWM = 1
CHANNEL_CELL_UNDER_TEST = 3
CHANNEL_BUFFER_BOTTOM = 1
CHANNEL_BUFFER_TOP = 0
CHANNEL_CURRENT_SENSOR = 4

# analog output
NOMINAL_V_PWM = 0
MIN_V_PWM = -10
MAX_V_PWM = 10

# cell under test
NOMINAL_CURRENT = 0
MIN_CURRENT = -10
MAX_CURRENT = 10
ABS_MIN_CURRENT = -15
ABS_MAX_CURRENT = 15

NOMINAL_VOLTAGE = 3.3
MIN_VOLTAGE = 1.5
MAX_VOLTAGE = 3.8
ABS_MIN_VOLTAGE = 1.3
ABS_MAX_VOLTAGE = 4

# buffer cells
ABS_MIN_VOLTAGE_BUFFER = 3.0
ABS_MAX_VOLTAGE_BUFFER = 3.6

# current sensor
AMPS_PER_VOLT_CURRENT_SENSOR = -5
OFFSET_CURRENT_SENSOR = -0.13

# controls
DELTA_T = 1/100
```

```

# for derivative control
# M_BACK = 1: susceptible to noise
# M_BACK > 1: less susceptible to noise, but go too far back in time and information is outdated and thus
#             counterproductive
M_BACK_CC = 2
M_BACK_VC = 1

# current control
K_P_CC = ((MAX_V_PWM - MIN_V_PWM) / (MAX_CURRENT - MIN_CURRENT)) * (1/8)
K_D_CC = 0.01 * K_P_CC
K_I_CC = 20 * K_P_CC

# voltage control
K_P_VC = ((MAX_V_PWM - MIN_V_PWM) / (MAX_VOLTAGE - MIN_VOLTAGE)) * (1/8)
K_D_VC = 0.01 * K_P_VC
K_I_VC = 60 * K_P_VC

#
DISPLAY_REFRESH_PERIOD_MS = 100
MULTIPLE_WRITING = 1
MULTIPLE_PLOTTING = 30

# A basic sanity check of (some of) the program constants. Far from exhaustive.
# DAQ channels are nonnegative integers
assert isinstance(CHANNEL_V_PWM, int) and CHANNEL_V_PWM >= 0
assert isinstance(CHANNEL_CURRENT_SENSOR, int) and CHANNEL_CURRENT_SENSOR >= 0
assert isinstance(CHANNEL_CELL_UNDER_TEST, int) and CHANNEL_CELL_UNDER_TEST >= 0
assert isinstance(CHANNEL_BUFFER_BOTTOM, int) and CHANNEL_BUFFER_BOTTOM >= 0
```

```
assert isinstance(CHANNEL_BUFFER_TOP, int) and CHANNEL_BUFFER_TOP >= 0

# different analog inputs can't be connected to the same channels
assert len({CHANNEL_CELL_UNDER_TEST, CHANNEL_BUFFER_BOTTOM,
    CHANNEL_BUFFER_TOP, CHANNEL_CURRENT_SENSOR}) == 4

# -10 <= MIN_V_PWM <= NOMINAL_V_PWM <= MAX_V_PWM <= 10
assert -10 <= MIN_V_PWM
assert MIN_V_PWM <= NOMINAL_V_PWM
assert NOMINAL_V_PWM <= MAX_V_PWM
assert MAX_V_PWM <= 10

# ABS_MIN_CURRENT <= MIN_CURRENT <= NOMINAL_CURRENT <= MAX_CURRENT <=
    ABS_MAX_CURRENT
assert ABS_MIN_CURRENT <= MIN_CURRENT
assert MIN_CURRENT <= NOMINAL_CURRENT
assert NOMINAL_CURRENT <= MAX_CURRENT
assert MAX_CURRENT <= ABS_MAX_CURRENT

# 1.3 <= ABS_MIN_VOLTAGE < MIN_VOLTAGE < ABS_MIN_VOLTAGE_BUFFER <
    NOMINAL_VOLTAGE
# < ABS_MAX_VOLTAGE_BUFFER < MAX_VOLTAGE < ABS_MAX_VOLTAGE <= 4
assert 1.3 <= ABS_MIN_VOLTAGE
assert ABS_MIN_VOLTAGE < MIN_VOLTAGE
assert MIN_VOLTAGE < ABS_MIN_VOLTAGE_BUFFER
assert ABS_MIN_VOLTAGE_BUFFER < NOMINAL_VOLTAGE
assert NOMINAL_VOLTAGE < ABS_MAX_VOLTAGE_BUFFER
assert ABS_MAX_VOLTAGE_BUFFER < MAX_VOLTAGE
assert MAX_VOLTAGE < ABS_MAX_VOLTAGE
assert ABS_MAX_VOLTAGE <= 4

# low positive proportional and integral gains, nonnegative derivative
    gains
# derivative and integral gains (much) lower than proportional gain
# cc
assert 0 < K_P_CC < 1
```

```python
assert 0 < K_I_CC < 70*K_P_CC
assert 0 <= K_D_CC < 0.1*K_P_CC
# vc
assert 0 < K_P_VC < 100
assert 0 < K_I_VC < 70*K_P_VC
assert 0 <= K_D_VC < 0.1*K_P_VC

assert isinstance(M_BACK_CC, int) and 1 <= M_BACK_CC <= 3
assert isinstance(M_BACK_VC, int) and 1 <= M_BACK_VC <= 3

assert 1/200 <= DELTA_T <= 1
assert DISPLAY_REFRESH_PERIOD_MS >= 100

# integer multiples
assert isinstance(MULTIPLE_WRITING, int) and MULTIPLE_WRITING >= 1
assert isinstance(MULTIPLE_PLOTTING, int) and MULTIPLE_PLOTTING >= 20


##########
# other
##########

def between(val, min_val, max_val):
    """
    Return True if min_val <= val <= max_val, else False.
    """
    return min_val <= val <= max_val


def clip(val, min_val, max_val):
    """
    Return min_val, if val < min_val
           max_val, if val > max_val
           val,     otherwise
    """
    if min_val > max_val:
```

```
        min_val, max_val = max_val, min_val
    return min(max_val, max(val, min_val))


def display_float(x, decimal_places) -> str:
    """
    Return x as a string with the specified number of decimal_places
   (>= 0) if x is numeric else x.
    """
    if is_numeric(x):
        return '{:.{}f}'.format(x, decimal_places)
    return x


def is_numeric(x) -> bool:
    """
    Return True if x is a number or numeric string, else False.
    """
    try:
        float(x)
        return True
    except (TypeError, ValueError):
        return False


##########
# Tester class
##########

class Tester(tk.Frame):
    """
    This class is only a base class and should be subclassed.

    See Also
    --------
    ManualTester, CycleTester
```

```
    """

    def __init__(self):
        super(Tester, self).__init__(tk.Tk())

        # Initialize tkinter properties
        self.master.protocol("WM_DELETE_WINDOW", self.quit)
        self.master.wm_title(type(self).__name__)
        self.master.minsize(width=800, height=600)
        self.master.grid_columnconfigure(0, weight=1)
        self.master.grid_rowconfigure(0, weight=1)

        self.grid(sticky=tk.NSEW)

        # default fonts are small, so make them slightly larger
        tk_default_font = tkFont.nametofont("TkDefaultFont")
        tk_default_font.configure(size=tk_default_font.actual('size')
+2)
        tk_text_font = tkFont.nametofont("TkTextFont")
        tk_text_font.configure(size=tk_text_font.actual('size')+2)

        # By default, the example detects all available devices and
selects the
        # first device listed.
        # If use_device_detection is set to False, the board_num
property needs
        # to match the desired board number configured with Instacal.
        use_device_detection = True
        self.board_num = 0
        try:
            if use_device_detection:
                self._configure_first_detected_device()
            self.device_info = mcculw.device_info.DaqDeviceInfo(self.
board_num)
        except Exception:
            raise
```

```
        # analog input support
        if not self.device_info.supports_analog_input:
            raise Exception('Error: The DAQ device does not support
    analog input')
        self.ai_info = self.device_info.get_ai_info()
        self.ai_range = self.ai_info.supported_ranges[0]

        # analog out support
        if not self.device_info.supports_analog_output:
            raise Exception('Error: The DAQ device does not support
    analog output')
        self.ao_info = self.device_info.get_ao_info()

        #
        self.program_quit = False
        self.controls_running = False

        #
        self.control_mode = Tester.ControlMode.current
        self.control_mode_var = tk.StringVar(value=self.control_mode.
    value)

        self.desired_current = NOMINAL_CURRENT
        self.desired_voltage = NOMINAL_VOLTAGE
        self.desired_v_pwm = NOMINAL_V_PWM

        self.measured_current = self.measure_current()
        self.measured_voltage = self.measure_voltage(
    CHANNEL_CELL_UNDER_TEST)
        self.voltage_buffer_bottom = self.measure_voltage(
    CHANNEL_BUFFER_BOTTOM)
        self.voltage_buffer_top = self.measure_voltage(
    CHANNEL_BUFFER_TOP)
        self.actual_v_pwm = self.a_out(CHANNEL_V_PWM, NOMINAL_V_PWM)
        self.abs_current_limit = abs(ABS_MAX_CURRENT)
```

```python

        # used to pass messages from controls thread to main thread for plotting purposes
        self.message_passing = queue.Queue()

        # store values related to plotting
        self.plotting = {}

        # for controls
        self.queue = deque()
        self.lock = threading.Lock()

        self.create_gui()
        self.update_display()
        self.ani = FuncAnimation(self.fig, self.update_plot, interval=1000)
        self._periodically_ping()
        self.thread_controls = self.start_thread_controls()
        self.check_rep()

    def _configure_first_detected_device(self):
        """
        See also:
            https://github.com/mccdaq/mcculw/blob/master/examples/ui/ui_examples_util.py
        """
        mcculw.ul.ignore_instacal()
        devices = mcculw.ul.get_daq_device_inventory(mcculw.enums.InterfaceType.ANY)
        if not devices:
            raise mcculw.ul.ULError(mcculw.enums.ErrorCode.BADBOARD)

        # Add the first DAQ device to the UL with the specified board number
        mcculw.ul.create_daq_device(self.board_num, devices[0])

```

```
    def check_rep(self):
        if DEBUG:
            # MIN_CURRENT <= self.desired_current <= MAX_CURRENT
            assert self.desired_current >= MIN_CURRENT
            assert self.desired_current <= MAX_CURRENT

            # MIN_VOLTAGE <= self.desired_voltage <= MAX_VOLTAGE
            assert self.desired_voltage >= MIN_VOLTAGE
            assert self.desired_voltage <= MAX_VOLTAGE

            # MIN_V_PWM <= self.desired_v_pwm <= MAX_V_PWM
            assert self.desired_v_pwm >= MIN_V_PWM
            assert self.desired_v_pwm <= MAX_V_PWM

    ##########
    # Tester.ControlMode enum
    ##########

    class ControlMode(enum.Enum):
        current = "cc"
        voltage = "vc"

    ##########
    # mcculw: https://github.com/mccdaq/mcculw/
    ##########

    def a_in(self, channel):
        """
        Return the analog input of a user-specified channel.

        See also:
            https://github.com/mccdaq/mcculw/blob/master/examples/
console/a_in.py
        """
        try:
            # Get a value from the device
```

```
            if self.ai_info.resolution <= 16:
                # Use the a_in method for devices with a resolution <= 16
                value = mcculw.ul.a_in(self.board_num, channel, self.ai_range)
                # Convert the raw value to engineering units
                eng_units_value = mcculw.ul.to_eng_units(self.board_num, self.ai_range, value)
            else:
                # Use the a_in_32 method for devices with a resolution > 16
                # (optional parameter omitted)
                value = mcculw.ul.a_in_32(self.board_num, channel, self.ai_range)
                # Convert the raw value to engineering units
                eng_units_value = mcculw.ul.to_eng_units_32(self.board_num, self.ai_range, value)

            # Display the raw value
            # print('Raw Value:', value)
            # Display the engineering value
            # print('Engineering Value: {:.3f}'.format(eng_units_value))
            return eng_units_value
        except Exception as e:
            logger.error(e, exc_info=True)
            raise

    def a_out(self, channel, voltage):
        """
        Write the voltage to a user-specified channel. Return the outputted voltage.

        See also:
            https://github.com/mccdaq/mcculw/blob/master/examples/console/v_out.py
```

```
        """
        try:
            ao_range = self.ao_info.supported_ranges[0]
            voltage = clip(voltage, ao_range.range_min, ao_range.range_max)

            # print('Outputting', voltage, 'Volts to channel', channel)
            # Send the value to the device (optional parameter omitted)
            mcculw.ul.v_out(self.board_num, channel, ao_range, voltage)
            return voltage
        except Exception as e:
            logger.error(e, exc_info=True)
            raise

    ##########
    # getters and setters
    ##########

    def get_desired_current(self) -> float:
        """
        Getter for self.desired_current.
        """
        self.check_rep()
        return self.desired_current

    def set_desired_current(self, desired_current) -> None:
        """
        Setter for self.desired_current.

        Sets self.desired_current if desired_current is numeric. Clips the value to be within the limits if necessary.
        """
        if is_numeric(desired_current):
            self.desired_current = clip(float(desired_current), MIN_CURRENT, MAX_CURRENT)
        self.check_rep()
```

```
    def get_desired_voltage(self) -> float:
        """
        Getter for self.desired_voltage.
        """
        self.check_rep()
        return self.desired_voltage

    def set_desired_voltage(self, desired_voltage) -> None:
        """
        Setter for self.desired_voltage.

        Sets self.desired_voltage if desired_voltage is numeric. Clips the value to be within the limits if necessary.
        """
        if is_numeric(desired_voltage):
            self.desired_voltage = clip(float(desired_voltage), MIN_VOLTAGE, MAX_VOLTAGE)
        self.check_rep()

    def get_desired_v_pwm(self) -> float:
        """
        Getter for self.desired_v_pwm.
        """
        self.check_rep()
        return self.desired_v_pwm

    def set_desired_v_pwm(self, desired_v_pwm) -> None:
        """
        Setter for self.desired_v_pwm.

        Sets self.desired_v_pwm if desired_v_pwm is numeric. Clips the value to be within the limits if necessary.
        """
        if is_numeric(desired_v_pwm):
```

```
            self.desired_v_pwm = clip(float(desired_v_pwm), MIN_V_PWM,
    MAX_V_PWM)
        self.check_rep()

    def get_measured_current(self) -> float:
        """
        Getter for self.measured_current.
        """
        self.check_rep()
        return self.measured_current

    def set_measured_current(self, measured_current) -> None:
        """
        Setter for self.measured_current.
        """
        self.measured_current = measured_current
        self.check_rep()

    def get_measured_voltage(self) -> float:
        """
        Getter for self.measured_voltage, the voltage of the cell under
    test.
        """
        self.check_rep()
        return self.measured_voltage

    def set_measured_voltage(self, measured_voltage) -> None:
        """
        Setter for self.measured_voltage, the voltage of the cell under
    test.
        """
        self.measured_voltage = measured_voltage
        self.check_rep()

    def get_actual_v_pwm(self) -> float:
        """
```

```
        Getter for self.actual_v_pwm.
        """
        self.check_rep()
        return self.actual_v_pwm

    def set_actual_v_pwm(self, actual_v_pwm) -> None:
        """
        Setter for self.actual_v_pwm.
        """
        self.actual_v_pwm = actual_v_pwm
        self.check_rep()

    def get_voltage_buffer_bottom(self) -> float:
        """
        Getter for self.voltage_buffer_bottom.
        """
        self.check_rep()
        return self.voltage_buffer_bottom

    def set_voltage_buffer_bottom(self, voltage_buffer_bottom) -> None:
        """
        Setter for self.voltage_buffer_bottom.
        """
        self.voltage_buffer_bottom = voltage_buffer_bottom
        self.check_rep()

    def get_voltage_buffer_top(self) -> float:
        """
        Getter for self.voltage_buffer_top.
        """
        self.check_rep()
        return self.voltage_buffer_top

    def set_voltage_buffer_top(self, voltage_buffer_top) -> None:
        """
        Setter for self.voltage_buffer_top.
```

```
        """
        self.voltage_buffer_top = voltage_buffer_top
        self.check_rep()

    def get_control_mode(self) -> ControlMode:
        """
        Getter for self.control_mode.
        """
        self.check_rep()
        return self.control_mode

    def set_control_mode(self, control_mode: ControlMode) -> None:
        """
        Setter for self.control_mode.
        """
        self.control_mode = control_mode
        self.check_rep()

    def get_abs_current_limit(self):
        """
        Getter for self.abs_current_limit. Only used for voltage control.
        """
        self.check_rep()
        return self.abs_current_limit

    def set_abs_current_limit(self, abs_current_limit: float):
        """"
        Setter for self.abs_current_limit. Only used for voltage control.
        """
        self.abs_current_limit = abs_current_limit
        self.check_rep()

    ##########
    # measure/display
```

```
    ##########

    def measure_current(self):
        """
        Measure the voltage at channel CHANNEL_CURRENT_SENSOR and multiply by AMPS_PER_VOLT_CURRENT_SENSOR
        and add OFFSET_CURRENT_SENSOR to get the current.
        """
        try:
            return AMPS_PER_VOLT_CURRENT_SENSOR * self.a_in(CHANNEL_CURRENT_SENSOR) + OFFSET_CURRENT_SENSOR
        except Exception as e:
            logger.error(e, exc_info=True)
            raise

    def measure_voltage(self, channel: int):
        """
        Measure the voltage at channel channel.
        """
        try:
            return self.a_in(channel)
        except Exception as e:
            logger.error(e, exc_info=True)
            raise

    def update_display(self):
        """
        Update the readings (measured current, voltage, v_pwm, etc.) on the GUI.
        """
        self.after(DISPLAY_REFRESH_PERIOD_MS, self.update_display)
        self.master.measured_current_value["text"] = '{} A'.format(display_float(self.get_measured_current(), 3))
        self.master.measured_voltage_value["text"] = '{} V'.format(display_float(self.get_measured_voltage(), 3))
```

```
        self.master.actual_v_pwm_value["text"] = '{} V'.format(display_float(self.get_actual_v_pwm(), 3))
        self.master.voltage_buffer_bottom_value["text"] = '{} V'.format(display_float(self.get_voltage_buffer_bottom(), 3))
        self.master.voltage_buffer_top_value["text"] = '{} V'.format(display_float(self.get_voltage_buffer_top(), 3))
        self.check_rep()

    def update_plot(self, frame):
        """
        Update the plot.
        """
        def round_to(x: float, base: float) -> float:
            # Round x to "nearest" base and return the rounded number.
            return base * round(x / base)

        new_data = False
        for _ in range(self.message_passing.qsize()):
            try:
                (time, measured_current, measured_voltage) = self.message_passing.get_nowait()
                self.plotting["times"].append(time)
                self.plotting["currents"].append(measured_current)
                self.plotting["voltages"].append(measured_voltage)
                new_data = True
            except queue.Empty:
                pass

            if self.plotting["currents"][-1] < self.plotting["min_current"]:
                self.plotting["min_current"] = self.plotting["currents"][-1]
            if self.plotting["currents"][-1] > self.plotting["max_current"]:
                self.plotting["max_current"] = self.plotting["currents"][-1]
```

```
            if self.plotting["voltages"][-1] < self.plotting["min_voltage"]:
                self.plotting["min_voltage"] = self.plotting["voltages"][-1]
            if self.plotting["voltages"][-1] > self.plotting["max_voltage"]:
                self.plotting["max_voltage"] = self.plotting["voltages"][-1]

        if new_data:
            # update data
            self.line_voltage.set_data(self.plotting["times"], self.plotting["voltages"])
            self.line_current.set_data(self.plotting["times"], self.plotting["currents"])

            # update axes limits
            self.ax_voltage.set_xlim(self.plotting["times"][0], self.plotting["times"][-1])
            # (lower, upper bound) is (round down to nearest 0.2, round up to nearest 0.2)
            lower_bound = round_to(self.plotting["min_voltage"], 0.2)
            upper_bound = round_to(self.plotting["max_voltage"], 0.2)
            if lower_bound > self.plotting["min_voltage"]:
                lower_bound -= 0.2
            if upper_bound < self.plotting["max_voltage"] or upper_bound == lower_bound:
                upper_bound += 0.2
            self.ax_voltage.set_ylim(lower_bound, upper_bound)
            # (lower, upper bound) is (floor of min towards -infinity, ceiling of max towards infinity)
            self.ax_current.set_ylim(int(self.plotting["min_current"] // 1), int(-(-self.plotting["max_current"] // 1)))
        return

    ##########
```

```
    # controls
    ##########

    def start_thread_controls(self):
        """
        Create and start a thread that handles the controls and return the created thread.
        """
        thread_controls = threading.Thread(name="", target=self.controls)
        thread_controls.start()
        return thread_controls

    def controls(self):
        """
        Handle current and voltage control.
        """
        while not self.program_quit:
            time.sleep(DELTA_T - (time.perf_counter() % DELTA_T))  # run no faster than DELTA_T

            # update measurements
            self.set_measured_current(self.measure_current())
            self.set_measured_voltage(self.measure_voltage(CHANNEL_CELL_UNDER_TEST))
            self.set_voltage_buffer_bottom(self.measure_voltage(CHANNEL_BUFFER_BOTTOM))
            self.set_voltage_buffer_top(self.measure_voltage(CHANNEL_BUFFER_TOP))

            if self.controls_running:
                self._controls()
        return

    def _controls(self):
        def datetime_now() -> str:
```

```
            # Return the current date and time in string format: "YYYYmonDD_HH;MM;SS",
            # where mon is the abbreviated month name.
            return datetime.datetime.now().strftime("%Y%b%d_%H;%M;%S")

        logger.info('in _controls')

        try:
            # create new csv
            filename = '{}.csv'.format(datetime_now())
            file = open(filename, 'w', newline='')
            writer = csv.writer(file, delimiter=',')
            try:
                row_to_write = ["time [s]", "measured current [A]", "measured voltage [V]", "v_pwm [V]"]
                if DEBUG and MULTIPLE_WRITING == 1:
                    row_to_write += ["proportional", "derivative", "integral"]
                writer.writerow(row_to_write)
            except PermissionError as e:
                logger.error(e, exc_info=True)

            # for averaging
            i_writing = 0
            sum_measured_currents_writing = 0
            sum_measured_voltages_writing = 0
            sum_actual_v_pwm_writing = 0
            i_plotting = 0
            sum_measured_currents_plotting = 0
            sum_measured_voltages_plotting = 0

            past_times = deque([0.0]*max(M_BACK_CC, M_BACK_VC), maxlen=max(M_BACK_CC, M_BACK_VC)+1)  # most recent at index 0
            # derivative on measurements (and not errors) to eliminate derivative kicks
            past_currents = deque([0.0]*M_BACK_CC, maxlen=M_BACK_CC+1)
```

```python
            past_voltages = deque([0.0]*M_BACK_VC, maxlen=M_BACK_VC+1)
            i_term = 0  # integral

            start_time = time.perf_counter()

            measured_current_0 = self.get_measured_current()
            measured_voltage_0 = self.get_measured_voltage()

            # write to csv
            try:
                row_to_write = [display_float(0, 2),
                                display_float(measured_current_0, 3),
                                display_float(measured_voltage_0, 3),
                                display_float(self.get_desired_v_pwm(),
    3)]
                if DEBUG and MULTIPLE_WRITING == 1:
                    row_to_write += [0, 0, 0]
                writer.writerow(row_to_write)
            except PermissionError as e:
                logger.error(e, exc_info=True)

            self.plotting = {
                "times": deque([0], maxlen=10000),
                "currents": deque([measured_current_0], maxlen=10000),
                "voltages": deque([measured_voltage_0], maxlen=10000),
                "min_current": float('inf'),
                "max_current": float('-inf'),
                "min_voltage": float('inf'),
                "max_voltage": float('-inf')
            }

            controls_state = {
                "prev_func": None,
                "init_time": start_time,
                "curr_time": start_time,
            }
```

```
        while self.controls_running:
            time.sleep(DELTA_T - ((time.perf_counter() - start_time) % DELTA_T))  # run no faster than DELTA_T

            # update time and measurements
            curr_time = time.perf_counter() - start_time
            measured_current = self.measure_current()
            measured_voltage = self.measure_voltage(CHANNEL_CELL_UNDER_TEST)
            self.set_measured_current(measured_current)
            self.set_measured_voltage(measured_voltage)

            past_times.appendleft(curr_time)
            past_currents.appendleft(measured_current)
            past_voltages.appendleft(measured_voltage)

            #
            voltage_buffer_bottom = self.measure_voltage(CHANNEL_BUFFER_BOTTOM)
            voltage_buffer_top = self.measure_voltage(CHANNEL_BUFFER_TOP)
            self.set_voltage_buffer_bottom(voltage_buffer_bottom)
            self.set_voltage_buffer_top(voltage_buffer_top)

            # if measurements out of ABSOLUTE ratings, immediately stop and break
            if not between(measured_current, ABS_MIN_CURRENT, ABS_MAX_CURRENT):
                raise Exception('measured current {} not in range ({}, {})'.format(measured_current, ABS_MIN_CURRENT, ABS_MAX_CURRENT))
            if not between(measured_voltage, ABS_MIN_VOLTAGE, ABS_MAX_VOLTAGE):
                raise Exception('measured voltage {} not in range ({}, {})'.format(measured_voltage, ABS_MIN_VOLTAGE, ABS_MAX_VOLTAGE
```

```
))
            # or (not between(voltage_buffer_bottom, ABS_MIN_VOLTAGE_BUFFER, ABS_MAX_VOLTAGE_BUFFER)) \
            # or (not between(voltage_buffer_top, ABS_MIN_VOLTAGE_BUFFER, ABS_MAX_VOLTAGE_BUFFER)):

            if len(self.queue) != 0:
                try:  # in case command.execute throws an exception
                    self.lock.acquire()
                    command = self.queue.popleft()
                    if command.execute != controls_state["prev_func"]:
                        controls_state["prev_func"] = command.execute
                        controls_state["init_time"] = curr_time
                    controls_state["curr_time"] = curr_time

                    success = command.execute(controls_state["curr_time"] - controls_state["init_time"], self)
                    if not success:
                        self.queue.appendleft(command)
                    self.lock.release()
                except Exception as e:
                    logger.error(e, exc_info=True)
                    raise

            desired_current = self.get_desired_current()
            desired_voltage = self.get_desired_voltage()

            if self.get_control_mode() == Tester.ControlMode.current:
                # calculate p, i, d terms
                error = desired_current - measured_current
                diff = (past_currents[0] - past_currents[M_BACK_CC]) / (past_times[0] - past_times[M_BACK_CC])
                p_term = K_P_CC * error
```

```
                d_term = clip(K_D_CC * diff, -abs(p_term), abs(p_term))  # make sure that abs of derivative term not greater than abs of proportional term
                i_term = clip(i_term + K_I_CC*error*(past_times[0]-past_times[1]), MIN_V_PWM, MAX_V_PWM)
                command = clip(p_term - d_term + i_term, MIN_V_PWM, MAX_V_PWM)
            else:  # Tester.ControlMode.voltage
                # calculate command for the voltage control,
                # as well as commands for current control at +/- self.get_abs_current_limit()
                # command voltage
                error_volt = desired_voltage - measured_voltage
                diff_volt = (past_voltages[0] - past_voltages[M_BACK_VC]) / (past_times[0] - past_times[M_BACK_VC])
                p_term_volt = K_P_VC * error_volt
                d_term_volt = clip(K_D_VC * diff_volt, -abs(p_term_volt), abs(p_term_volt))
                i_term_volt = clip(i_term + K_I_VC * error_volt * (past_times[0] - past_times[1]), MIN_V_PWM, MAX_V_PWM)
                command_volt = p_term_volt - d_term_volt + i_term_volt

                # command current min
                error_min_curr = -self.get_abs_current_limit() - measured_current
                diff_min_curr = (past_currents[0] - past_currents[M_BACK_CC]) / (past_times[0] - past_times[M_BACK_CC])
                p_term_min_curr = K_P_CC * error_min_curr
                d_term_min_curr = clip(K_D_CC * diff_min_curr, -abs(p_term_min_curr), abs(p_term_min_curr))
                i_term_min_curr = clip(i_term + K_I_CC*error_min_curr*(past_times[0]-past_times[1]), MIN_V_PWM, MAX_V_PWM)
                command_min_curr = p_term_min_curr - d_term_min_curr + i_term_min_curr

```

```
                    # command current max
                    error_max_curr = self.get_abs_current_limit() - measured_current
                    diff_max_curr = (past_currents[0] - past_currents[M_BACK_CC]) / (past_times[0] - past_times[M_BACK_CC])
                    p_term_max_curr = K_P_CC * error_max_curr
                    d_term_max_curr = clip(K_D_CC * diff_max_curr, -abs(p_term_max_curr), abs(p_term_max_curr))
                    i_term_max_curr = clip(i_term + K_I_CC*error_max_curr*(past_times[0]-past_times[1]), MIN_V_PWM, MAX_V_PWM)
                    command_max_curr = p_term_max_curr - d_term_max_curr + i_term_max_curr

                    # current limit, update integral term
                    command = clip(command_volt, command_min_curr, command_max_curr)
                    if command == command_volt:
                        p_term = p_term_volt
                        d_term = d_term_volt
                        i_term = i_term_volt
                    elif command == command_min_curr:
                        p_term = p_term_min_curr
                        d_term = d_term_min_curr
                        i_term = i_term_min_curr
                    else:  # command_max
                        p_term = p_term_max_curr
                        d_term = d_term_max_curr
                        i_term = i_term_max_curr
                    command = clip(command, MIN_V_PWM, MAX_V_PWM)

                # actuate
                self.set_desired_v_pwm(command)
                self.set_actual_v_pwm(self.a_out(CHANNEL_V_PWM, self.get_desired_v_pwm()))

                # for writing
```

```
                sum_measured_currents_writing += measured_current
                sum_measured_voltages_writing += measured_voltage
                sum_actual_v_pwm_writing += self.get_actual_v_pwm()
                i_writing += 1
                if i_writing == MULTIPLE_WRITING:
                    try:
                        row_to_write = [display_float(curr_time, 2),
                                        display_float(
sum_measured_currents_writing / MULTIPLE_WRITING, 3),
                                        display_float(
sum_measured_voltages_writing / MULTIPLE_WRITING, 3),
                                        display_float(
sum_actual_v_pwm_writing / MULTIPLE_WRITING, 3)]
                        if DEBUG and MULTIPLE_WRITING == 1:
                            row_to_write += [p_term, d_term, i_term]
                        writer.writerow(row_to_write)
                    except PermissionError as e:
                        logger.error(e, exc_info=True)
                    i_writing = 0
                    sum_measured_currents_writing = 0
                    sum_measured_voltages_writing = 0
                    sum_actual_v_pwm_writing = 0

                # for plotting
                sum_measured_currents_plotting += measured_current
                sum_measured_voltages_plotting += measured_voltage
                i_plotting += 1
                if i_plotting == MULTIPLE_PLOTTING:
                    self.message_passing.put_nowait((curr_time,

sum_measured_currents_plotting / MULTIPLE_PLOTTING,

sum_measured_voltages_plotting / MULTIPLE_PLOTTING))
                    i_plotting = 0
                    sum_measured_currents_plotting = 0
                    sum_measured_voltages_plotting = 0
```

```
            file.close()
        except Exception as e:
            logger.error(e, exc_info=True)
        finally:
            # set v_pwm back to nominal before returning
            self.set_desired_v_pwm(NOMINAL_V_PWM)
            self.set_actual_v_pwm(self.a_out(CHANNEL_V_PWM, self.
    get_desired_v_pwm()))

            self.stop()
        logger.info('returning from _controls')
        return

    def _periodically_ping(self):
        self.after(30*1000, self._periodically_ping)
        if not self.controls_running:
            logger.info('periodically pinging')
            self.set_desired_v_pwm(NOMINAL_V_PWM)
            self.set_actual_v_pwm(self.a_out(CHANNEL_V_PWM, self.
    get_desired_v_pwm()))
        return

    ##########
    # gui
    ##########

    def create_gui(self):
        """
        Create the GUI.
        """
        self._create_gui_device_label()
        self._create_gui_frame_desired()
        self._create_gui_frame_measured()
        self._create_gui_frame_other()
        self._create_gui_frame_plot()
        self._create_gui_frame_buttons()
```

```

    def _create_gui_device_label(self):
        device_label = tk.Label(self)
        device_label["text"] = ('Board Number ' + str(self.board_num)
                                + ": " + self.device_info.product_name
                                + " (" + self.device_info.unique_id + ")")
        device_label.pack(fill=tk.NONE, anchor=tk.NW)
        return

    def _create_gui_frame_desired(self):
        return

    def _create_gui_frame_measured(self):
        frame = tk.LabelFrame(self, text="Measured", padx=30, pady=3)
        frame.pack(fill=tk.X, anchor=tk.W, padx=3, pady=3)
        return self._create_gui_frame_measured_helper(frame)

    def _create_gui_frame_measured_helper(self, parent_frame):
        # measured current
        measured_current_label = tk.Label(parent_frame, text="current:")
        measured_current_label.grid(row=0, column=0, sticky=tk.W)
        self.master.measured_current_value = tk.Label(parent_frame)
        self.master.measured_current_value.grid(row=0, column=1)

        # measured voltage
        measured_voltage_label = tk.Label(parent_frame, text="voltage:")
        measured_voltage_label.grid(row=1, column=0, sticky=tk.W)
        self.master.measured_voltage_value = tk.Label(parent_frame)
        self.master.measured_voltage_value.grid(row=1, column=1)

        # v_pwm
        actual_v_pwm_label = tk.Label(parent_frame, text="v_pwm:")
        actual_v_pwm_label.grid(row=2, column=0, sticky=tk.W)
```

```
        self.master.actual_v_pwm_value = tk.Label(parent_frame)
        self.master.actual_v_pwm_value.grid(row=2, column=1)
        return

    def _create_gui_frame_other(self):
        frame_other = tk.LabelFrame(self, text="Other", padx=30, pady=3)
        frame_other.pack(fill=tk.X, anchor=tk.W, padx=3, pady=3)

        # voltage buffer cell bottom
        voltage_buffer_bottom_label = tk.Label(frame_other, text="buffer bottom:")
        voltage_buffer_bottom_label.grid(row=3, column=0, sticky=tk.W)
        self.master.voltage_buffer_bottom_value = tk.Label(frame_other)
        self.master.voltage_buffer_bottom_value.grid(row=3, column=1)

        # voltage buffer cell top
        voltage_buffer_top_label = tk.Label(frame_other, text="buffer top:")
        voltage_buffer_top_label.grid(row=4, column=0, sticky=tk.W)
        self.master.voltage_buffer_top_value = tk.Label(frame_other)
        self.master.voltage_buffer_top_value.grid(row=4, column=1)
        return

    def _create_gui_frame_plot(self):
        self.fig = plt.figure()
        self.fig.tight_layout()
        self.canvas = FigureCanvasTkAgg(self.fig, self)
        self.canvas.get_tk_widget().pack(side="top", fill='both', expand=True)

        self.ax_voltage = self.fig.add_subplot(111)
        self.ax_voltage.set_xlabel('time [s]')
        self.ax_voltage.set_ylabel('cell voltage [V]')
        self.ax_voltage.tick_params(axis='y', colors='blue')
        self.ax_voltage.yaxis.label.set_color('blue')
```

```
        self.line_voltage, = self.ax_voltage.plot([], [], 'b')

        self.ax_current = self.ax_voltage.twinx()
        self.ax_current.set_ylabel('cell current [A]')
        self.ax_current.tick_params(axis='y', colors='red')
        self.ax_current.yaxis.label.set_color('red')
        self.line_current, = self.ax_current.plot([], [], 'r')
        return

    def _create_gui_frame_buttons(self):
        frame_buttons = tk.Frame(self)
        frame_buttons.pack(fill=tk.X, side=tk.RIGHT, anchor=tk.SE)

        # start button
        self.master.start_button = tk.Button(frame_buttons, text="Start
    ", bg='green')
        self.master.start_button["command"] = self.start
        self.master.start_button.grid(row=0, column=0, padx=3, pady=3)

        # quit button
        quit_button = tk.Button(frame_buttons, text="Quit")
        quit_button["command"] = self.quit
        quit_button.grid(row=0, column=1, padx=3, pady=3)
        return

    ##########
    # gui button callbacks
    ##########

    def start(self):
        """
        Callback for start_button.
        """
        # measure and verify voltages of cells in range
        measured_voltage = self.measure_voltage(CHANNEL_CELL_UNDER_TEST
    )
```

```
        if not between(measured_voltage, ABS_MIN_VOLTAGE, ABS_MAX_VOLTAGE):
            tk.messagebox.showerror("Error", ("Cell voltage reading from channel {} is {}."
                                     " Please ensure a valid connection and try again.")
                                     .format(CHANNEL_CELL_UNDER_TEST, measured_voltage))
            return

        """
        voltage_buffer_bottom = self.measure_voltage(CHANNEL_BUFFER_BOTTOM)
        if not between(voltage_buffer_bottom, ABS_MIN_VOLTAGE_BUFFER, ABS_MAX_VOLTAGE_BUFFER):
            tk.messagebox.showerror("Error", ("Cell voltage reading from channel {} is {}."
                                     " Please ensure a valid connection and try again.")
                                     .format(CHANNEL_BUFFER_BOTTOM, voltage_buffer_bottom))
            return

        voltage_buffer_top = self.measure_voltage(CHANNEL_BUFFER_TOP)
        if not between(voltage_buffer_top, ABS_MIN_VOLTAGE_BUFFER, ABS_MAX_VOLTAGE_BUFFER):
            tk.messagebox.showerror("Error", ("Cell voltage reading from channel {} is {}."
                                     " Please ensure a valid connection and try again.")
                                     .format(CHANNEL_BUFFER_TOP, voltage_buffer_top))
            return
        """

        self.controls_running = True
```

```
        self.master.start_button["command"] = self.stop
        self.master.start_button["text"] = "Stop"
        self.master.start_button["bg"] = 'red'
        return

    def stop(self):
        """
        Callback for stop button.
        """
        self.controls_running = False
        self.master.start_button["command"] = self.start
        self.master.start_button["text"] = "Start"
        self.master.start_button["bg"] = 'green'
        return

    def quit(self):
        """
        Callback for quit button.
        """
        if tk.messagebox.askokcancel("Quit", "Are you sure you want to quit?"):
            self.stop()
            self.program_quit = True
            self._quit()

    def _quit(self):
        if self.thread_controls not in threading.enumerate():
            self.master.destroy()
        else:
            self.after(10, self._quit)  # check again in 10 ms to prevent deadlock
        return


##########
# ManualTester class
```

```
##########

class ManualTester(Tester):
    """
    Manual control version of the program.
    """

    def __init__(self):
        super().__init__()
        self.queue = deque([SetConstantCurrentUntilMeasuredVoltage(NOMINAL_CURRENT, ComparisonOperator.false)], maxlen=1)
        self.check_rep()

    ##########
    # gui
    ##########

    def _create_gui_frame_desired(self):
        self.frame_desired_measured = tk.Frame(self)
        self.frame_desired_measured.pack(fill=tk.X)
        self.frame_desired_measured.grid_columnconfigure(0, weight=1, uniform="group1")
        self.frame_desired_measured.grid_columnconfigure(1, weight=1, uniform="group1")
        self.frame_desired_measured.grid_rowconfigure(0, weight=1)

        parent_frame = tk.LabelFrame(self.frame_desired_measured, text="Desired", padx=30, pady=3)
        parent_frame.grid(row=0, column=0, sticky="nsew", padx=3, pady=3)

        radio_button_options = [("Current (A):", "cc"), ("Voltage (V):", "vc")]
        for row, (text, value) in enumerate(radio_button_options):
            rb = tk.Radiobutton(parent_frame, text=text,
```

```
                                variable=self.control_mode_var, value=value, command=self._radio_command)
            rb.grid(row=row, column=0, sticky=tk.W)

        # desired current
        float_icmd = self.register(self.validate_desired_current_entry)
        self.master.desired_current_entry = tk.Entry(
            parent_frame, validate='key', validatecommand=(float_icmd, '%P'), width=8)
        self.master.desired_current_entry.grid(row=0, column=1)
        self.master.desired_current_entry.insert(0, display_float(self.get_desired_current(), 3))
        self.master.desired_current_entry.bind("<Return>", self.update_desired_current_entry)
        self.master.desired_current_entry.bind("<FocusOut>", self.sync_desired_current_entry)

        self.master.desired_current_update_button = tk.Button(parent_frame, text="Update")
        self.master.desired_current_update_button["command"] = self.update_desired_current_entry
        self.master.desired_current_update_button.grid(row=0, column=2, padx=3, pady=3)

        # desired voltage
        float_vcmd = self.register(self.validate_desired_voltage_entry)
        self.master.desired_voltage_entry = tk.Entry(
            parent_frame, validate='key', validatecommand=(float_vcmd, '%P'), width=8)
        self.master.desired_voltage_entry.grid(row=1, column=1)
        self.master.desired_voltage_entry.insert(0, display_float(self.get_desired_voltage(), 3))
        self.master.desired_voltage_entry.bind("<Return>", self.update_desired_voltage_entry)
        self.master.desired_voltage_entry.bind("<FocusOut>", self.sync_desired_voltage_entry)
```

```
        self.master.desired_voltage_update_button = tk.Button(
    parent_frame, text="Update")
        self.master.desired_voltage_update_button["command"] = self.
    update_desired_voltage_entry
        self.master.desired_voltage_update_button.grid(row=1, column=2,
     padx=3, pady=3)
        return

    def _create_gui_frame_measured(self):
        frame = tk.LabelFrame(self.frame_desired_measured, text="
    Measured", padx=30, pady=3)
        frame.grid(row=0, column=1, sticky="nsew", padx=3, pady=3)
        return self._create_gui_frame_measured_helper(frame)

    ##########
    # gui other
    ##########

    @staticmethod
    def validate_desired_current_entry(entry):
        """
        Return True if the entry is blank, a period, a minus sign, a
    minus sign followed by a period,
        or a number, else False.
        """
        # the user may not have started typing, or could be typing in a
     decimal or negative number
        if entry == '' or entry == '.' or entry == '-' or entry == '-.'
     or is_numeric(entry):
            return True
        return False

    @staticmethod
    def validate_desired_voltage_entry(entry):
        """
```

```
        Return True if the entry is blank, or a positive number, else
    False.
        """
        # the user may not have started typing
        if entry == '' or (is_numeric(entry) and float(entry) > 0):
            return True
        return False

    def update_desired_current_entry(self, event=None):
        """
        Add to queue constant current command.
        Clip and display contents of desired_current_entry to 3 decimal
     places if necessary.
        """
        desired_current_entry_text = self.master.desired_current_entry.
    get()
        if is_numeric(desired_current_entry_text):
            desired_current = clip(float(desired_current_entry_text),
    MIN_CURRENT, MAX_CURRENT)
            self.lock.acquire()
            self.queue.append(SetConstantCurrentUntilMeasuredVoltage(
    desired_current, ComparisonOperator.false))
            self.lock.release()
            self.replace_entry_text(self.master.desired_current_entry,
    display_float(desired_current, 3))
        self.check_rep()

    def sync_desired_current_entry(self, event=None):
        """
        Update desired_current_entry value to that of self.
    desired_current.
        """
        self.replace_entry_text(self.master.desired_current_entry,
    display_float(self.get_desired_current(), 3))
        self.check_rep()

```

```python
    def update_desired_voltage_entry(self, event=None):
        """
        Add to queue constant voltage command.
        Clip and display contents of desired_voltage_entry to 3 decimal places if necessary.
        """
        desired_voltage_entry_text = self.master.desired_voltage_entry.get()
        if is_numeric(desired_voltage_entry_text):
            desired_voltage = clip(float(desired_voltage_entry_text), MIN_VOLTAGE, MAX_VOLTAGE)
            self.lock.acquire()
            self.queue.append(SetConstantVoltageUntilMeasuredVoltage(desired_voltage, ComparisonOperator.false, abs_current_limit=10))
            self.lock.release()
            self.replace_entry_text(self.master.desired_voltage_entry, display_float(desired_voltage, 3))
        self.check_rep()

    def sync_desired_voltage_entry(self, event=None):
        """
        Update desired_voltage_entry value to that of self.desired_voltage.
        """
        self.replace_entry_text(self.master.desired_voltage_entry, display_float(self.get_desired_voltage(), 3))
        self.check_rep()

    @staticmethod
    def replace_entry_text(entry, new_text):
        """
        Replace the text in entry with new_text.
        """
        entry.delete(0, tk.END)
        entry.insert(0, str(new_text))
        return
```

```

    def _radio_command(self):
        if self.control_mode_var.get() == "cc":
            self.update_desired_current_entry()
        else:  # vc
            self.update_desired_voltage_entry()


##########
# CycleTester class
##########

class CycleTester(Tester):
    """
    Cycling version of the program.
    """

    def __init__(self):
        super().__init__()

        # no GUI for this, so change sequences and cycling_commands to what you want here
        sequences = {
            1: SetConstantVoltageUntilMeasuredVoltage(3.4, ComparisonOperator.between, 3.39, 3.41, abs_current_limit=5),
            2: SetConstantCurrentUntilMeasuredVoltage(-10, ComparisonOperator.less_than_equal, 3.05),
            3: SetConstantCurrentUntilMeasuredVoltage(10, ComparisonOperator.greater_than_equal, 3.5),
            4: SetConstantVoltageUntilMeasuredCurrent(3.8, ComparisonOperator.less_than_equal, 2),
            5: SetConstantCurrentUntilMeasuredVoltage(-5, ComparisonOperator.less_than_equal, 3.4),
            6: SetConstantVoltageUntilMeasuredVoltage(3.4, ComparisonOperator.false),
        }
```

```
        cycling_commands = [
            (1, 1, 1),  # (start_seq, end_seq, repeat)
            (2, 4, 1),
            (5, 6, 1),
        ]

        """
        sequences = {
            1: SetConstantVoltage(3.4, 5),
            2: SetConstantCurrentUntilMeasuredVoltage(-10, ComparisonOperator.less_than_equal, 1.5),
            3: SetConstantVoltageUntilMeasuredCurrent(1.5, ComparisonOperator.greater_than_equal, -0.025),
            4: SetConstantCurrentUntilMeasuredVoltage(10, ComparisonOperator.greater_than_equal, 3.8),
            5: SetConstantVoltageUntilMeasuredCurrent(3.8, ComparisonOperator.less_than_equal, 1),
            6: SetConstantCurrentUntilMeasuredVoltage(-10, ComparisonOperator.less_than_equal, 3.4),
            7: SetConstantVoltage(3.4, 5),
        }
        cycling_commands = [
            (1, 1, 1),  # (start_seq, end_seq, repeat)
            (2, 5, 1),
            (6, 7, 1),
        ]
        """

        for (start_seq, end_seq, repeat) in cycling_commands:
            for _ in range(repeat):
                for i in range(start_seq, end_seq+1):
                    self.queue.append(sequences[i])

        self.check_rep()
```

```python
##########
# CyclingCommand and ComparisonOperator classes
##########

class CyclingCommand(abc.ABC):
    @abc.abstractmethod
    def execute(self, elapsed_time: float, tester: Tester) -> bool:
        pass


class SetConstantCurrentUntilMeasuredVoltage(CyclingCommand):
    def __init__(self, desired_current: float, comparison_operator, *operator_args):
        assert desired_current >= MIN_CURRENT
        assert desired_current <= MAX_CURRENT

        self.desired_current = desired_current
        self.comparison_operator = comparison_operator
        self.operator_args = operator_args

    def execute(self, elapsed_time, tester):
        if elapsed_time == 0:
            tester.set_control_mode(Tester.ControlMode.current)
            tester.set_desired_current(self.desired_current)
        return self.comparison_operator(tester.get_measured_voltage(), *self.operator_args)


class SetConstantVoltageUntilMeasuredVoltage(CyclingCommand):
    def __init__(self, desired_voltage: float, comparison_operator, *operator_args, abs_current_limit=abs(MAX_CURRENT)):
        assert desired_voltage >= MIN_VOLTAGE
        assert desired_voltage <= MAX_VOLTAGE
        assert abs_current_limit >= MIN_CURRENT
        assert abs_current_limit <= MAX_CURRENT

```

```python
        self.desired_voltage = desired_voltage
        self.comparison_operator = comparison_operator
        self.operator_args = operator_args
        self.abs_current_limit = abs_current_limit

    def execute(self, elapsed_time, tester):
        if elapsed_time == 0:
            tester.set_control_mode(Tester.ControlMode.voltage)
            tester.set_desired_voltage(self.desired_voltage)
            tester.set_abs_current_limit(self.abs_current_limit)
        return self.comparison_operator(tester.get_measured_voltage(), *self.operator_args)


class SetConstantVoltageUntilMeasuredCurrent(CyclingCommand):
    def __init__(self, desired_voltage: float, comparison_operator, *operator_args):
        assert desired_voltage >= MIN_VOLTAGE
        assert desired_voltage <= MAX_VOLTAGE

        self.desired_voltage = desired_voltage
        self.comparison_operator = comparison_operator
        self.operator_args = operator_args

    def execute(self, elapsed_time, tester):
        if elapsed_time == 0:
            tester.set_control_mode(Tester.ControlMode.voltage)
            tester.set_desired_voltage(self.desired_voltage)
            tester.set_abs_current_limit(abs(MAX_CURRENT))
        return self.comparison_operator(tester.get_measured_current(), *self.operator_args)


class ComparisonOperator:
    @staticmethod
    def less_than(a, b) -> bool:
```

```
        """
        Return True if a < b else False.
        """
        return operator.lt(a, b)

    @staticmethod
    def less_than_equal(a, b) -> bool:
        """
        Return True if a <= b else False.
        """
        return operator.le(a, b)

    @staticmethod
    def greater_than_equal(a, b) -> bool:
        """
        Return True if a >= b else False.
        """
        return operator.ge(a, b)

    @staticmethod
    def greater_than(a, b) -> bool:
        """
        Return True if a > b else False.
        """
        return operator.gt(a, b)

    @staticmethod
    def between(val, min_val, max_val) -> bool:
        """
        Return True if min_val <= val <= max_val, else False.
        """
        return min_val <= val <= max_val

    @staticmethod
    def false(val) -> bool:
        """
```

```
        Return False.
        """
        return False


##########
# main
##########

if __name__ == "__main__":
    if sys.version_info < (3, 8):
        raise Exception('Python 3.8 or higher required.')

    # logging
    logger = logging.getLogger(__name__)
    logger.setLevel(logging.DEBUG)
    f_handler = logging.FileHandler('log123.log')
    f_format = logging.Formatter('%(asctime)s - %(levelname)s - %(
   message)s')
    f_handler.setFormatter(f_format)
    logger.addHandler(f_handler)

    logger.info("Hello, World!")

    if '-cycle' in sys.argv:
        tester1 = CycleTester()
    else:
        tester1 = ManualTester()
    tester1.master.mainloop()

    logger.info("Bye!")
```

Listing E.1: Source code for the cell tester circuit.

Bibliography

[1] Y. Shang, Q. Zhang, N. Cui, and C. Zhang, "A Cell-to-Cell Equalizer Based on Three-Resonant-State Switched-Capacitor Converters for Series-Connected Battery Strings," *Energies*, vol. 10, no. 2, p. 206, Feb. 2017. [Online]. Available: http://www.mdpi.com/1996-1073/10/2/206

[2] D. Mahoney, D. Longo, J. Heinzel, and J. McGlothin, "Advanced Shipboard Energy Storage System," in *ASNE EMTS Symposium*, Philadelphia, PA, May 2012. [Online]. Available: https://apps.dtic.mil/sti/pdfs/ADA577181.pdf

[3] Y. Yu, R. Saasaa, A. A. Khan, and W. Eberle, "A Series Resonant Energy Storage Cell Voltage Balancing Circuit," *IEEE Journal of Emerging and Selected Topics in Power Electronics*, vol. 8, no. 3, pp. 3151–3161, Sep. 2020. [Online]. Available: https://ieeexplore.ieee.org/document/8706971/

[4] Y. Barsukov and J. Qian, "Cell-Balancing Techniques: Theory and Implementation," in *Battery power management for portable devices*, ser. Artech House power engineering series. Boston: Artech House, 2013, pp. 111–138.

[5] Y. Shang, C. Zhang, N. Cui, and J. M. Guerrero, "A Cell-to-Cell Battery Equalizer With Zero-Current Switching and Zero-Voltage Gap Based on Quasi-Resonant LC Converter and Boost Converter," *IEEE Transactions on Power Electronics*, vol. 30, no. 7, pp. 3731–3747, Jul. 2015. [Online]. Available: http://ieeexplore.ieee.org/document/6872814/

[6] Texas Instruments and Y. Barsukov, "Battery Cell Balancing: What to Balance and How." [Online]. Available: https://www.ti.com/download/trng/docs/seminar/Topic%202%20-%20Battery%20Cell%20Balancing%20-%20What%20to%20Balance%20and%20How.pdf

[7] M. Uno, "Single- and Double-Switch Cell Voltage Equalizers for Series-Connected Lithium-Ion Cells and Supercapacitors," in *Energy Storage - Technologies and Applications*, A. Zobaa, Ed. InTech, Jan. 2013. [Online]. Available: https://www.intechopen.com/chapters/42277

[8] C. Pascual and P. T. Krein, "Switched capacitor system for automatic series battery equalization," in *Proceedings of APEC 97 - Applied Power Electronics Conference*, vol. 2. Atlanta, GA, USA: IEEE, 1997, pp. 848–854. [Online]. Available: http://ieeexplore.ieee.org/document/575744/

[9] Texas Instruments, "LM266x Switched Capacitor Voltage Converter," Oct. 2014. [Online]. Available: https://www.ti.com/lit/ds/snvs002e/snvs002e.pdf?ts=1623105759671

[10] Maxim Integrated, "MAX889 High-Frequency, Regulated, 200mA, Inverting Charge Pump," Jul. 2000. [Online]. Available: https://datasheets.maximintegrated.com/en/ds/MAX889.pdf

[11] Analog Devices, "LT1054/LT1054L Switched-Capacitor Voltage Converter with Regulator," Jan. 2020. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/LT1054-1054L.pdf

[12] W. Kester, B. Erisman, and G. Thandi, "Section 4: Switched Capacitor Voltage Converters," in *Practical Design Techniques for Power and Thermal Management*. Analog Devices, 1998. [Online]. Available: https://www.analog.com/media/en/training-seminars/design-handbooks/Practical-Design-Techniques-Power-Thermal/Section4.pdf

[13] ROHM Semiconductor, "Electronics Basics: Total Gate Charge." [Online]. Available: https://www.rohm.com/electronics-basics/transistors/total-gate-charge

[14] Microchip Technology and A. Hussain, "AN786: Driving Power MOSFETs in High-Current, Switch Mode Regulators," 2002. [Online]. Available: http://ww1.microchip.com/downloads/cn/AppNotes/cn_00786a.pdf

[15] Y. Yuanmao, K. W. E. Cheng, and Y. P. B. Yeung, "Zero-Current Switching Switched-Capacitor Zero-Voltage-Gap Automatic Equalization System for Series Battery String," *IEEE Transactions on Power Electronics*, vol. 27, no. 7, pp. 3234–3242, Jul. 2012. [Online]. Available: http://ieeexplore.ieee.org/document/6112683/

[16] K.-M. Lee, Y.-C. Chung, C.-H. Sung, and B. Kang, "Active Cell Balancing of Li-Ion Batteries Using \$LC\$ Series Resonant Circuit," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 9, pp. 5491–5501, Sep. 2015. [Online]. Available: http://ieeexplore.ieee.org/document/7054535/

[17] Y. Shang, C. Zhang, N. Cui, J. M. Guerrero, and K. Sun, "A crossed pack-to-cell equalizer based on quasi-resonant LC converter with adaptive fuzzy logic equalization control for series-connected lithium-ion battery strings," in *2015 IEEE Applied Power Electronics Conference and Exposition (APEC)*. Charlotte, NC, USA: IEEE, Mar. 2015, pp. 1685–1692. [Online]. Available: http://ieeexplore.ieee.org/document/7104574/

[18] P. Scherz and S. Monk, *Practical Electronics for Inventors*, 4th ed. New York: McGraw-Hill Education, 2016.

[19] Electronics Tutorials, "Op-amp Multivibrator." [Online]. Available: https://www.electronics-tutorials.ws/opamp/op-amp-multivibrator.html

[20] ITECH Electronics, “IT-M3400 Bidirectional DC Power Supply.” [Online]. Available: https://www.itechate.com/en/product/dc-power-supply/IT-M3400.html

[21] ON Semiconductor, “NVMYS1D3N04C MOSFET – Power, Single N-Channel 40 V, 1.15 m, 252 A,” Jul. 2019. [Online]. Available: https://www.onsemi.com/pdf/datasheet/nvmys1d3n04c-d.pdf

[22] Vishay Siliconix, “SQD40031EL Automotive P-Channel 30 V (D-S) 175 °C MOSFET,” Mar. 2018. [Online]. Available: https://www.vishay.com/docs/76011/sqd40031el.pdf

[23] Nexperia, “PSMNR70-30YLH N-channel 30 V, 0.82 m, 300 A logic level MOSFET in LFPAK56 using NextPowerS3 technology,” Nov. 2019. [Online]. Available: https://assets.nexperia.com/documents/data-sheet/PSMNR70-30YLH.pdf

[24] Infineon Technologies, “IPD90P03P4-04 OptiMOS®-P2 Power-Transistor,” Jul. 2008. [Online]. Available: https://www.infineon.com/dgdl/Infineon-IPD90P03P4_04-DS-v01_00-en.pdf?fileId=db3a30431ddc9372011e07ecafdb27ed

[25] S. Leeb, “Lecture 4: MOSFET Switches and Drive,” MIT, 2017.

[26] International Rectifier, “Application Note AN-944: Use Gate Charge to Design the Gate Drive Circuit for Power MOSFETs and IGBTs.” [Online]. Available: https://www.infineon.com/dgdl/Infineon-Use_Gate_Charge_to_Design_the_Gate_Drive_Circuit_for_Power_MOSFETs_and_IGBTs-AN-v01_00-EN.pdf?fileId=5546d46267354aa001673ba630970081

[27] CT-Concept Technologie AG, “Application note AN-1001: IGBT and MOSFET Drivers Correctly Calculated,” Apr. 2016. [Online]. Available: https://www.power.com/sites/default/files/product_document/application_note/AN-1001_IGBT_and_MOSFET_Drivers_Correctly_Calculated.pdf

[28] L. Balogh, “Application Report SLUA618A: Fundamentals of MOSFET and IGBT Gate Driver Circuits,” Oct. 2018. [Online]. Available: https://www.ti.com/lit/ml/slua618a/slua618a.pdf

[29] D. Perreault, “Thermal Modeling and Heat Sinking,” MIT, 2020.

[30] J. Wilson, “Thermal Conductivity of Solders,” Aug. 2006. [Online]. Available: https://www.electronics-cooling.com/2006/08/thermal-conductivity-of-solders/

[31] International Rectifier, “Application Note AN-0994: Maximizing the Effectiveness of your SMD Assemblies,” May 2012. [Online]. Available: https://www.infineon.com/dgdl/Infineon-an-994-AN-v06_00-EN.pdf?fileId=5546d46265f064ff01667ab5829d4d44

[32] Advanced Circuits, “PCB Trace Width Calculator.” [Online]. Available: https://www.4pcb.com/trace-width-calculator.html

[33] All About Circuits, “Trace Resistance Calculator.” [Online]. Available: https://www.allaboutcircuits.com/tools/trace-resistance-calculator/

[34] Freescale Semiconductor, A. Arendarik, and R. pod Radhoštem, “Application Note AN4428: Active Cell Balancing in Battery Packs,” Jan. 2012. [Online]. Available: https://www.nxp.com/docs/en/application-note/AN4428.pdf

[35] B. Beauregard, “Improving the Beginner’s PID – Introduction,” Apr. 2011. [Online]. Available: http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/