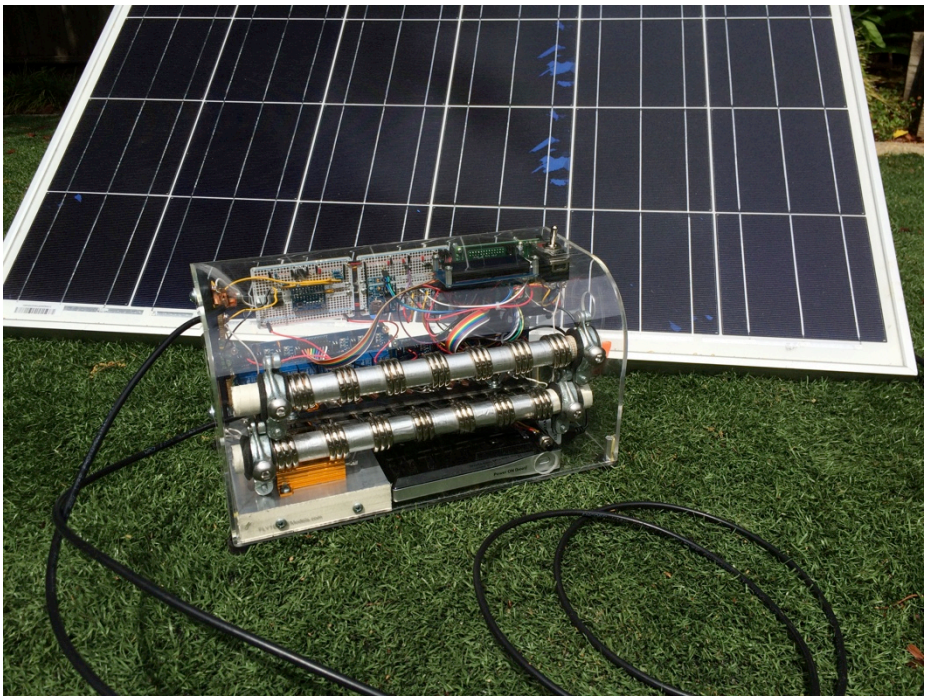
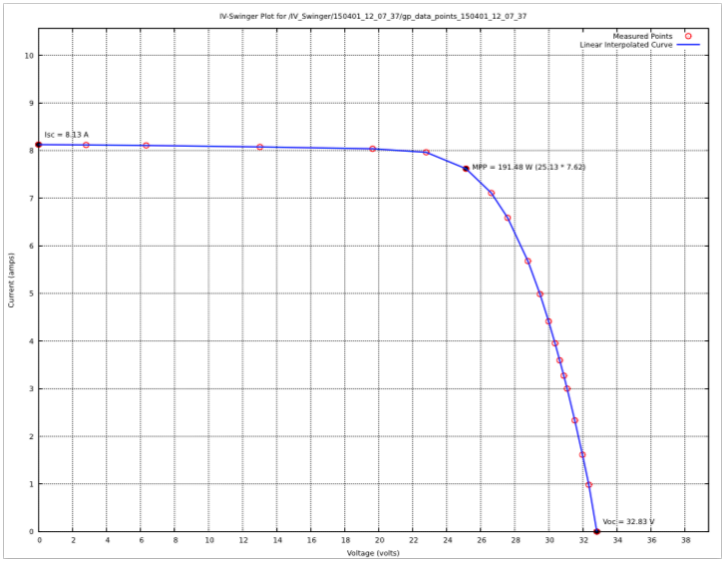
IV Swinger

Design, Construction and Operation

Document Revision: 1.1 (13-Apr, 2016) Chris Satterlee





IV Swinger is an open source hardware and software project.

Current versions of the license files, documentation, Fritzing file (hardware description), and software can be found at:

https://github.com/csatt/IV_Swinger

Table of Contents

Table of Figures

Table of Equations

1 Introduction

This document contains a detailed description of the design, construction and operation of the IV Swinger. It is assumed that the reader is already familiar with what the IV Swinger is and what it is used for. At a minimum, the reader should have read the "IV Swinger User Guide" before reading this document. Additionally it is highly recommended that the reader has viewed the following video on YouTube:

IV Swinger Demo: https://www.youtube.com/watch?v=xNytkONOcW0

The following two YouTube videos provide some background on IV curves and the motivation for designing and building the IV Swinger. They are optional viewing:

IV Swinger Background Part I: https://www.youtube.com/watch?v=xrC5VoMxGJM
IV Swinger Background Part II: https://www.youtube.com/watch?v=0MmQlo-HBuE

They are optional because anyone reading this document is most likely already well versed in everything discussed in those videos.

The following video is optional but recommended:

IV Swinger Design and Construction: https://www.youtube.com/watch?v=m6l7vpuYwQ0

Everything in that video will be discussed in much more detail in this document, yet the video could be useful to get a "show and tell" overview before diving in.

This document is intended to provide enough information for readers to build their own IV Swingers. However, (with some exceptions) it does not contain step-by-step instructions on how to do so. Due to component and materials availability it is unlikely that it will be possible to create exact clones, and it will become less and less likely as more time passes.

1.1 GitHub Repository / Licensing

All of the IV Swinger code and documentation (including this document) are available in a public GitHub repository at https://github.com/csatt/IV_Swinger. You can use your web browser to look at the files and download them to your computer. Or, if you have git installed on your computer, you can clone the entire repository with the following command:

```
git clone https://github.com/csatt/IV_Swinger.git
```

If you are cloning the repository on a Raspberry Pi that will be used in your own IV Swinger, run the clone command from the /home/pi directory and the files will land in their proper places.

1.2 Design Objectives

The objectives of the IV Swinger design were the following:

- **To be an educational tool**

The initial target use of the IV Swinger is for Gil Masters' "Electric Power: Renewables and Efficiency" (CEE 176B) course at Stanford. But any college-level (or possibly high-school level) course that covers photovoltaic IV curves could benefit from having an IV Swinger.

- **To be low cost**

Commercial IV curve tracers such as the Solmetric PVA-1000S cost over $5000. This is beyond the budget for most college courses. The objective for the IV Swinger was to have a total cost of parts in the low hundreds of dollars. The labor cost is assumed to be zero - anyone building one should be doing it for the fun of it (or possibly for academic credit).

- **To support a single modern PV solar panel**

Commercial IV curve tracers can handle the high voltage and power of a whole string of panels in series. This is not necessary for the experiments that are currently performed in an academic lab setting. The IV Swinger is designed to handle a single PV solar panel with I_{SC} <= 10A, V_{OC} <= 80V and P_{mpp} <= 450W.

- **To be portable**

IV curve tracing experiments are of course performed outdoors where the sun shines, possibly on a rooftop. The location may be out of the range of an extension cord. Therefore a design goal was for the IV Swinger to be battery-powered and small and light enough to be carried by hand.

- **To be easy to use**

A typical student using the IV Swinger will only use it a small number of times, so a long learning curve would be counterproductive. An important design goal was to make it as simple and intuitive to use as possible.

- **To have the internals visible from outside**

The IV Swinger's transparent acrylic case exposes all of its innards to be seen by the user. This is more than just to make it look "cool" (which admittedly it does). It is also so it is not just a "black box" that magically spits out IV curves. Students who have traced an IV curve manually can pretty easily see that the IV Swinger is just a machine that automates the same process they did by hand.

The following were considered to be of secondary importance and will either never be met by the IV Swinger design or are deferred to a future revision or variant:

- **To be durable**

The IV Swinger is not particularly durable in its current incarnation. The acrylic case would not survive a drop. Many of the electrical connections are not soldered.

- **To support strings of panels**

Supporting a whole string of panels would extend the utility of the IV Swinger beyond the academic realm and potentially add some possibilities within the academic realm. This is not a goal now, but if it could be done without substantially increasing the cost it would be a nice enhancement.

- **To instantly display the IV curve**

Adding a small graphical display to the IV Swinger would allow it to show a low resolution IV curve at the time the measurement is taken. This would give the user immediate feedback without having to transfer the thumb drive to a computer. This would add some cost and additional hardware and software work but would be a nice enhancement.

- **To be productizable**

There are no plans for productization of the IV Swinger. The overriding goal is to be a small *pro bono* contribution to educating the world on how PV solar panels work, which hopefully will contribute to continued growth of this most promising of renewable energy technologies.

1.3 Where Did the Name Come From?

The name "IV Swinger" comes from the expression "swinging out an IV curve", which is how Gil Masters refers to the process of plotting an IV curve using the manual method (light bulb load bank, ammeter, voltmeter). Other people talk about "tracing" or "sweeping out" an IV curve, but I believe Gil is unique in his use of the "swinging" terminology. It is such a reflection of his enthusiasm and positive attitude! It sounds so fast and fun! The reality is that doing it manually is slow and labor intensive - more like "slogging out an IV curve" if you ask me. So I named this device the IV Swinger in the hope that it would make tracing an IV curve as fast and fun as Gil made doing it manually sound.

2 Overview

2.1 High-level Block Diagram

The diagram in Figure 2-1 below represents the IV Swinger at a high level.

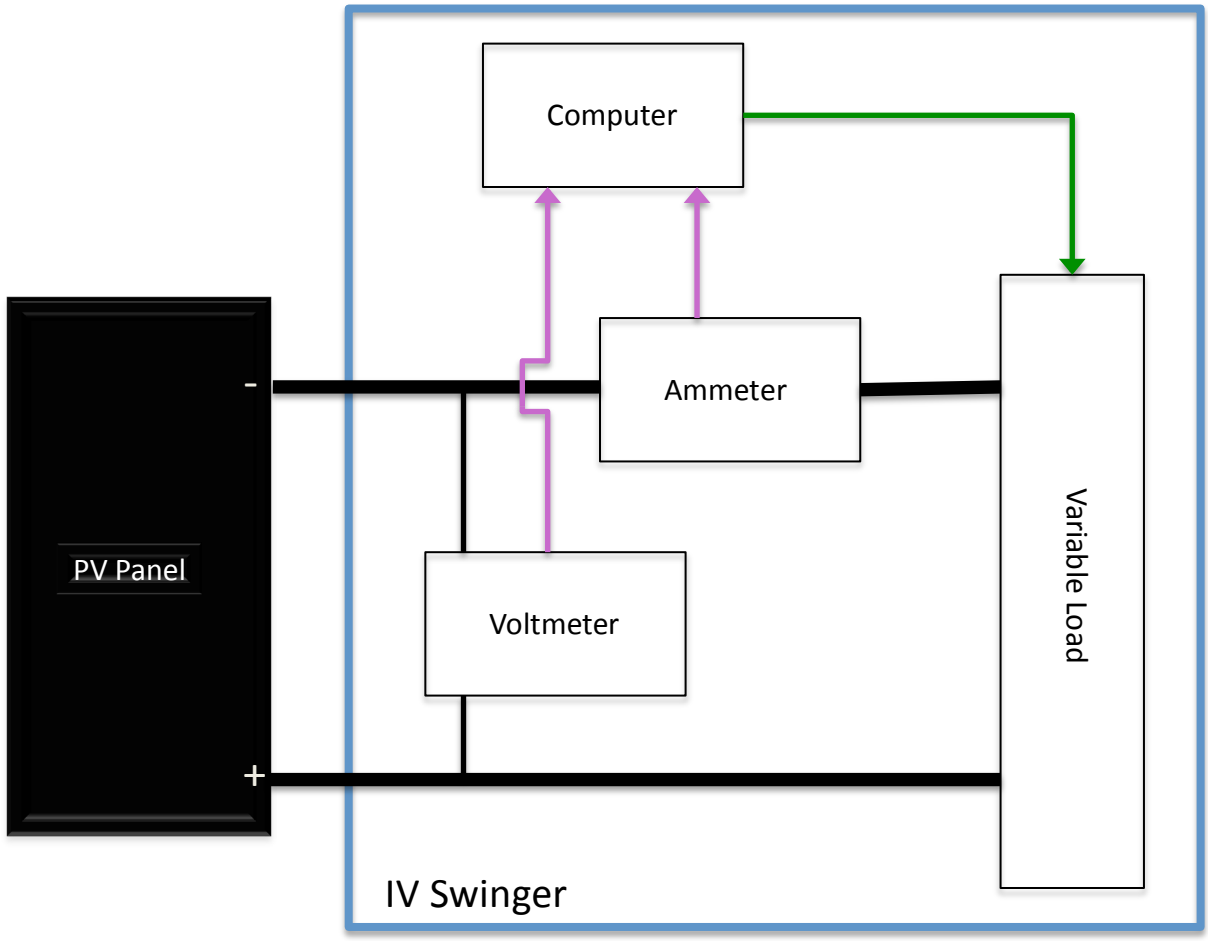


Figure 2-1: High-level Block Diagram

2.2 High-level Description

The components in Figure 2-1 can be directly mapped to a human generating an IV curve manually using a load bank and a multimeter. The variable load maps to the light bulb bank and its switches (or potentiometer/rheostat and its knob). The ammeter and voltmeter map to the multimeter. The computer maps to the human.

The bold lines from the PV panel to the variable load represent the load circuit. These are the wires that carry the current generated by the PV panel to and from the load. The ammeter is in series on one leg of the load circuit, measuring the current. The voltmeter is in parallel, between the outputs of the PV panel, measuring the voltage. The computer controls the resistance of the variable load (green arrow) and reads the values from the ammeter and voltmeter (pink arrows).

The variable load is implemented with a chain of immersion heating coils and power resistors. Relays are used to either include or exclude (bypass) each of the loads in the chain. Software running on the computer controls the relays to incrementally increase the resistance of the variable load. At each increment it reads the current and voltage values and records them. The resulting set of data points are used to plot the IV curve.

2.3 Detailed Drawing and Schematic

Figure 2-2 below is a detailed drawing created using Fritzing, which is a wonderful (and free) tool. This drawing shows all of the components of the IV Swinger and how they are connected. Later sections of this document will "zoom in" to different parts of this drawing. This figure itself is high resolution, so if you are reading a soft copy of this document, you can zoom into the figure with your computer and see the details.

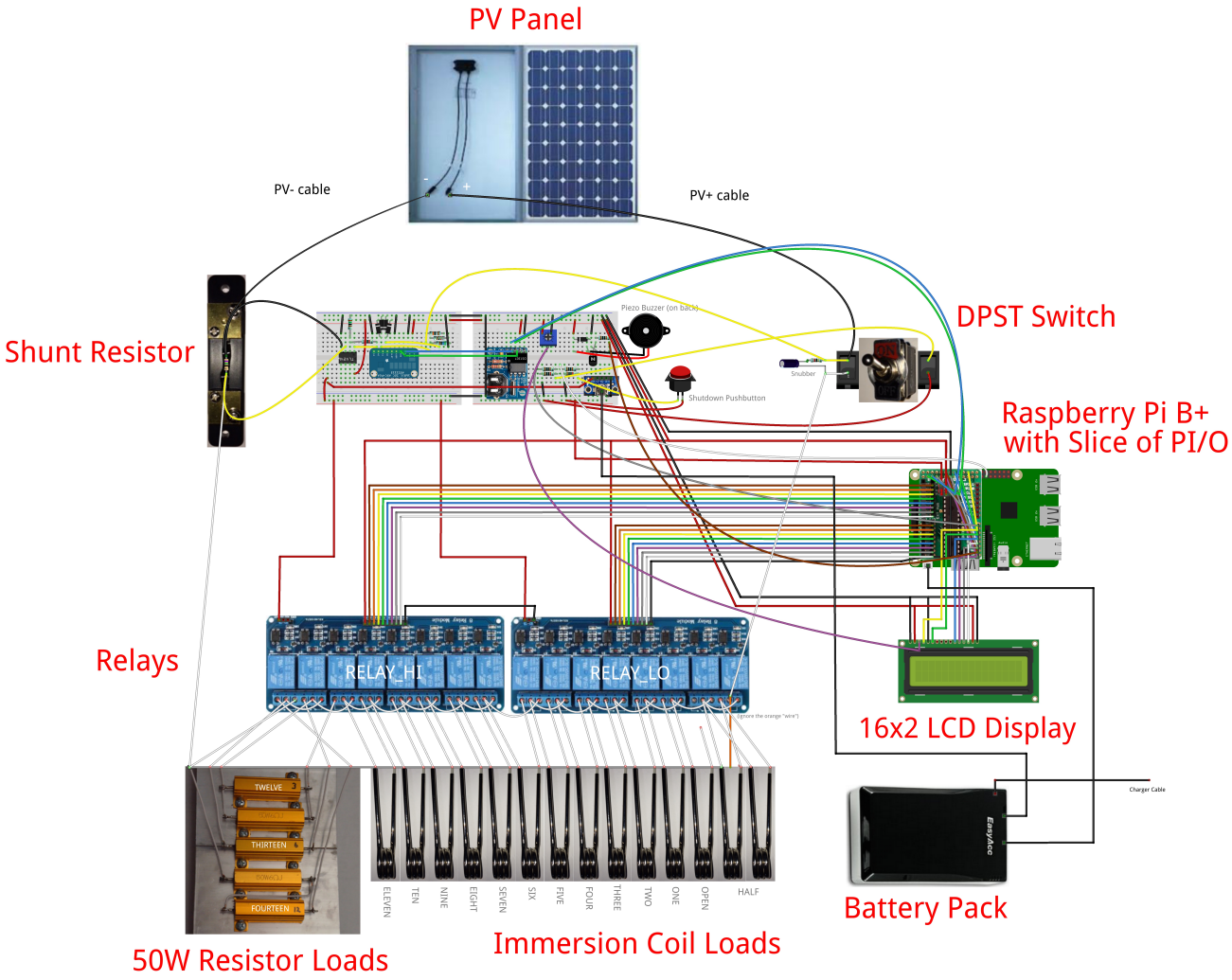


Figure 2-2: Detailed Drawing

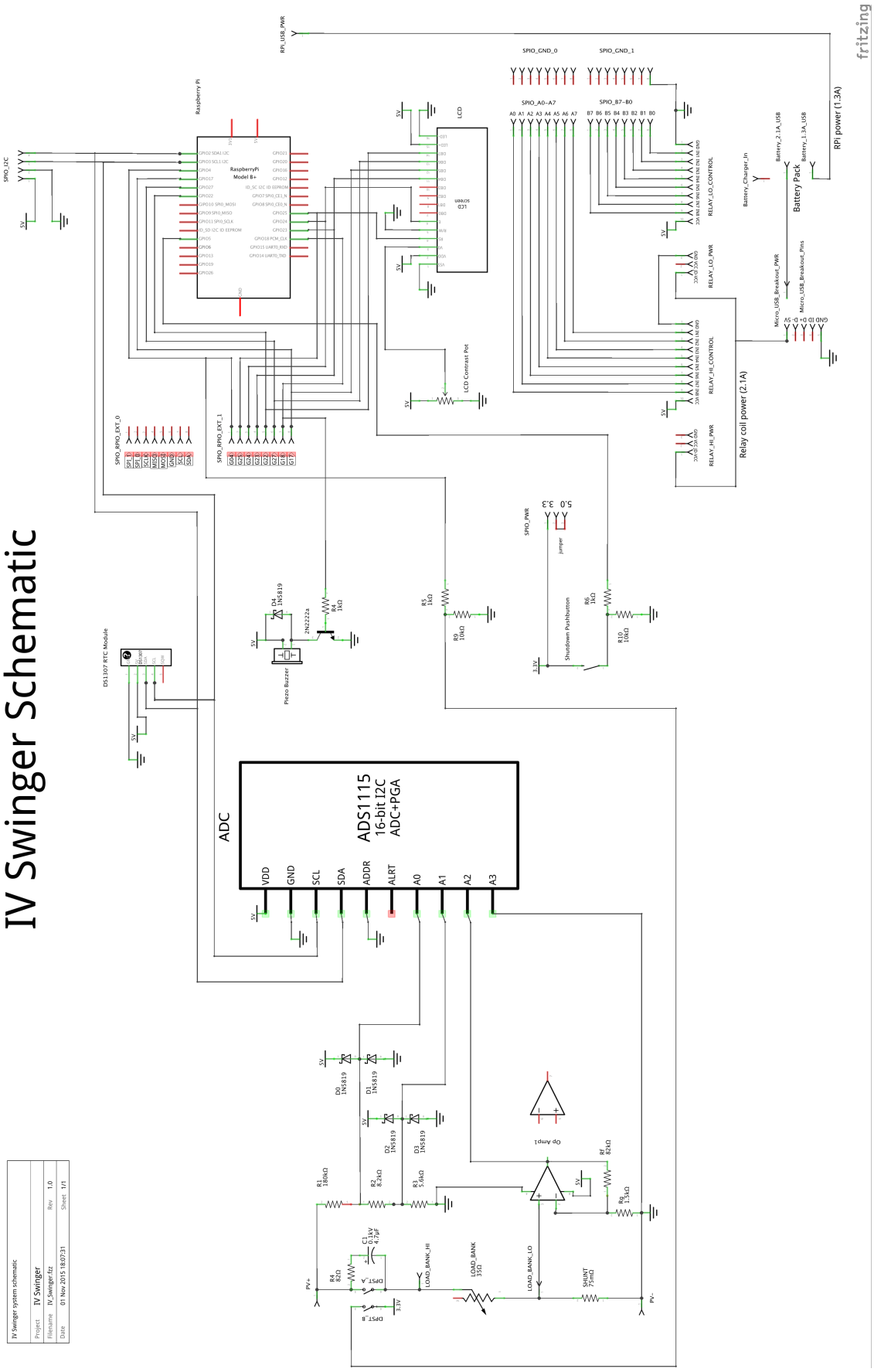


Figure 2-3: Schematic Diagram

Figure 2-3 above is a schematic diagram, also generated with Fritzing. In fact the two figures are just different views of the same design so they are guaranteed to be consistent with each other. As with Figure 2-2, later sections of this document will "zoom in" to different parts of the schematic.

The Fritzing source file used to generate the drawings in Figure 2-2 and Figure 2-3 is included in the IV Swinger GitHub repository (IV_Swinger/Fritzing/IV_Swinger.fzz). If you are building an IV Swinger, it is recommended that you install Fritzing and explore the IV Swinger design using the tool in addition to reading this document. You will be able to zoom in as far as you need to in order to see all the components and their connections.

2.4 Photographs: 6 Views

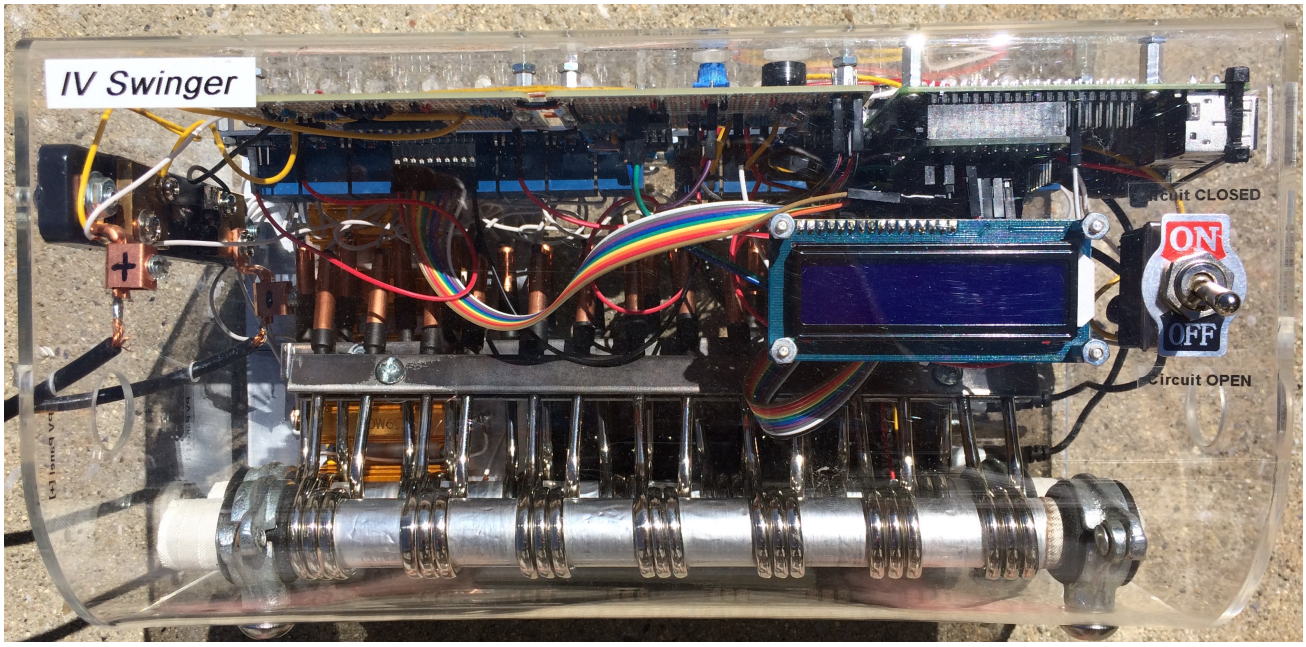


Figure 2-4: Top View

Figure 2-5: Front View

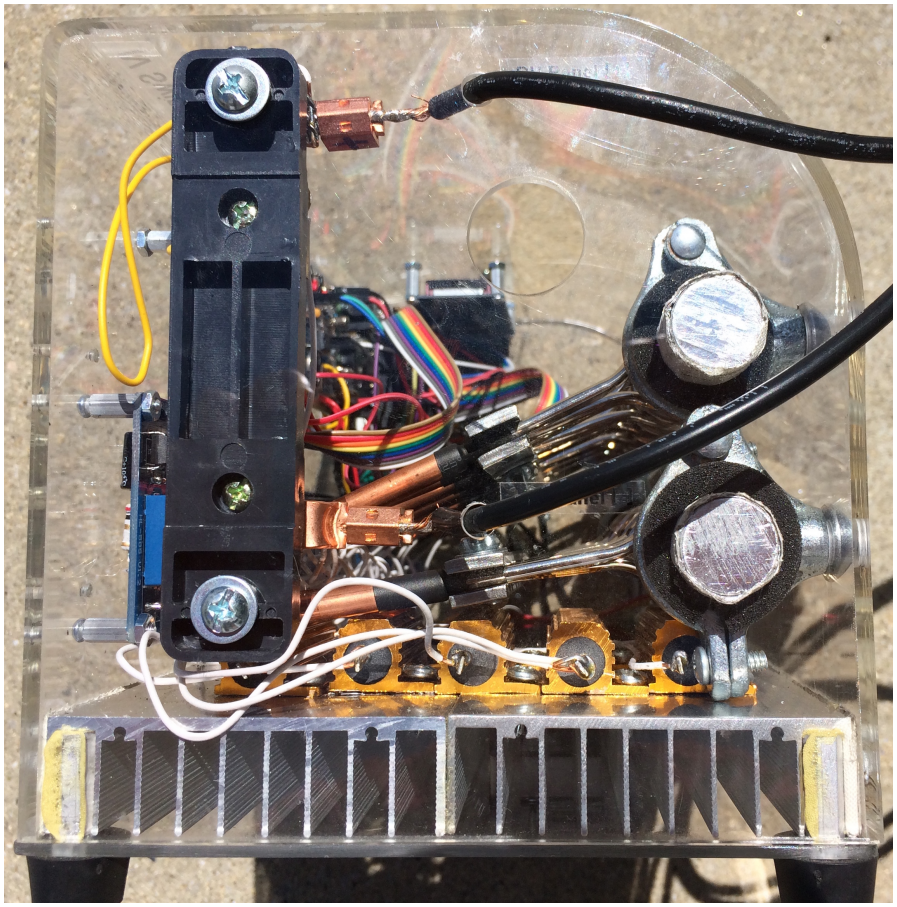


Figure 2-6: Left Side View

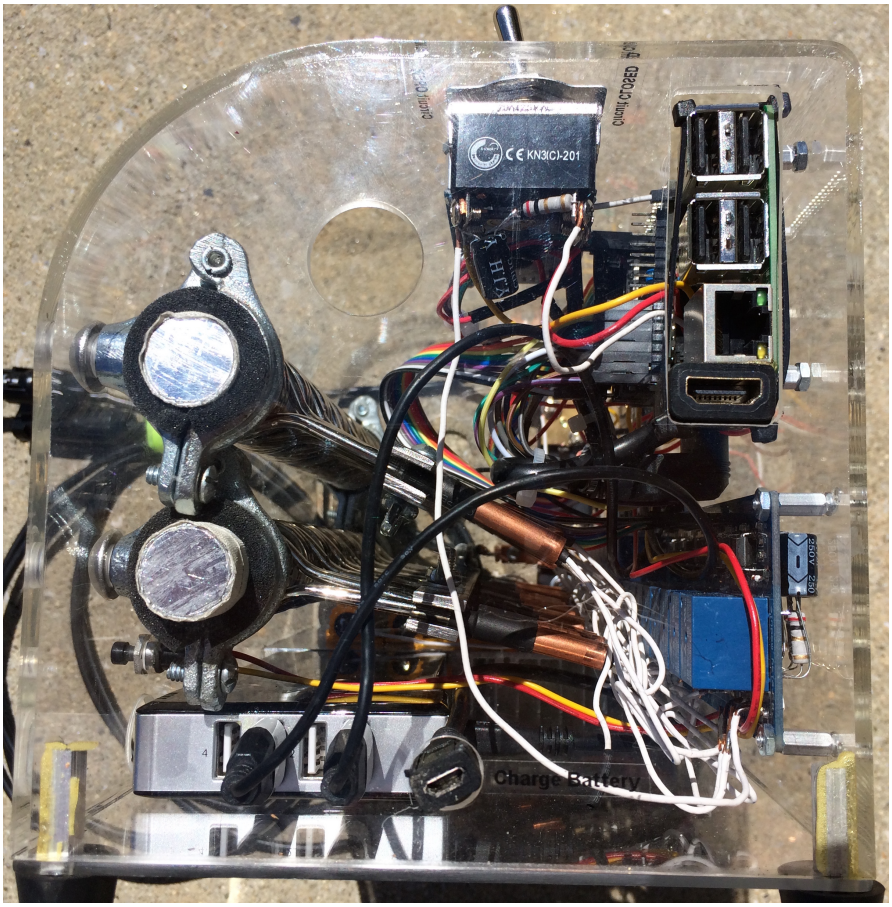


Figure 2-7: Right Side View

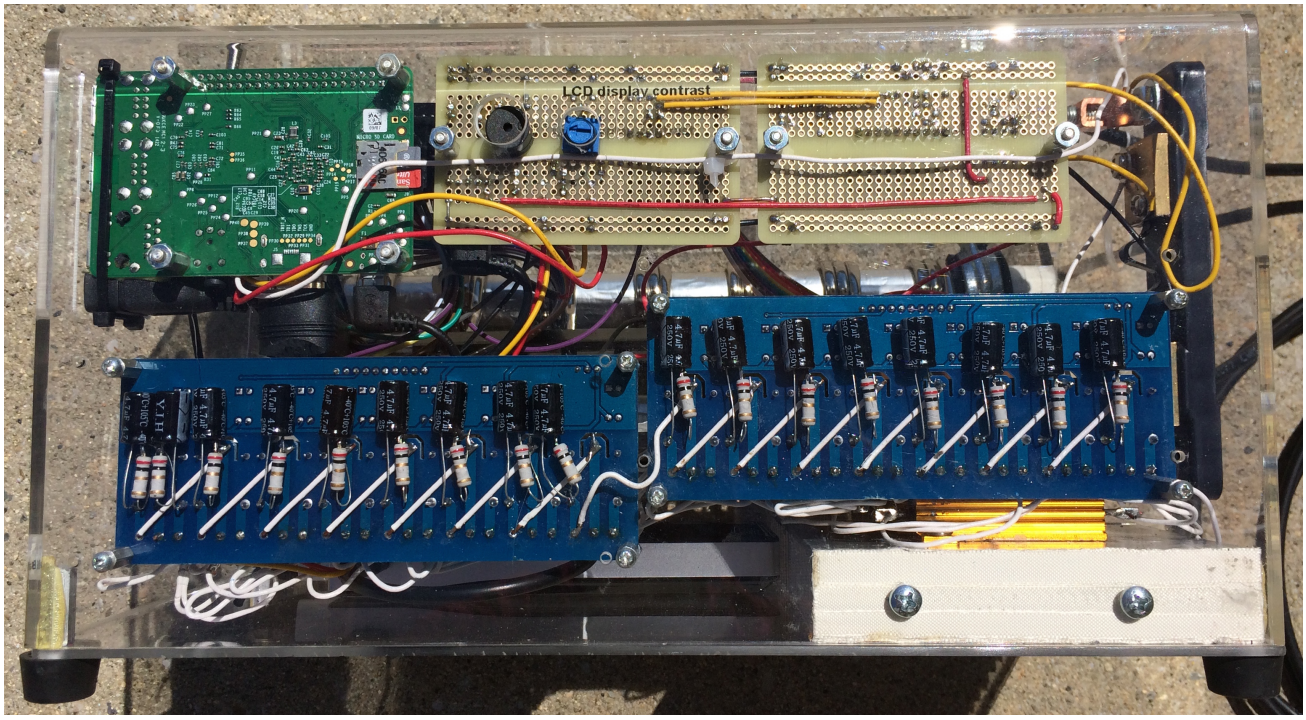


Figure 2-8: Back View

Figure 2-9: Bottom View

3 Variable Load Circuit

The variable load circuit carries the current generated by the PV panel. It starts at the positive MC4 connector and ends at the negative MC4 connector. Its components are: the cables, one side of the double-pole single-throw (DPST) switch, the relay banks, the immersion coil loads, the 50 watt power resistor loads, and the shunt resistor (part of the ammeter).

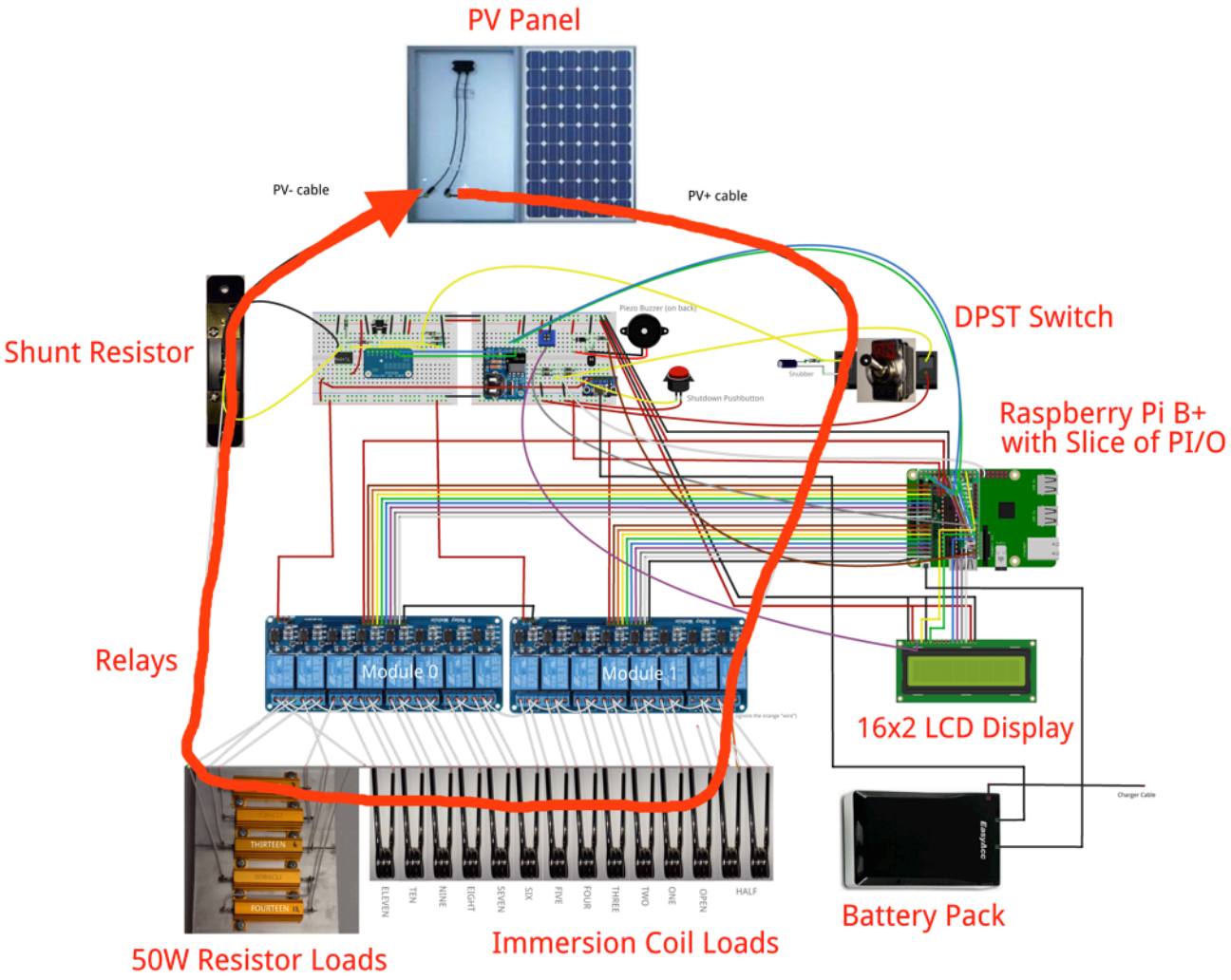


Figure 3-1: Variable Load Circuit

The red line in Figure 3-1 above shows the variable load circuit. Of course the exact path through the relays and loads varies depending on the relay controls.

3.1 Load Circuit Requirements

The requirements of the variable load circuit are the following:

- Electronic controllability
- Support for a wide range of IV curves (range, resolution)
- Adequate power dissipation
- Reasonable size
- Low cost
- Longevity

3.1.1 Electronic controllability

A fundamental purpose of the IV Swinger is to be able to take all of the measurements in a small amount of time so that conditions cannot change significantly between the first and last measurement. Any human actions during the "swinging" of the IV curve are unacceptable. This implies that the computer must be capable of changing the resistance of the load electronically.

Continuous or discrete?

The ability to vary the load in arbitrarily small increments would be very desirable, but was not a requirement for the first incarnation of IV Swinger. This is something to explore as a future enhancement. It might be possible using a PWM-controlled buck-boost converter such as those used in maximum power point trackers. (This is discussed further in Section 12.2 on page 113).

3.1.2 Support for a wide range of IV curves (range, resolution)

The height of an IV curve is its I_{SC} value and the width is its V_{OC} value. The panel type and conditions determine I_{SC} and V_{OC}. The resistance range, number of increments, and values per increment of the variable load must be chosen carefully in order to generate good IV curves. It is important to have load points on the part(s) of the curve that are inflecting. An example of a "bad" IV curve would be one where all of the sampled points are on the flat nearly horizontal part at the top. The graph would look like a straight line to the last sampled point and then another straight line diagonally down to the V_{OC} point. The knee of the curve, which is the interesting part, would be chopped off and the IV curve would be pretty useless. Figure 3-2 below shows this case. The GOOD curve and the BAD curve have the same number of sampled points, but the points on the BAD curve don't reach the knee. The resistance of the maximum load value is insufficient for the V_{OC} value of 35 volts in this case. If V_{OC} had been 15 volts, everything would have been fine.

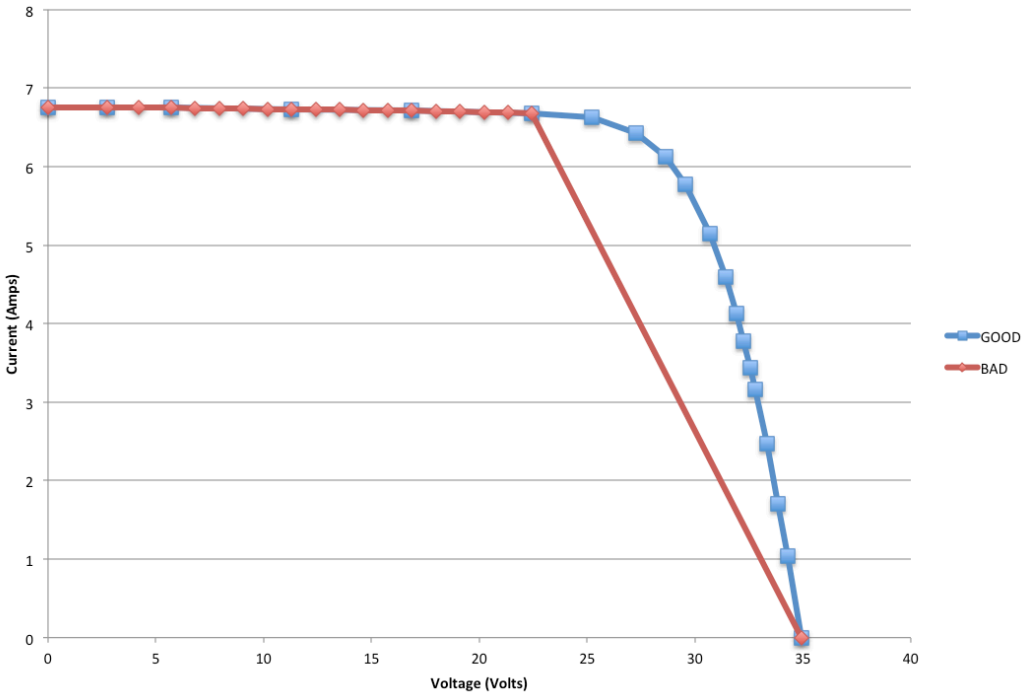


Figure 3-2: Insufficient Range Example 1

Knowing the maximum V_{OC} value does not by itself determine the required range of the variable load. A low I_{SC} value can also cause the same problem. I_{SC} is proportional to the insolation, so a panel receiving 500 W/m^2 (50% of the rated "full sun" value of 1000 W/m^2) will have an I_{SC} of 50% of the rated I_{SC} value. The whole curve is shifted down by a constant number of amps (50% of the I_{SC} - it is not "scaled" by 50%). Figure 3-3 below shows the IV curve for the same panel at full sun insolation and at half sun insolation. The diagonal lines are constant resistance lines. You can see that with these eleven load values a pretty reasonable IV curve would be generated for the full sun case. But with the same eleven load values all the points on the half sun case are before the knee of the curve, so the generated graph would again look similar to the BAD one in Figure 3-2. This tells us that looking at the rated I_{SC} values for solar panels that we want to support is not sufficient if we want to get good results for those panels when they are exposed to low insolation.

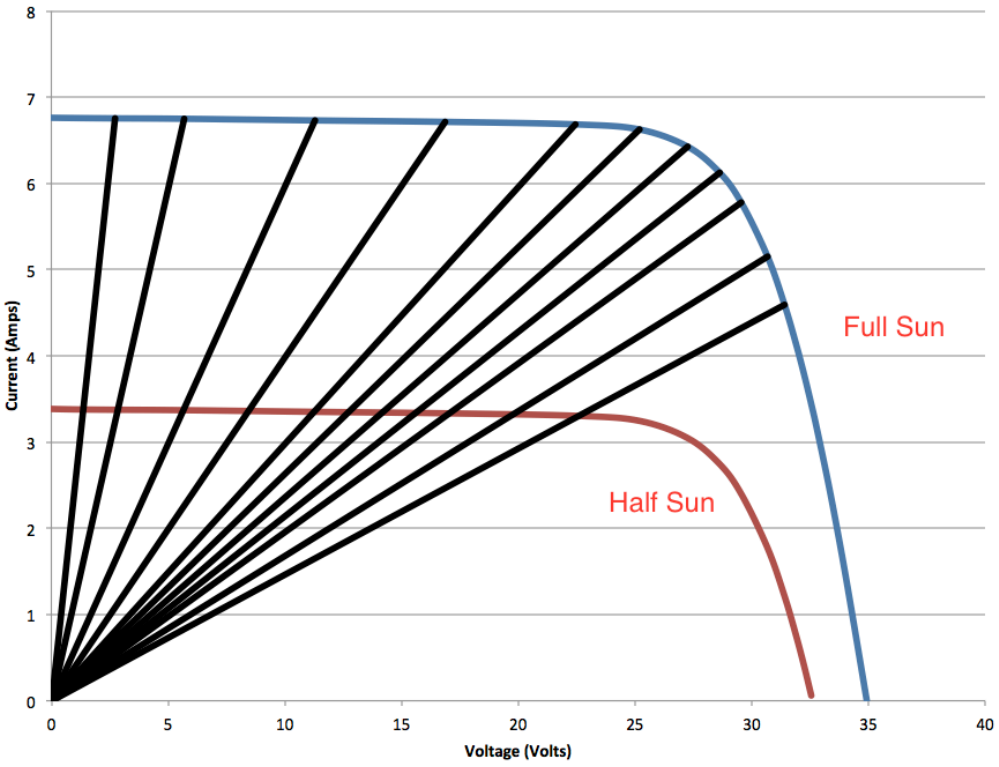


Figure 3-3: Insufficient Range Example 2

Something else to recognize is that small increments in resistance are very desirable for the beginning of the curve, but we want larger increments for the end of the curve. This is because the slope of the constant resistance lines is I/V (reciprocal of the resistance, i.e. conductance). For us to get points that are down near the V_{OC} end of the curve, we need high resistances. Using a single increment of resistance throughout the range is not ideal. For example, 0.5Ω might be a good increment at the beginning of the curve (I_{SC} end), but at the end of the curve, adding 0.5Ω produces a point that is so close to its predecessor that it provides no added value.

To put some bounds on the problem, it is necessary to identify some representative solar panels for which good results are desired. A great resource WAS http://solar-panels.findthebest.com but that sadly appears to no longer exist. It had a list of almost 800 solar panels that could be sorted based on rated power, I_{SC}, V_{OC}, etc. In September 2014, the highest power panel was the SunPower 435W which had an I_{SC} of 6.43A and a V_{OC} of 85.6V. Most panels have an $I_{SC} < 10A$ and $V_{OC} < 80V$. But there are some strange ones. Thin film panels have a very high V_{OC} and very low I_{SC} (e.g. V_{OC}=249V, I_{SC} =0.83A). Other technologies are the opposite (e.g. V_{OC}=21V, I_{SC}=17.3A). Covering that whole range is not practical, so it was decided to limit the supported panels to those with $I_{SC} < 10A$ and $V_{OC} < 80V$.

As previously shown in Figure 3-3, a low value of I_{SC} can result in the sampled points not reaching the knee of the curve. Actually, a low value of I_{SC} is fine if V_{OC} is also low. The real problem is IV curves that are “wide and low”, i.e. where the V_{OC}/I_{SC} ratio is large. V_{OC}/I_{SC} is roughly the load value (in ohms) at the knee of the curve (i.e. the maximum power point- MPP). If the maximum resistance of the load is equal to V_{OC}/I_{SC} of the panel, the plotted curve will only reach the MPP, which isn’t really far enough. A good curve can be obtained if Equation 1 below is satisfied.

Equation 1: Maximum required load resistance

$$R_{max} > 3.5 \cdot \frac{V_{oc,rated}}{I_{sc,rated}}$$

The 3.5x multiplier is to cover 50% insolation (2x) and another 1.75x to get sufficiently past the MPP (2 x 1.75 = 3.5).

So now we just need to identify the panel with the highest rated V_{OC}/I_{SC} that we want to accommodate. The SunPower SP333 has a rated V_{OC} =65.3V and a rated I_{SC} =6.46A, so the rated V_{OC}/I_{SC} is about 10 Ω and **R_{max} is 35 Ω**. This is a reasonable target worst case and is the design point used for the IV Swinger.

Figure 3-4 is a photograph of a hand drawing I made when I was trying to work out the requirements for the variable load range and resolution. It shows the full sun and half sun IV curves for three panels: the SP333 (blue), SP200 (green), and REC260 (red). The constant resistance lines were at 1.2Ω increments. This was clearly not enough resolution for the knees of the full-sun SP200 and REC260 cases so I drew two dots on the curves between each of these lines to see how 0.4Ω increments would look, and decided in this empirical way that **0.4Ω increments** would be pretty good.

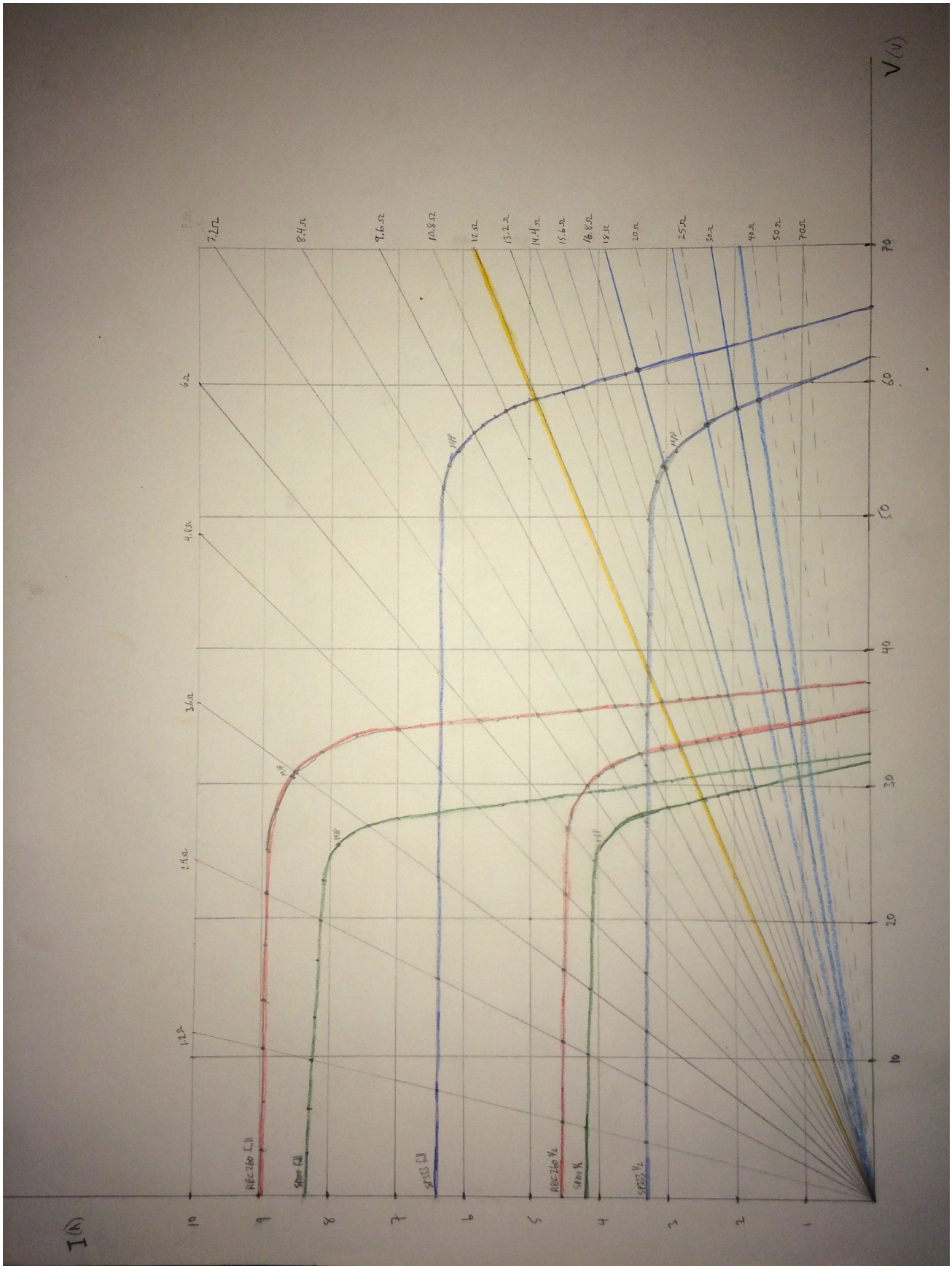


Figure 3-4: Range and Resolution Exploration

3.1.3 Adequate power dissipation

The loads have to be able to handle the power that the solar panel generates.

$$P = I^2R$$

Since the highest current value is I_{SC}, the loads each have to be able to handle I_{SC}^2R, where R is the individual load's resistance. We've decided on a maximum supported I_{SC} of 10A, so the loads each have to be able to handle 100R watts. E.g. a 1Ω load must be able to dissipate 100W.

There are a couple of mitigating factors however. One is the fact that the rated power dissipation of a resistive device indicates how much power it can dissipate *continuously over an indefinite amount of time*. If the power is applied very briefly, there won't be enough time for it to get hot enough to cause a failure even if more than the rated power is applied during that brief amount of time. Since the IV Swinger sweeps the curve pretty quickly, it is almost certain that the rated power of the loads can safely be exceeded. Another mitigating factor is that some of the loads are only used toward the end of the IV curve where the current is much lower than I_{SC}. And of course the maximum rated power of the panel is always an upper limit.

There are two issues with choosing a load with a lower rated power dissipation based on the "short time" argument. One issue is that we can't completely guarantee that the "short time" will not be exceeded. For example, the computer could crash when the circuit is closed with only one load selected. The other issue is that it isn't easy to figure out the relationship between rated power dissipation and time. Some power resistors have a "short-term overload" rating, but it is for a given amount of time such as 5 seconds – not an equation. For this reason, we'll stick with the assumption that each load on the front of the chain must be able to dissipate I_{SC}^2R watts.

The other argument is safe, however. We do want to use higher resistance loads near the end of the curve and it is not necessary for them to each be able to handle the entire I_{SC} current.

It is important to understand what the heat sinking requirements are for the chosen loads. The ability to dissipate the rated power assumes proper heat sinking.

3.1.4 Reasonable size

The portability requirement for the IV Swinger dictates that the load bank (load elements, heat sinks, and switching components) be of reasonable size and weight. Commercial power resistors can be very large and heavy.

3.1.5 Low cost

In addition to being big and heavy, commercial power resistors are expensive. We need a cheaper solution. A key observation is that *the precision of the resistance value is of almost no importance*. Temperature stability of the resistance is also unimportant. This is because both current and voltage are measured at each load value. We don't care what the exact resistance of each load is. This non-requirement makes some much less expensive options possible.

3.1.6 Longevity

Given its target use case, the IV Swinger should last many years even if it can survive only a few hundred uses. Designing it for a few thousand will provide some margin of error.

The loads themselves should be able to withstand a few thousand cycles. The switching elements also need to be able to withstand a few thousand cycles. The latter is the bigger challenge because switching DC is notoriously treacherous.

3.2 Load Circuit Design

3.2.1 Wire

The wire used for the load circuit ideally should have a low enough resistance that its contribution to the loading is minimal. This is important mostly for the case where none of the load elements is selected so the circuit is as close to a short circuit as practical. It also must be able to handle 10A of current safely. The website http://www.powerstream.com/Wire_Size.htm has information on the resistance per length and current carrying capacities of different wire gauges. 18-gauge wire is considered safe for currents up to 16A and has a resistance of 0.006385Ω/ft. For an estimated 20 feet of wire, this is about 1/8Ω. 20-gauge wire is considered safe for currents up to 11A, but its resistance is 60% higher. 16-gauge wire is too large (i.e. stiff) to work with, so 18-gauge wire was chosen.

For reliability/solderability, solid core wire was used. The cheapest 18-gauge solid core wire was thermostat wire, available at Home Depot ($10/50ft). The catch is that it comes as a 2-conductor pair with an outer sheathing, so the sheathing must be removed with care taken not to nick the insulation on the inner wires (I used the tiny scissors on a Swiss Army knife). One of the inner wires has white insulation and the other has red insulation – I only used the white.

A single 18-gauge solid core wire is very easy to bend/manipulate, but when there are many of them (e.g. 28) in parallel the aggregate stiffness is surprising. This can make it challenging to get the assembled load bank inserted into the enclosure. For this reason, it might be preferable to use 20-gauge wire despite the higher resistance. However, by pre-bending the 18-gauge wire into a Z shape, it becomes reasonably easy to insert the load bank into the enclosure, and therefore 18-gauge wire remains the recommendation.

3.2.2 Immersion heating coils

Given the requirement for power dissipation, it was clear that the choice for the load elements would have to be something that gets hot and/or emits light[1]. Power resistors are an obvious choice but they are big and expensive[2]. From the range and resolution exploration (Figure 3-4), we want 25 – 30 loads of around 0.4Ω each. This can be accomplished with a smaller number of larger loads, however, by creating fractional loads. For example, three 1.2Ω loads in parallel is 0.4Ω. Two 0.8Ω loads in parallel is

[1] I did try to think of other possibilities such as doing work or storing the energy somehow, but couldn't think of anything small and cheap that was along those lines.

[2] There are actually smaller, cheaper power resistors than I had found originally. EBay has 100W 1Ω power resistors in the same form factor as the 50W 6Ω power resistors. They are from China and in the $2/each range. Might be worth considering.

0.4Ω. By switching in the fractional load between each full load increment, the effect of having a larger number of the smaller load value can be achieved with a smaller total number of load elements.

100W 12VDC incandescent light bulbs are used in the CEE176B lab for manual IV curve tracing. Their resistance is $V^2/P = 1.44\Omega$. This is close to what we're looking for. But:

- They are pretty bulky
- They don't quite support a 10A I_{SC} (I^2R=144W)
- Their resistance is a bit higher than we'd like
- They are pretty hard to buy these days

The good thing about light bulbs is that they are a consumer item so they are inexpensive (at least they were when you could find them!).

After much searching for something that would meet the requirements, a good solution was finally found: DC immersion heating coils. These are for people to heat their coffee with when they are traveling in an RV or other vehicle with a 12VDC power outlet. From reviews on Amazon, they are a pretty terrible product; they don't get your coffee very hot, they run your battery down, and they burn out very quickly if they are not in liquid while powered on! But they are really remarkably well suited for the IV Swinger load elements.

Figure 3-5: Immersion Coil

3.2.2.1 Power

The rated power of the immersion coils is 120W.

3.2.2.2 Resistance

Based on the power rating of 120W and the voltage of 12V, the resistance of each immersion coil should be $R= V^2/P = 12^2/120 = 1.2\Omega$.

However, the measured resistance ranges from about 0.7Ω to 1.0Ω, with the majority being about **0.8Ω**.

1.2Ω would have worked OK, but 0.8Ω is a very nice value because it means we can create a 0.4Ω resistance by connecting two in parallel. With only one such "half load", we can increment by 0.4Ω[3].

3.2.2.3 Current

Based on the power rating of 120W and voltage of 12V, the rated current is P/V=120/12=10A. This means the heating coils should be able to handle the maximum I_{SC} of 10A that we targeted for the IV Swinger.

3.2.2.4 Thermal

The coils are intended to be immersed in water/coffee/tea, and as the Amazon reviews can attest, they will burn out quickly if they are run at full power in the open air. There must be some way for the generated heat to be transferred away from the heating filament.

Immersing the coils in water might be the first thought, but water is heavy and would also be a challenge to keep from leaking. Another thought would be to entomb the coils in some kind of clay or concrete. In any case, there must be some material in contact with the coils. That material must have good thermal conductivity and thermal mass.

Aluminum has excellent thermal conductivity, which is why it is typically used for heat sinks. Clamping the immersion coils around an aluminum rod was an almost obvious solution. Fortunately the inner diameter of the coil is very close to ¾ inch, which is a standard diameter for metal rods. It would have been nice if the inner diameter were exactly ¾ inch, but it is actually 2 cm (25/32 inch), so it is necessary to wrap the ¾ inch rod with aluminum foil to get a tight fit.

Here is a very rough thermal analysis:

Mass of two 11-inch long 2cm diameter aluminum rods: 476g (see footnote)[4]

Specific heat of aluminum: 0.897 $J \cdot g^{-1} \cdot K^{-1}$

We'll assume that there's a 333W SunPower module driving the IV swinger and the software crashes right at the MPP. We'll also assume that the user doesn't notice this for 60 seconds. The power should be fairly evenly split between the two aluminum rods. We won't account for any cooling of the rods by the ambient air.

333 W = 333 J/s

333 J/s * 60 s = 19980 J

19980 J / (0.897 J/g·K * 476g) = 47 K

[3] If the resistance had been the calculated value of 1.2Ω, we would have needed two sets of three in parallel to increment by 0.4Ω.

[4] Volume = π*(1cm)^2 * 11in * 2.54 cm/in = 88cm^3. Density = 2.7g/cm^3. Mass = Volume * Density = 88 cm^3 * 2.7 g/cm^3 = 238g per rod. 2 rods = 476g

So in this scenario, the aluminum rods would increase in temperature by about 47 K, which is the same as 47˚ C. If the rods start at 40˚ C, they would heat up to 87˚ C. This is less than the boiling point of water, which presumably the coils are designed to be able to handle.

As I said, this is a very rough calculation. Obviously the time assumed (60 seconds) influences the result in direct proportion. But it certainly appears that there should be no thermal issues in all but the most unfortunate of circumstances.

In addition to being heat sinks, the aluminum bars are also very convenient for mechanically holding the heating coils in place.

Note that using a ¾ inch *steel* rod is not a good substitute. Not only does steel have a lower thermal conductivity than aluminum, but it also increases the inductance of the coils by a factor of 100. This is very bad for the relays, as will be described in Section 3.2.6.

3.2.2.5 Mechanical assembly

The construction of the immersion coil portion of the load bank is pretty straightforward and can be seen in the photos in Section 2.4 starting on page 14, but some details are provided here for the benefit of anyone attempting to construct a clone of the IV Swinger.

There are a total of 14 immersion coils, split evenly between the upper and lower aluminum rod. As noted later in this document, one of the coils is not used in the current implementation, so it would be OK to reduce the number to 13, with 6 on one rod and 7 on the other.

3.2.2.5.1 Immersion coil preparation

As shown in Figure 3-5 above, the immersion coils come with a power cord and cigarette lighter plug. The plastic case must be removed and the wires de-soldered from the two ends of the heating element. The heating element is pretty brittle, and cannot withstand being bent. When the recommended 18-gauge wire is used for the connections from the coils to the relays, the process of "cramming" the assembled load bank and relays into the case can cause the heating elements to break off flush with the end of the metal tubing. I found this out the hard way. The solution was to use a short (1.25") copper tubing guard to protect the protruding end of the heating element. Additionally, I shortened both legs of each coil by 3/4". I had two reasons for cutting off 3/4" from the coil legs:

- I needed to do this to salvage the ones with broken heating elements
- The shorter legs made it much easier to fit the assembly into the enclosure (reducing the stress on the leads)

The 3/4" shortening may not be necessary, but adding the 1.25" copper tubing guards is highly recommended. The copper tubing is 1/4" O.D. refrigeration tubing, available from Home Depot ($10/10ft). This fits nicely around the end of the immersion coil tubing (with a layer of shrink wrap between). The idea is to keep the end of the heating element inside the copper tube where it cannot be bent. The 18-gauge wire emerges from the copper tube - so if anything bends, it will be the wire and not the heating element. Additionally, heat shrink tubing is used to protect the solder joint connecting the 18-gauge wire to the heating element. A larger piece of heat shrink tubing is used to hold the copper tubing in place on the leg of the immersion coil.

Figure 3-6 below is a view from the right side of the IV Swinger showing the copper tubing guards in place on the legs of the immersion coils toward the right side of the photo. The black heat shrink tubing that holds the copper tubing in place can be seen.

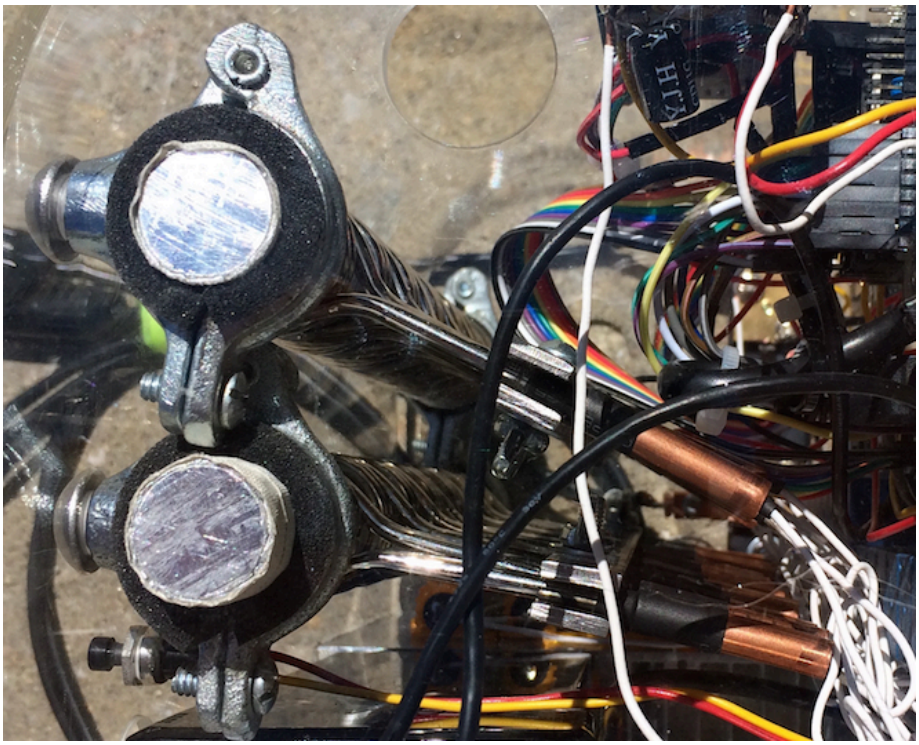


Figure 3-6: Immersion coils with copper tubing guards

Figure 3-7 below is a close-up from the bottom of the IV Swinger. In this photo you can also see the end of the smaller heat shrink tubing around the white connection wire, extending just past the end of the copper tubing. Notice the one farthest to the left; you can see that its wire is sharply bent to the left. The wire and the heat shrink take the stress, with the end of the heating element and the solder joint safely protected inside the copper tubing.

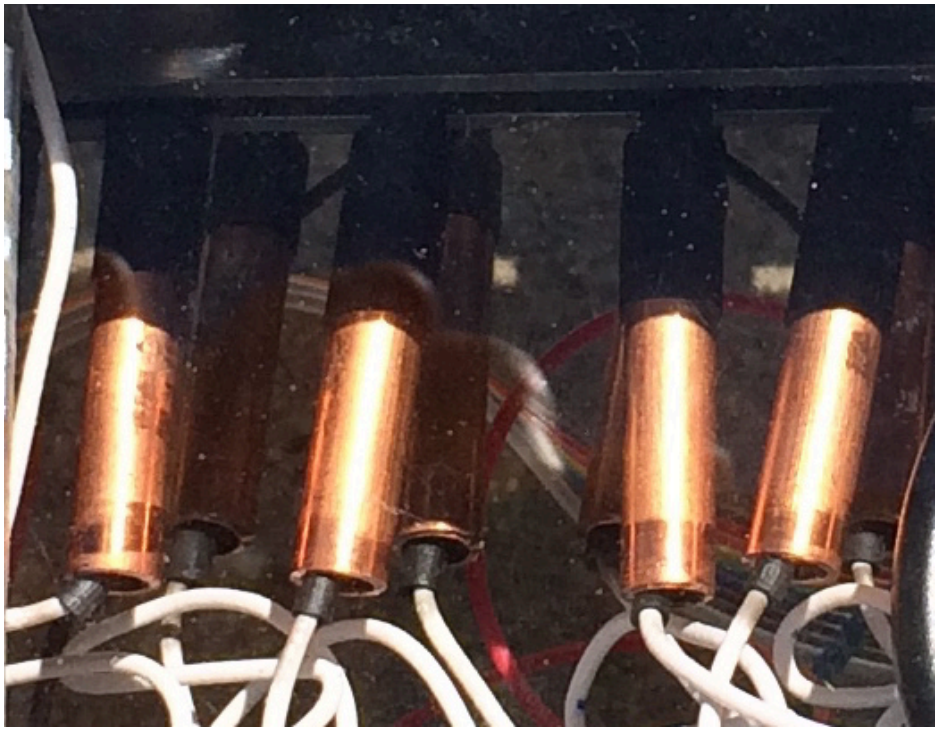


Figure 3-7: Copper tubing heating element guards - close-up

The smaller heat shrink tubing is 2.5mm. The larger heat shrink tubing is 7mm. A small tubing cutter is needed to cut the copper tubing, and also to cut the immersion coil tubing. The Ridgid 32975 ($9.99 on Amazon) would work.

The assembly process is as follows:

1. (Optional?) Cut 3/4" of tubing from each leg of the immersion coils. Cut heating element wire and cloth sleeve 3/4" past new end of tube. Use a razor blade to cut the cloth sleeve so that it extends halfway from the end of the tube to the end of the heating element wire.
2. Cut 1.25” length of copper tubing for each coil tube (28 total). Attempt to “de-burr” the holes with a Phillips screwdriver so they aren’t so sharp.
3. Cut 28 pieces of the 7mm heat shrink tubing – 7/8” long each
4. Cut 28 pieces of the 2.5mm heat shrink tubing – 7/8” long each
5. Cut 28 pieces of the 18-gauge wire – 6.5” long each. Mark them with Sharpie at the following points: 0.25”, 2.25”, 3.25”, and 5.5”
6. Strip both ends of each wire at the marks (0.25” on one end and 1” on the other end)
7. Insert the 1" stripped end of the wire into the cloth sleeve with the heating element until the end of the heating element wire is lined up with the end of the insulation of the 18-gauge wire.
8. Use an alligator clip on the cloth sleeve to hold the wire touching the heating element and solder the wire to the heating element
9. Slide the large heat shrink over one end of a copper tube and thread the wire and the coil tube through that
10. Slide the small heat shrink over the wire and around the end of the coil tube (perhaps 3/16") and use a match to shrink it on
11. Back the copper tubing piece to where it straddles the end of the coil tube, making sure that some of the small heat shrink still extends past the end of the copper tube
12. Adjust the larger heat shrink piece so it is approximately halfway past the end of the copper tube and use a match to shrink it onto both the copper tube and the coil tube.

13. Use the remaining two marks on the wire to bend it into a Z
14. Repeat steps 7-13 for each of the 28 coil legs

3.2.2.5.2 Aluminum rod preparation

As mentioned above, the rods must be wrapped with aluminum foil in order to increase the diameter from 3/4" to 2 cm. I used "Heavy Duty" foil, which is 0.94 mils thick according to Wikipedia. Without going into the math, this comes out to a length of about 47" of foil. However, due to the difficulty of wrapping the foil around the rods completely tightly without any wrinkles, a length of 40" turns out to be best. Since the foil is wider than the 11" rods, it is necessary to trim the ends off after rolling the foil around the rods (which is easier than trying to cut the foil lengthwise first and then keep the roll completely straight). I used a combination of a razor blade and diagonal cutters to trim the foil off. Scotch tape can be used to keep the foil from unwrapping.

3.2.2.5.3 Clamping coils to aluminum rods

The coils can then be slid over the foil-wrapped aluminum rods. The legs have to be spread apart to open the hole enough to get them on. They should be evenly spaced along the length of the rod with space left at the ends for the pipe hangers (see Figure 2-5 on page 15). To clamp them around the rods, the legs have to be squeezed together. For this purpose, four 9" lengths of 1/2" x 1/8" steel flat bar ($3.57 for 36" from Home Depot) are used. Holes are drilled through the bars in the positions between the last two pairs of coil legs on each end. A 3/4" machine screw is passed through each hole and a washer and wing nut are used to squeeze the two bars together. This clamps all of the pairs of coil legs together, which tightens the coils around the foil-wrapped rods. Note that the wing nuts should be on the bottom side of the top clamp and on the top side of the bottom clamp.

The ends of the clamp bars can be seen above in Figure 3-6. Figure 2-4 on page 14 provides a view from the top of the IV Swinger that shows the clamp on the top set of coils.

3.2.2.5.4 Suspension of coil load assemblies in the enclosure

The assemblies with the coils clamped to the aluminum rods must be suspended inside the enclosure so that if/when they get hot, they aren't touching the enclosure or any of the other components. For this purpose, 3/4" zinc-plated split-ring pipe hangers were used. 1/2" long button head bolts were used to attach the pipe hangers, and foam weather strip was used to insulate the pipe hangers from the aluminum rod and to fill the extra space. In the photos you may notice that the ends of the rods are also wrapped with glass cloth electrical tape, but this was overkill.

3.2.2.5.5 Coil load positions

In Figure 3-1 on page 18 and elsewhere later in this document, each of the load coils is referenced by a name that identifies its position in the chain. Figure 3-8 below shows the physical positions of the coil loads in the current IV Swinger design, as seen from the front of the box.

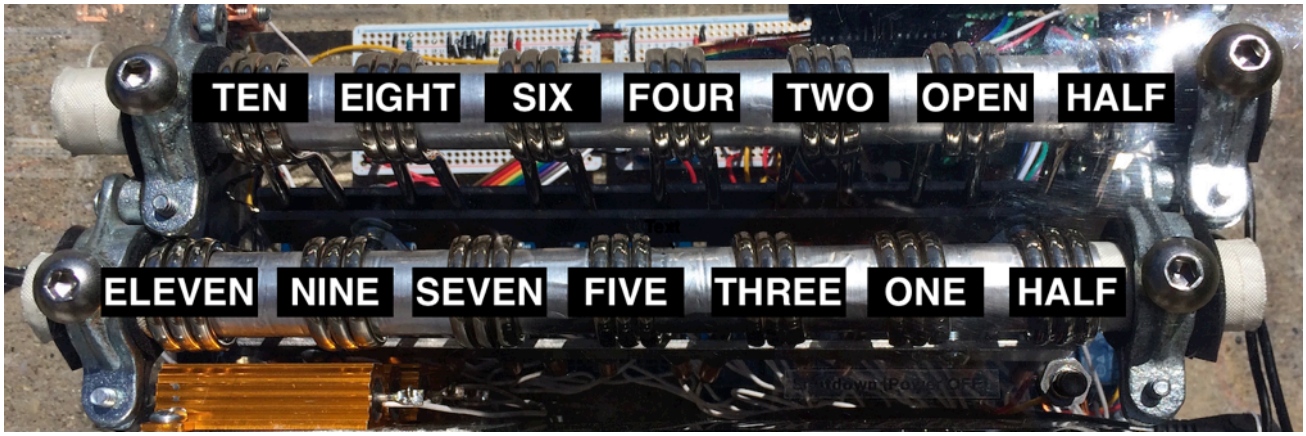


Figure 3-8: Coil load positions

They are essentially in the order that they are connected to the relays, which sit behind them. The chain alternates between the top and bottom rod to distribute the heat as evenly as possible between the two. If thermal distribution had been a larger concern, the HALF, ONE and TWO loads (which get the hottest) could have been put in the middle, with the large numbered loads at both ends. That would have made the wires longer and the connections confusing, however.

As mentioned elsewhere, the OPEN coil is not used, and could be eliminated.

3.2.2.6 Cost

On Amazon, prices for the immersion coils range from $7 to $24 each. But due to the miracles of the modern global economy, they can be purchased on EBay directly from Hong Kong for under $3 each (search for "12v immersion"). As of this writing, I see them listed for as low as $2.59 each with free shipping. The IV Swinger uses 14 of them, so the total cost is currently about $36 for all of them. Each one comes with a cord and cigarette-lighter-style plug, none of which is used. If the IV Swinger were ever to be produced in larger quantities, it would be interesting to find out if it would be possible to purchase the heating element by itself.

The 11-inch 6061 aluminum rods (3/4" diameter) are $14.12 for a quantity of 5 on EBay.

The costs of the remaining items can be found in Section 11 on page 112.

3.2.3 Power resistors

As described in Section 3.1.2, a higher resistance load increment is needed for the end of the IV curve. From the range and resolution exploration (Figure 3-4), we want the last three resistance increments to be on the order of 3Ω, 6Ω, and 12Ω. It would take 4, 8, and 15 immersion heating coils in series to construct these three increments, which is more than we need for all the other data points combined. These load elements don't have to be capable of dissipating as much power as the others because the current is low at the tail of the IV curve.

Again, a consumer product is desirable for cost reasons. A good solution was found: 50W 6Ω power resistors that are sold for automotive use. They are used with LED turn signal lamps to add load to the flasher circuit (which is designed for incandescent lamps) so it works properly.

3.2.3.1 Resistance

The rated resistance is 6Ω. For the 3Ω load, two are connected in parallel. For the 12Ω load, two are connected in series. A total of five 6Ω resistors are needed.

3.2.3.2 Power

The rated power of each resistor is 50W. The 3Ω load can handle 100W because it is two 50W resistors in parallel.

3.2.3.3 Current

Based on the power rating of 50W and resistance of 6Ω, the rated current is:

$$I_{rated} = \sqrt{\frac{P}{R}} = \sqrt{\frac{50W}{6\Omega}} = 2.9A$$

The 3Ω load can handle twice this, or 5.8A. This is a lot less than the I_{SC} of 10A, but these resistors are only used when the current is much lower than I_{SC} (or when I_{SC} is much lower than 10A).

3.2.3.4 Thermal

These power resistors must be fastened to something metal in order to dissipate the rated 50W. For this an aluminum heat sink was chosen. Again, a consumer item was cheaper than an industrial heat sink. The chosen heat sink is intended for cooling LED lighting in aquariums. A thermal analysis has not been performed.

3.2.3.5 Mechanical assembly

Figure 3-9 below shows the power resistor load assembly. The 240x76x21mm heat sink is cut into two 120mm long halves that are placed side to side. This just happens to fit perfectly in the acrylic case that was chosen. Self-tapping screws are used to fasten the resistors to the heat sinks. The middle one straddles the two heat sink halves and should be screwed on first followed by the other four. The top two resistors are the 3Ω load in the TWELVE position in the chain and are connected together in parallel with the 18-gauge wire soldered as shown. The middle resistor is the 6Ω load in the THIRTEEN position in the load chain and is not connected to any of the others. The bottom two are the 12Ω load in the FOURTEEN position and are wired together in series as shown. Visible in Figure 2-5 on page 15 and Figure 2-8 on page 16 is glass cloth electrical tape on the sides of the heat sinks (top and bottom of the assembly as shown in Figure 3-9). The purpose of this was to protect the acrylic from the heat sinks. It is probably not necessary.

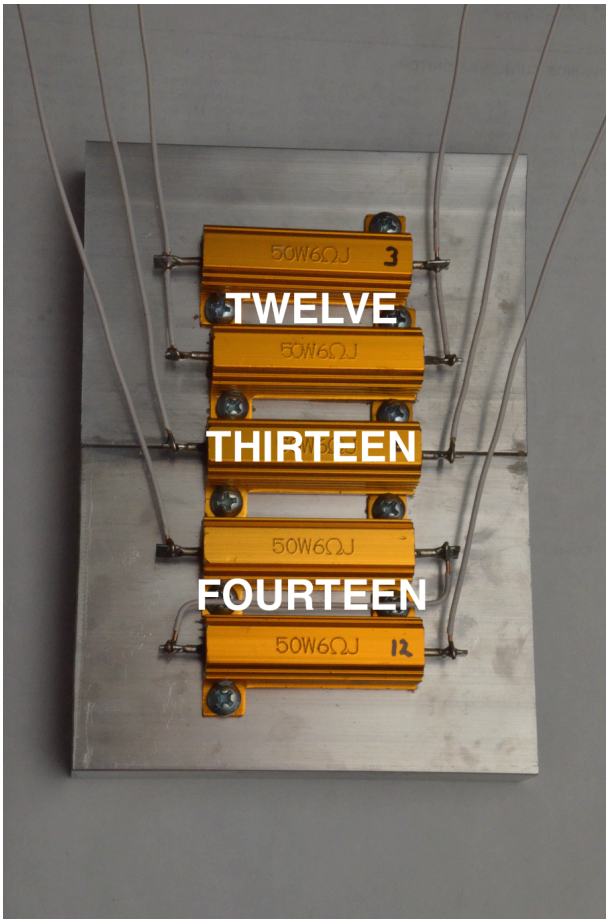


Figure 3-9: Power resistor load assembly

3.2.3.6 Cost

On Amazon the power resistors are about $2 each when purchased in quantities of 2 or 4 (we need 5). The heat sink is under $9.

3.2.4 Relays

Given the decision to use discrete load elements, an electronically controlled switching mechanism is needed. The least expensive option for this is a relay. A single-pole double-throw (SPDT) relay is a device that uses an electromagnet to switch a common (C) terminal from being connected to a "normally closed" (NC) terminal to being connected to a "normally open" (NO) terminal as shown in Figure 3-10 below.

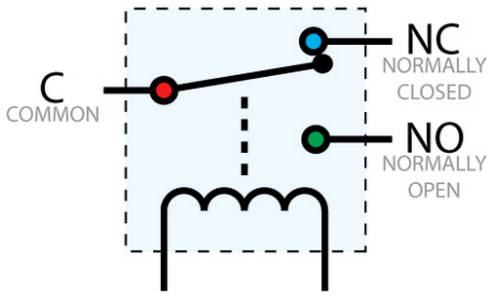


Figure 3-10: SPDT relay schematic drawing

Physically, the inside of a relay is shown in Figure 3-11 below.

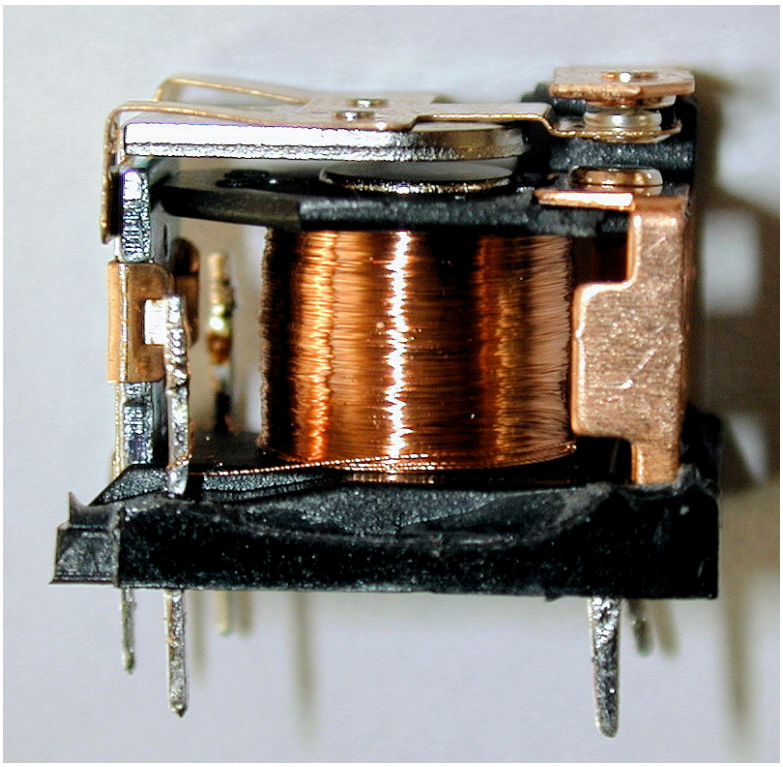


Figure 3-11: Inside a physical relay

When there is no current flowing through the coil, the electromagnet is “off” and the spring metal holds the middle (C) contact up against the NC contact. When current flows through the coil, the electromagnet is “on”, and it pulls the C contact down to the NO contact. When a relay switches there is a very audible “click”.

One relay is used per load element. When the relay is not activated, the current in the load circuit passes directly from the C terminal to the NC terminal and bypasses the load element. When the relay is activated, the current in the load circuit passes from the C terminal to the NO terminal and through the load element.

Modules with multiple relays are readily available and very inexpensive. In addition to the relays themselves, the modules have other necessary supporting components all mounted on a PCB that allows for easy mounting. Figure 3-12 below is a photograph of an 8-relay module of the type used in the IV Swinger along with a circuit diagram for one relay.

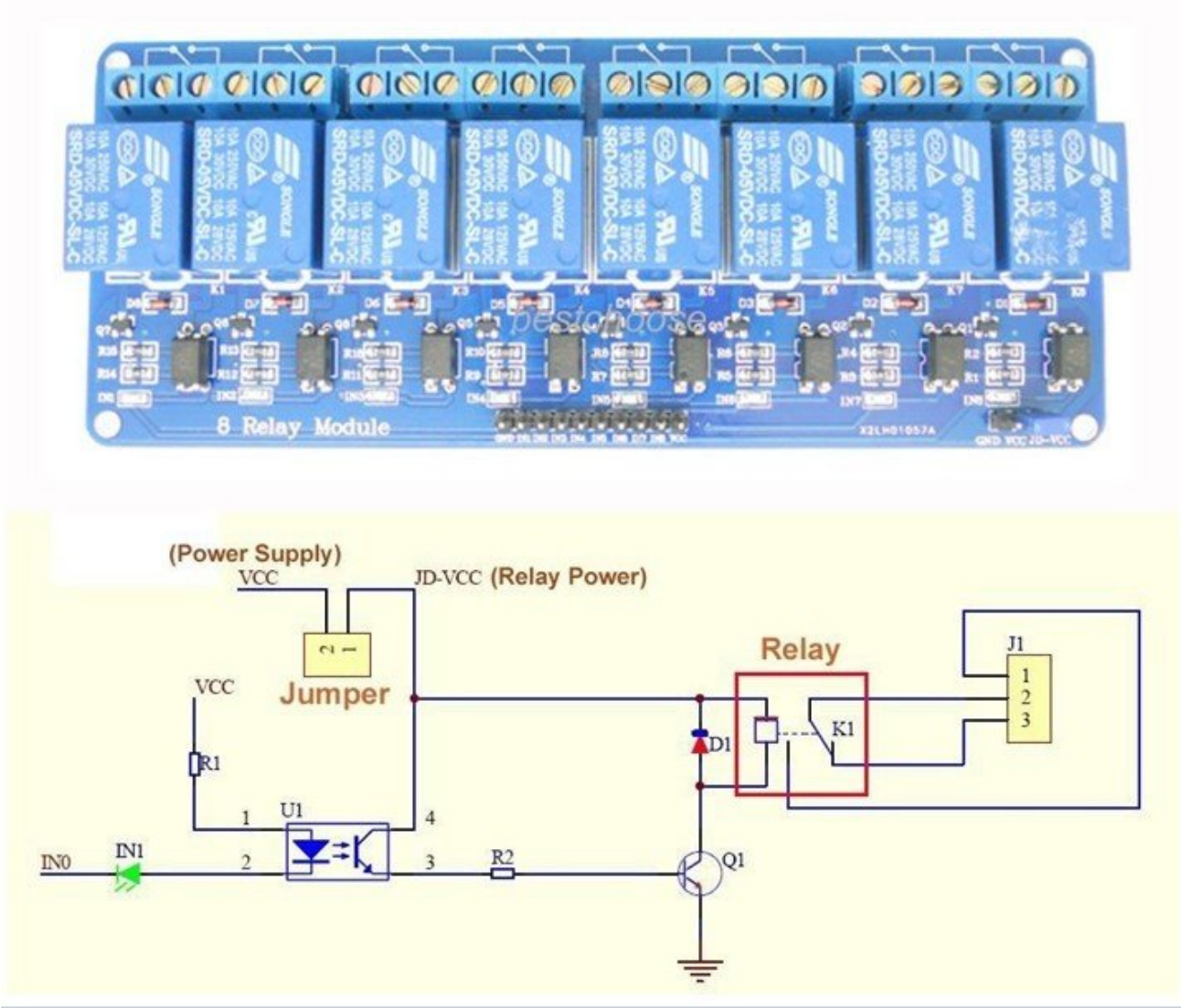


Figure 3-12: 8-relay module

The relays themselves are the blue boxes in the middle. At the top of the photo are the terminal blocks where the connections are made. The middle connection on each is the common (C) terminal. The one on the left is the Normally Closed (NC) terminal, and the one on the right is the Normally Open (NO) terminal. These terminal blocks have holes where the wires are inserted and tiny screws to hold them in place. Removing the terminal blocks and soldering the wires directly to the board increases reliability. Immediately below each relay is a red LED that lights up when the relay is active (shown in green on the circuit diagram). The other components on the board are optoisolators, transistors, resistors and diodes, which are all shown in the circuit diagram. The 10 pins in the center at the bottom are GND, IN1:IN8, and VCC. The GND pin is connected to the common ground used by all components in the IV Swinger (actually tied to the negative side of the PV). VCC is +5V, shared with the +5V used to power the Raspberry Pi. The IN1:IN8 pins are controlled by the Raspberry Pi to determine whether the associated relay is activated or deactivated. These pins are "active low", which means that a low (zero) voltage activates the relay and a high (+5V) voltage deactivates the relay[5]. In the lower right corner of the board are three more pins: GND, VCC, and JD-VCC. GND and VCC are connected by PCB traces to the GND and VCC pins in the group of 10 in the middle. The JD-VCC pin is shown with a jumper connecting it to

[5] Exercise for the reader: look at the circuit diagram and understand why this is the case.

the VCC pin next to it. This jumper must be removed. As shown in the circuit diagram, JD-VCC is the power for the electromagnet coils in each relay (I have no idea what "JD" means). It is also +5V, so the jumper is provided for cases where it is acceptable to use the same power source for VCC and JD-VCC. We want to keep them as separate as possible. Although they are both driven by the same battery pack, they are driven by different USB ports on the battery pack. The reason for this is to isolate the Raspberry Pi as much as possible from the power transients that result from the relatively large gulps of current that the relay coils consume. It is also the case that the current requirements of the Raspberry Pi and the current requirements of the relays together are more than any one of the battery pack outputs can supply. Tying VCC and JD-VCC together with the jumper could result in the Raspberry Pi crashing due to noisy power and/or insufficient current.

3.2.4.1 Cost

A brand name 8-channel SainSmart relay module is around $18 on Amazon, which already seems like a great deal. But it is easy to find no-name clones for around $7 each on EBay that appear to be of comparable quality. They all use Songle brand relays (SRD-05VDC-SL-C) and are all made in China, regardless of how much you pay. The IV Swinger uses two 8-channel relay modules.

3.2.4.2 Current and Voltage Limitations

The important specifications are those of the Songle SRD-05VDC-SL-C relay. The most important of those is how much load current it can handle. The case has some values printed on it:

It's not clear what the difference is between the ones on the left and the ones on the right, but the current is 10A in both cases. The modules are always advertised as handling up to 10A. Since 10A is the maximum I_{SC} we have chosen, this sounds good.

The actual Songle relay specification (which can be found with a Google search) has the following table, however:

7. CONTACT RATING

Item \ Type	SRD	
	FORM C	FORM A
Contact Capacity Resistive Load (cosΦ=1)	7A 28VDC 10A 125VAC 7A 240VAC	10A 28VDC 10A 240VAC
Inductive Load (cosΦ=0.4 L/R=7msec)	3A 120VAC 3A 28VDC	5A 120VAC 5A 28VDC
Max. Allowable Voltage	250VAC/110VDC	250VAC/110VDC
Max. Allowable Power Force	800VAC/240W	1200VA/300W
Contact Material	AgCdO	AgCdO

The relays used on the module are "Form C" (as indicated by the C at the end of the part number). So it appears that the maximum current is 7A for a resistive load and only 3A for an inductive load, and the markings on the case are misleading. Our loads are somewhat inductive, so that places the rated current somewhere between 3A and 7A. What does it mean if we exceed this value (which we will)? It means the relay contacts will wear out before the rated number of cycles, which is a minimum of 10,000 if the constraints are met. If we were drastically exceeding the rated current, overheating (or even fire) would be a concern, but given that we're staying below the value printed on the case this is highly unlikely.

It takes some hand waving to feel comfortable with this, but here are some mitigating factors:

- 10A is the maximum I_{SC}. Most panels have a lower rated I_{SC} than 10A.
- Most measurements will be taken when the insolation is less than full sun, so the current will be lower than the rated I_{SC}
- 10,000 cycles is the minimum lifetime so typical should be higher than this
- 1000 cycles would probably be enough for the lifetime of an IV Swinger used in an academic lab
- "Snubber" circuits are used to suppress arcing between the relay contacts; in theory this should prolong their life (this will be discussed in section 3.2.6)
- If the relays do start to fail, it's an inexpensive repair

As for voltage, it is unclear what the significance is of the 28V value. The table says that the maximum allowable voltage is 110 VDC, which is higher than our maximum V_{OC} value of 80V.

The bottom line is that these relays are very close to meeting the requirements and the alternatives are so much more expensive that there's really no other reasonable choice. There's no question, however, that the relays are the IV Swinger's Achilles' heel; it is highly likely that a relay will be the first thing to fail. If IV Swinger were a commercial product with a warranty, a lot more effort would be required to understand and quantify the impact of exceeding the relay specifications.

3.2.4.3 Current/Power consumption

The relays require a negligible amount of power when they are in the inactive state, but each one requires about half a watt of power in the active state. This is the power required to energize the electromagnet coil.

- Coil current: 89.3mA

- Coil resistance: 55Ω ±10%
- Coil power: 0.39W – 0.48W

To energize all 16 relays simultaneously requires 16 * 0.0893A = **1.43A** and up to 7.7W of power. This information is needed when we look at the battery pack requirements.

3.2.4.4 Relay Terminal Connections

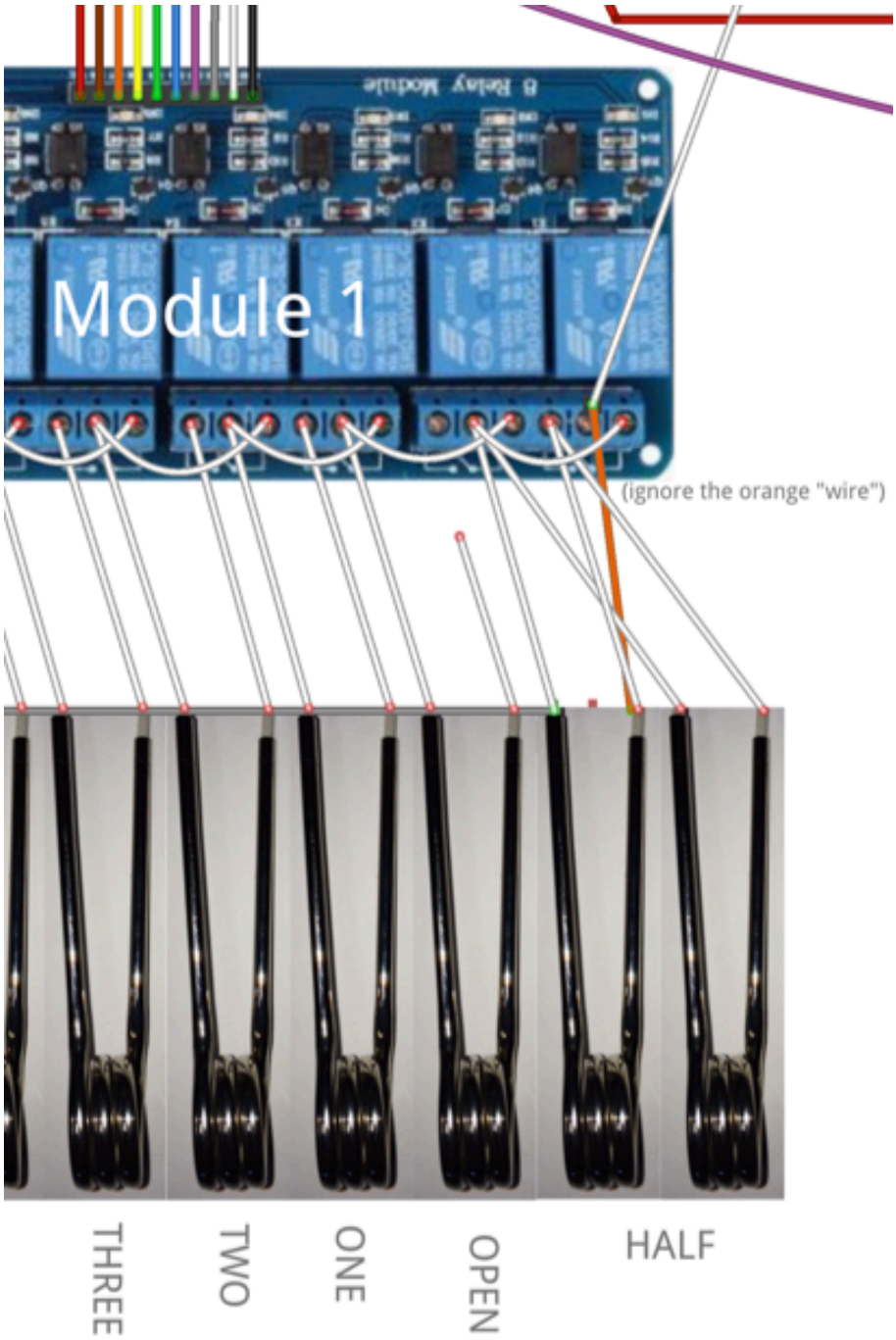
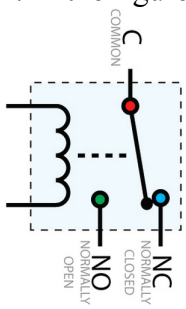


Figure 3-13: Relay connections to loads

Figure 3-13 above is a zoomed in view from Figure 2-2 that shows the connections made at the relay terminals on the head end of the load chain[6]. In the figure, the relays are oriented like this:



The white wire coming from the upper right comes from the DPST switch (discussed in the next section). This is where the current from the PV panel enters the load chain (see Figure 3-1 on page 18 for a high level view of the current flow, which is from RIGHT to LEFT in this picture). This wire is attached to the middle (C) terminal of the first relay.

The right (NC) terminal of every relay is connected to its neighbor's middle (C) terminal. This is the path that bypasses the associated load. Since the C terminal is connected to the NC terminal when the relay is not activated (coil not energized), the default is for all loads to be bypassed. The current flow looks as shown in Figure 3-14 when none of the relays is activated.

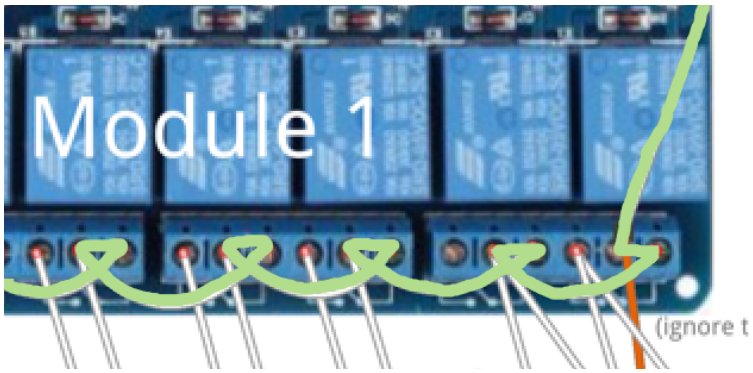


Figure 3-14: Current flow with loads bypassed

The first two coils are wired in parallel to form the 0.4Ω "HALF" load and are connected between the NO terminal of the first relay and the C terminal of the second relay. When the first relay is activated the current flows through both of the HALF load coils as shown in Figure 3-15 below.

[6] In case you are curious, the orange "wire" is used so that in the Fritzing schematic view the whole load bank can be represented as a variable resistor.

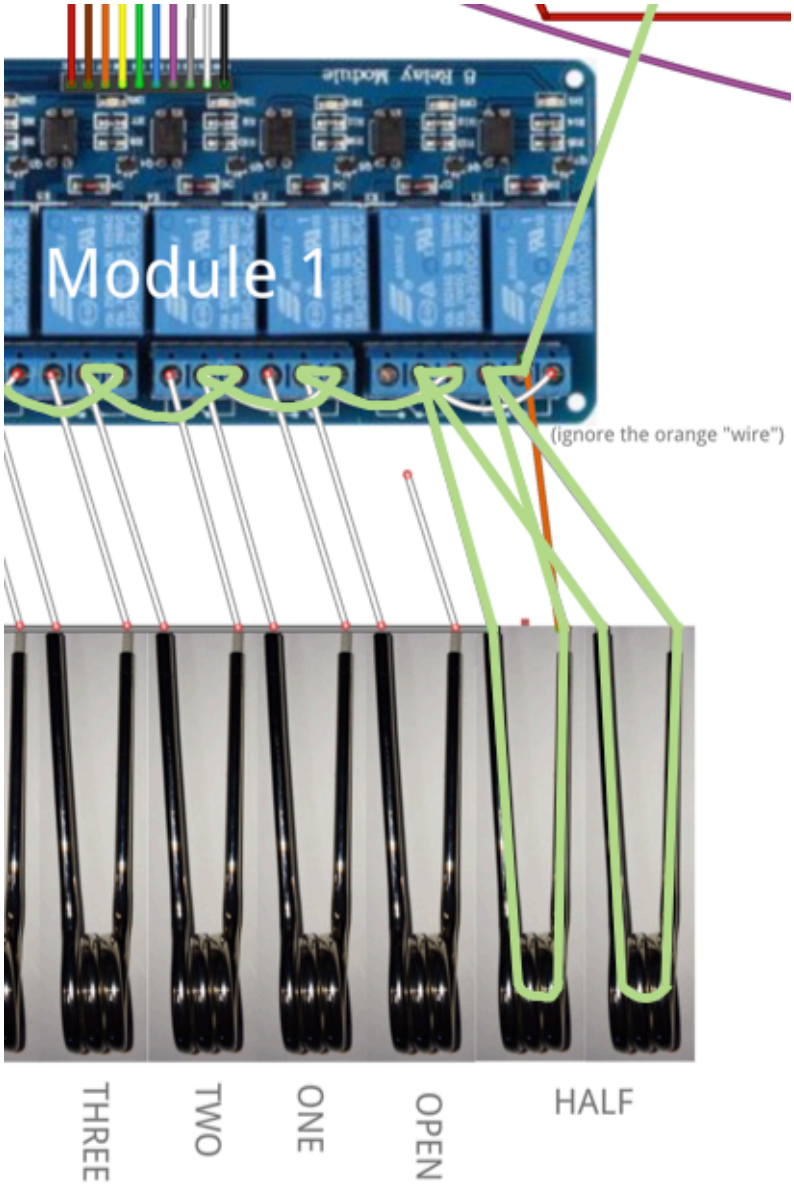


Figure 3-15: Current flow with HALF load selected

The remaining loads are connected the same way. The NO terminal of each relay is connected to one side of its load and the other side of that load is connected to the C terminal of its downstream neighbor to its left.

Notice that the coil between the HALF load and the ONE load is labeled OPEN and its upstream leg is not connected to the second relay. This was a late design change made to allow the software to open the circuit without user intervention. The reason for this will be discussed later in the document. Obviously that coil serves no purpose with this configuration and could be eliminated. It also would be more elegant to use the relay at the other end of the chain for the OPEN “load” (but that would also necessitate a minor software change).

NOTE: The figures/photos in this section show the screw terminal blocks. For reliability it is recommended that these be removed and the wires be soldered to the circuit board. The bypass wires can be placed on the back side of the board as seen in Figure 2-8 on page 16 and in Figure 3-18 on page 47.

Those photographs also show the snubber circuits discussed in section 3.2.6.2, but not shown in the figures above.

3.2.5 DPST switch

The double-pole single-throw (DPST) switch is on the top of the IV Swinger and looks like the photos in Figure 3-16 below.

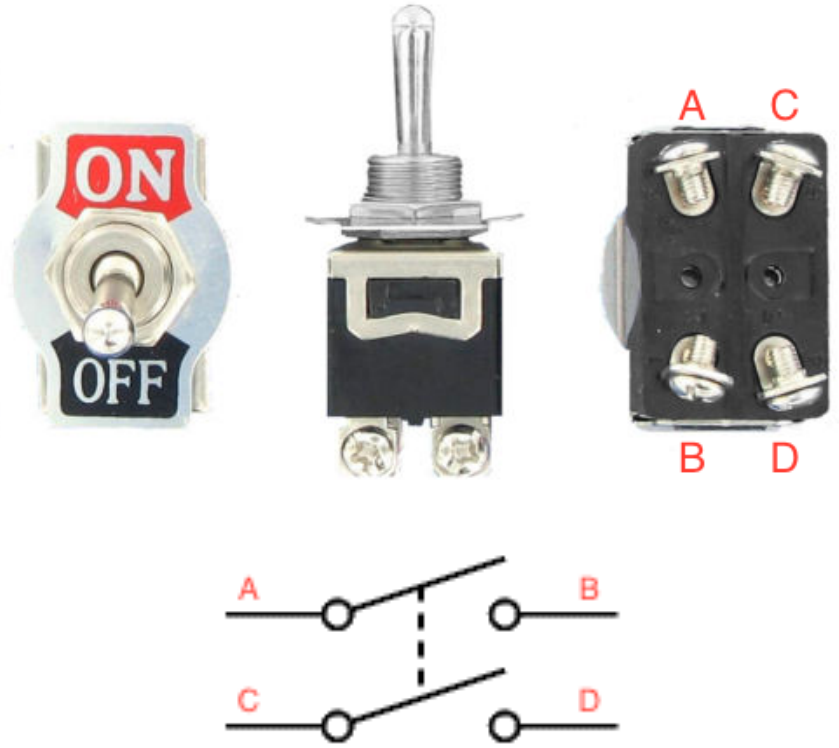


Figure 3-16: DPST switch

The symbolic representation of a DPST is also shown in Figure 3-16. The single toggle lever controls two circuits, and they both open and close in tandem as represented by the dotted line. The rightmost photo shows all four of the terminals on the bottom of the switch. Note that the switch connects/disconnects terminal A to/from terminal B and connects/disconnects terminal C to/from terminal D. Turning the switch from OFF to ON makes (closes) the connections and turning it from ON to OFF breaks (opens) the connections.

3.2.5.1 Connections

The A-B side of the DPST is used to open or close the load circuit itself. The C-D side is connected to a GPIO pin on the Raspberry Pi so the software can detect if the load circuit is open or closed. This will be discussed in more detail in Section 5.7.

3.2.5.2 Cost

The chosen DPST switch pictured in Figure 3-16 is sold in quantities of 3 for $4.49 on EBay ($1.50 each).

3.2.5.3 Ratings

This switch is rated for 20A @ 125VAC and 15A @ 250VAC. It is not rated for DC at all! Switches rated for DC are more expensive and harder to find.

The issue with switching DC is the arcing that occurs when the switch is opened while current is flowing (closing is not a problem). This is discussed in section 3.2.6 below. In order to increase the longevity of the DPST switch, a snubber circuit is used. However, the design now uses one of the 16 relays to break the circuit at the end of the test so that when the DPST is opened, current is no longer flowing. Unless the operator opens (turns OFF) the switch before being prompted, the DPST should never experience arcing. Even if that occurs occasionally it is highly doubtful that the switch will wear out before the relays.

3.2.6 Arc reduction

As already mentioned, switching DC is very tough on the contacts of the switch/relay. This is because arcing occurs when the switch is opened and that is due to the physics of inductors. All loads have some amount of inductance, especially those that include coils.

The voltage across an inductor is given by the equation:

$$V = L\frac{di}{dt}$$

L is the inductance and *di/dt* is the rate of change of the current (A/s). When a switch closes, the current starts at 0 and increases as the inductor's magnetic field builds up (storing energy). *di/dt* starts out as high as the power supply can provide but then starts dropping to zero (i.e. current increasing more and more slowly) at which point the voltage across the inductor is zero. When a switch opens, however, there's a problem. The current instantaneously drops from some positive value to zero. This means *di/dt* is infinitely negative and the voltage across the inductor is infinitely negative! Of course this cannot be. Even though there is air between the switch contacts and air is normally an insulator, the magnitude of the voltage is so high that the air is ionized, i.e. changed to plasma, which is highly conductive. On a larger scale this is called lightning. On a small scale it is called arcing. For a short time there is a miniature lightning bolt between the switch contacts. This is not good for the contacts. Arcing must be avoided if possible and minimized otherwise.

Something important to note is what happens when a SPDT relay switches. Conceptually we think of the current instantaneously diverting from one path to the other path. But there is actually a short time when the common (C) contact is "on its way" from the NC contact to the NO contact (or vice versa) and is not physically touching either one. The flight time of the C contact is on the order of 5-10ms, and this is more than enough time for an arc to do its damage (in fact the arc will almost certainly have stopped before the C contact reaches the other side).

Figure 3-17 below is a photo of a relay contact damaged by arcing. I took this photo after opening up a relay that had failed in the IV Swinger before making design changes to reduce arcing; it's a true

problem that must be dealt with. The contact started out as a nice smooth rounded metal bump, but the arcing created a large pit. With this damage the contact no longer reliably conducts current in the closed position.

Figure 3-17: Relay contact damaged by arcing

3.2.6.1 Minimizing inductance

The strength of arcing is directly proportional to the inductance (L) of the load (since *V=L*di/dt*). So minimizing the inductance of the load is one way to reduce arcing. A coil of wire is an inductor and the immersion heaters that are used for the IV Swinger are coils of wire. The inductance of a coil of wire is given by the following equation:

Equation 2: Inductance of a coil

$$L = \frac{\mu N^2 A}{l}$$

μ: magnetic permeability of the core
N: number of turns
A: area
l: length

The only one that we have control over is μ. The magnetic permeability of air and anything else that a magnet does not stick to are all close to the same as a vacuum, which is the lowest value possible. Ferrous materials (those containing iron) have a high magnetic permeability. Carbon steel has a μ value of about 100 times that of a vacuum. The photo in Figure 3-17 is the damaged contact from one of the relays when I had been using a carbon steel rod as the heat sink for the immersion coils. Big mistake!

That was increasing the inductance by a factor of 100. Aluminum has a μ value close to the minimum. It also has better thermal conductivity than steel, and is lighter and cheaper!

3.2.6.2 Snubbers

Another way to reduce the effect of arcing is to attempt to suppress it. A so-called "snubber" circuit is simply a resistor and capacitor in series placed across the switch contacts. A capacitor looks like an open circuit to DC, so in the steady state (switch open or switch closed) it has no effect. But a capacitor looks like a short circuit at the moment that the switch is opening or closing. This provides an alternate path for the current so it doesn't have to jump between the contacts. In reality, no electrons are actually going "through" the capacitor; they are just charging the capacitor up. If the capacitor is small, it will charge up too soon to suppress the arcing.

I'll pause here to note that the first result if you do a Google search for "snubber arc suppression" is a company called Arc Suppression Technologies (http://www.arcsuppressiontechnologies.com). They claim that snubbers don't work. They provide some empirical evidence of this, but it's not clear that the snubber capacitors in their experiments were not simply far too small.

Inductors and capacitors both store energy. Inductors store energy in their magnetic field. Capacitors store energy in their electric field. In theory, the snubber capacitor needs to be big enough to absorb all of the energy stored in the inductive load when the switch is opened.

The energy stored by an inductor is given by:

$$\frac{1}{2}LI^2$$

The energy stored by a capacitor is given by:

$$\frac{1}{2}CV^2$$

Since we want these to be equal, the required capacitance is given by Equation 3 below.

Equation 3: Snubber capacitance

$$C = \frac{LI^2}{V^2}$$

First we need to estimate the inductance of the load. Using Equation 2 from above we can estimate the inductance of one of the immersion coils with its aluminum core.

Equation 4: Inductance of one immersion coil

- $\mu = 1.257 \times 10^{-6}$ H/m (aluminum)
- N = 3
- $A = \pi \cdot r^2 = \pi \cdot (0.012m)^2 = 4.524 \times 10^{-4}\ m^2$
- $l = 1.3 \times 10^{-2} m$

$$L = \frac{\mu N^2 A}{l} = \frac{\left(1.257 \cdot 10^{-6}\ \frac{H}{m}\right) \cdot 3^2 \cdot (4.524 \cdot 10^{-4})\ m^2}{1.3 \cdot 10^{-2} m} = 0.394 \cdot 10^{-6} H = 0.394\ \mu H$$

The worst case is when all 14 immersion coils are selected, so the total inductance would be 14x0.394µH, which is **5.5µH** (series inductances add). Note that we're not making any estimate for the 6Ω power resistors but that is because they are only used at low current values.

Now to complete the calculation of capacitance in Equation 3 we need values for V and I. For I we should assume our maximum I_{SC} current of 10A. V is a bit trickier. It is the voltage across the capacitor when it has charged up following the opening of the switch. This should be V_{OC}. In this case since V is on the bottom of Equation 3 we want to choose a worst-case small value of V_{OC}. For lack of a better guess, we'll choose 10V. At least that makes the math work out nicely since $I^2=V^2$. That means C=L. L is 5.5µH so C is 5.5µF. There are so many assumptions in this calculation that it could be wildly off. It is most likely too high. A standard capacitance value is **4.7µF**, so that was chosen for the snubbers.

A snubber also includes a resistor in series with the capacitor. The purpose of the resistor is to limit the discharge current from the capacitor when the switch is closed again since a high current is bad for the switch. The desired resistance is also difficult to determine. On one hand a smaller resistance is better because it allows the capacitor to charge up more quickly when the switch opens. But too small a resistance won't adequately limit the discharge current. We'll assume that the relay contacts can handle 50A for the very short time that it takes to discharge the capacitor. If the voltage across the resistor is 80V (maximum V_{OC}), R=V/I=80/50≈**2Ω**.

We only need snubbers across the switch/relay contacts that open when there is current flowing. Each relay has two sides: the C-NC side and the C-NO side. Having a snubber on each side would provide protection for switching in either direction. That would be the most flexible because it would allow the software to sequence the adding and removing of load elements in any order. For example, it could start with all loads selected, and then de-select them one by one (i.e. swing the IV curve from the V_{OC} end to the I_{SC} end instead of the other way). But there is little value to such flexibility, so it's not worth the effort and expense to do this. Most of the relays only need a snubber across the C-NC side because the only switching under load happens when the relay switches from the NC contact to the NO contact when its load is added to the chain. The one exception is the relay that controls the "half" increment load. That one needs two snubbers because it is both added and removed from the chain during the sequencing.

Figure 3-18 below shows the snubbers added to the back of the relay module. Note the dual snubbers on the far left – that is the one that controls the half load.

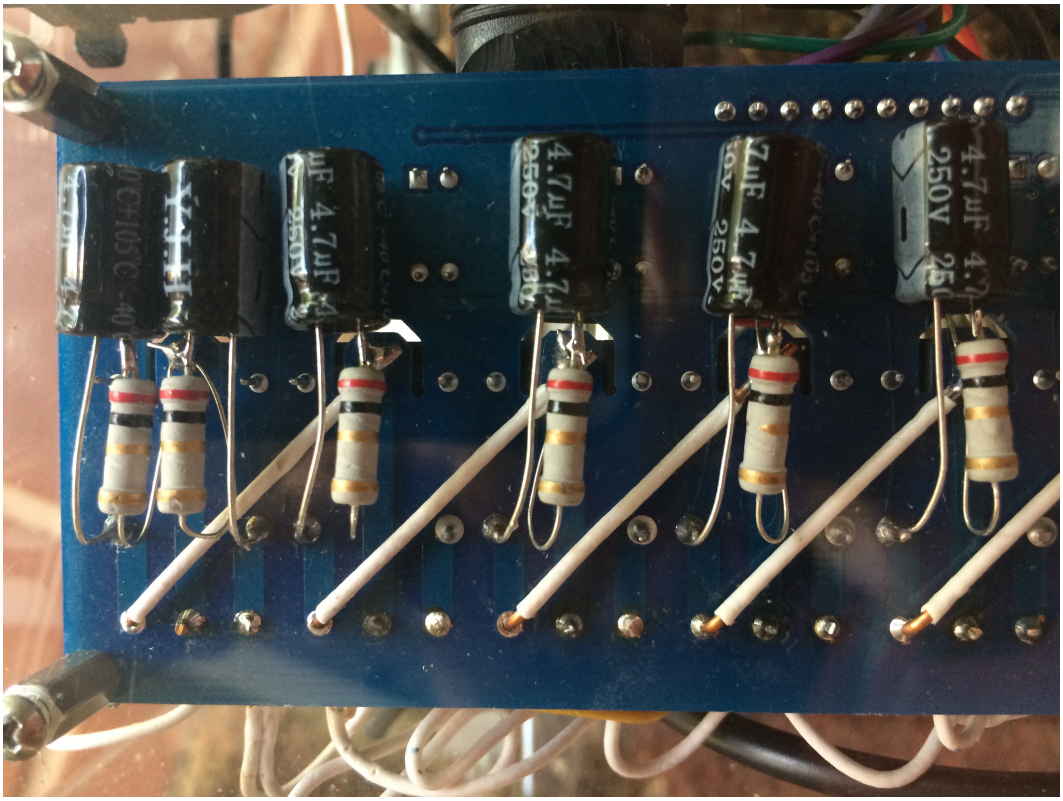


Figure 3-18: Relay snubbers

3.2.6.3 Software role in arc reduction

Even with the aluminum bars and snubbers, there will still be some amount of arcing and it will eventually cause the relays to fail. Most of the relays switch exactly once per IV curve trace. The exception again is the relay that controls the "half" increment load. In order to generate a data point between each of the "full" load values, it has to alternate being on and off. There are 12 full load relays, so this means it would get about 12x the use of any of the others. It will certainly be the first to fail. Indeed, that is the one that sacrificed its life for Figure 3-17.

One idea to eliminate this discrepancy was to have the software swing the IV curve in two passes. The first pass has the half load bypassed and the second pass has it selected. With this algorithm, each of the full load relays switches twice and the half load relay switches once. The problem with this (found empirically) is that the results are ugly! The insolation often changes enough in the ~1 second between passes that the flat top of the curve looks jagged (no wonder swinging IV curves manually doesn't produce great results).

A better idea was to use the half load only where it is needed, which is where the curve is inflecting. There is no value to having more points on the linear parts of the curve. The software increments the load in full steps, but checks the slope between each pair of points on the fly. If the slope of the current pair is very close to the slope of the previous pair, then it continues with another full step. But if there is enough of an inflection, it "backs up" and takes a half step measurement. A typical IV curve only requires two half-step measurements around its knee to look just as good as if there had been a half step between all full steps. Shading cases have multiple inflections (see Figure 13-1 on page117), so they use the half step more than twice. On the average, the half step relay probably switches three times per IV curve. So it will still wear out 3x as fast as the others, but that is a lot better than 12x.

4 Meters

There are two meters in the IV Swinger: a voltmeter and an ammeter. This section describes the requirements and design of the meters.

4.1 Meter requirements

4.1.1 Don't affect what is being measured

As is the case for all instrumentation, it is important that the presence of the meters has a negligible effect on what they are measuring (no "Heisenberg" effect). In this case that means the meters should not change the voltage or the current that they are measuring. This may sound obvious, but consider that the whole purpose of tracing an IV curve is to measure the effects of loads on the circuit. Therefore the meters must have a negligible contribution to the load.

4.1.2 Software readability

The values read from the meters are not useful if they are only presented on a human-readable display. Software running on the computer must be able to read and record the values.

4.1.3 Accuracy and Precision

The accuracy of the measured values is not critical since the main purpose of the IV Swinger is as an educational tool. It won't affect the learning process if the values are off by 5% or even 10%. Precision is more important. i.e. it is OK if the values are inaccurate by 5% or 10% as long as all measurements have the same offset from reality. Specifically, it is important that values measured on one run can be compared with values measured on a different run with fairly high precision. Even more important are the relative values of measurements taken on the same run. If the measured values don't have adequate significant digits, the graph will have stair steps rather than smooth sloped lines. And if the values do have adequate significant digits but the measurements don't have that actual amount of precision, the graph will be "noisy".

Although accuracy isn't critical, it is desirable – for credibility if nothing else. Calibrating the meters to a known reference is desirable.

4.1.4 Speed

Conditions can change significantly in a very short period of time so we want to "swing" out the IV curve as quickly as possible. The amount of time it takes is dependent on how long it takes to switch loads and how long it takes to take the measurements. The load switching is likely to overshadow the measurement time, but we still need to be cognizant of time in the meter design.

4.1.5 Robustness

The meter electronics need to be tolerant of human error. In particular there must be protection against reverse connection of the PV cables, and there must be protection against connection of PV panels that exceed the specified maximum voltage and current.

4.2 Meter Design

4.2.1 Analog-to-Digital Converter (ADC)

An Analog-to-Digital Converter (ADC) translates a measured voltage level to a digital value that can be read by software. This is just what we need. Even though the ammeter measures current, it does so by measuring the voltage across a resistor and applying Ohm's Law.

The IV Swinger uses the ADS1115 16-bit 4-channel ADC from Texas Instruments. It comes mounted on a small PCB sold by Adafruit as shown in Figure 4-1 below. It costs $14.95.

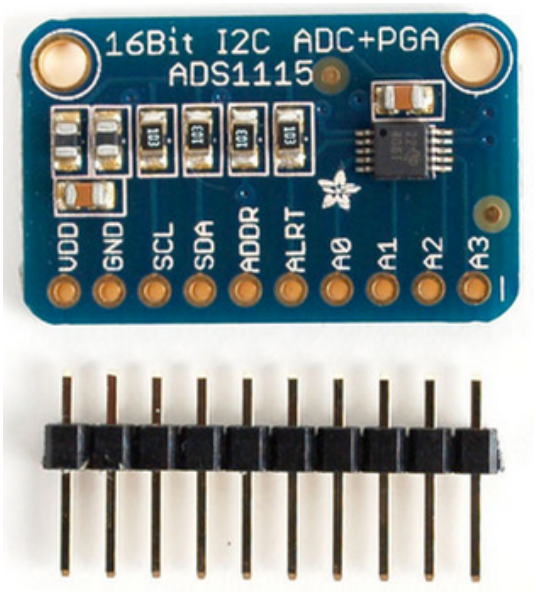


Figure 4-1: ADS1115 board

We have two voltages that we need to measure – one for the voltmeter and one for the ammeter. So 4 channels is twice as many as we need. However, there is a "differential" mode where instead of the voltage being measured relative to ground, it is measured between two of the channels. We use this feature, so two channels are used for the voltmeter and two are used for the ammeter.

The computer interface of the ADS1115 is the widely used I^2C bus. This is a two-wire serial bus that is also used by other components on the IV Swinger (MCP23017, and real-time clock). I^2C is a pretty slow communication channel, but plenty fast for controlling and reading the ADC.

The digital values that are generated by the ADS1115 are 16 bits. The upper bit is a sign bit, allowing it to measure both positive and negative values. This is useful so we can detect the case where the PV cables are connected backwards. The remaining 15 bits allow a resolution of 32,768 (2^{15}) increments across the voltage range.

The maximum allowable voltage on the inputs is the supply voltage (+5V in our case). We'll discuss in sections 4.2.2 and 4.2.3 below what circuitry is needed to scale the voltages we're measuring to be in this range and how the inputs are protected. The ADS1115 also has a Programmable Gain Amplifier (PGA) that helps in this regard by amplifying the input signal so that the full resolution is available over a smaller range than the full 5V. When a gain of 1 is programmed, the range is ±4.096V, i.e. the maximum negative 16-bit digital value represents -4.096V and the maximum positive digital value

represents +4.096V. In that case the resolution increment is 4.096V/32768 = 125 µV. But if you know that you're measuring values that are all small in magnitude, the PGA gain can be increased to get the benefit of the full resolution over the smaller range of interest. For example when the gain is programmed to 8, the range is ±0.512V, i.e. the maximum negative digital value represents -0.512V and the maximum positive digital value represents +0.512V. In that case, the resolution increment is 0.512V/32768 = 16 µV. Of course if the actual voltage exceeds this magnitude, the value read will be the saturated (maximum) value – so care must be taken to use an appropriate gain value so this doesn't happen.

The ADS1115 datasheet is available from TI at: http://www.ti.com/lit/ds/symlink/ads1115.pdf

Adafruit provides free support software for the ADS1115 that hides most of the complexity and makes programming simple.

Note that there is another ADC called the ADS1015 that is virtually identical to the ADS1115 except it is a 12-bit rather than a 16-bit ADC. It is a bit less expensive (Adafruit board price is $9.95). This would reduce the resolution from 32k increments down to 2k increments. That is probably more than sufficient for the IV Swinger, but for a $5 difference it's probably not worth changing. The software change is trivial however.

4.2.2 Voltmeter

The maximum voltage that we need to measure is the maximum V_{OC} value of 80V. This is much higher than the 5V limit of the ADC inputs, so it is necessary to scale it down. This is accomplished with a simple voltage divider circuit.

A typical voltage divider is two resistors in series, but we use three. These are resistors R1, R2, and R3 shown in the schematic view in Figure 4-3 on page 54 and the breadboard view in Figure 4-4 on page 55. The top of the voltage divider is connected to the positive cable from the PV panel (PV+). The bottom of the voltage divider is connected to the negative cable from the PV panel (PV-), which is defined as ground in the IV Swinger. The ADC differential inputs A0 and A1 measure the voltage across resistor R2.

In addition to the voltage divider, there are two Schottky diode clamp circuits (diodes D0/D1 and D2/D3). The Schottky clamps assure that the voltage seen at the ADC inputs cannot be greater than +5V (plus V_{fwd} of the diode) or less than 0V (minus V_{fwd} of diode). This protects the ADC inputs and is recommended on p.11 of the ADS1115 data sheet. 1N5819 Schottky diodes are used for the clamps. They have a V_{fwd} of 0.6V at 1A and 0.9V at 3A. This is a bit higher than we'd like because the ADS1115 data sheet says that the input voltage must be between -0.3V and +5.3V. The 1N5817 would be a better choice – it has a lower V_{fwd} (but still higher than 0.3V).

The reason for resistor R3 is to limit the current in the event that the PV is connected backwards. Without R3, if PV- is greater than +5V, current would flow unimpeded through Schottky diode D3 into the +5V rail and there might not be enough load to sink that much current, and that could damage the battery pack.

The equation for a three-resistor voltage divider where the output is measured across the middle (R2) resistor is:

$$V_{R2} = \frac{R2}{R1 + R2 + R3} \cdot V_{in}$$

or, solving for V_{in}:

$$V_{in} = \frac{R1 + R2 + R3}{R2} \cdot V_{R2}$$

The ADC measures V_{R2}, so the second equation is used by the software to determine the voltage between the PV+ and PV- outputs of the solar panel.

The values used for the resistors are:

$$R1 = 180\ k\Omega$$
$$R2 = 8.2\ k\Omega$$
$$R3 = 5.6\ k\Omega$$

So R2/(R1+R2+R3) ≈ 1/24, which means the PV voltage is divided by approximately 24 before being measured by the ADC. In order to exceed the 5V input limit of the ADC, a voltage of 5*24=120V would have to be seen. This is 40V of safety margin over the 80V maximum V_{OC} design point.

Resistors in the kΩ range are chosen to limit the amount of current through the voltage divider. At 80V, the current would be V/R=80V/(180000Ω+8200Ω+5600Ω)= 0.4 mA. This is negligible compared to the current being generated by the PV panel, so it satisfies the requirement that the meters not affect what is being measured. The input impedance of the ADC input must also be considered. Table 2 on p.13 of the ADS1115 spec lists the differential input impedance for different PGA gain values. At the low end it is 170kΩ. The R2 and R3 resistances need to be significantly lower than this so most of the current goes through them and not into the ADC inputs.

The power rating of the resistors has to be checked too. Power is I^2R. The maximum current as noted in the previous paragraph is 0.4 mA. R1 is the largest of the resistors. $I^2R = (0.0004A)^2 \cdot 180000\Omega =$ 0.029W. So ¼ W resistors are more than adequate.

As mentioned above, the purpose of resistor R3 is to limit the current if the user connects the PV backwards. If this happens, diode D2 will be forward-biased, clamping the voltage at the A1 ADC input to about 5V. If the PV voltage is 80V, the voltage across R3 will be 75V and the current through R3 will be I = V/R = 75V/5600Ω = 13.4 mA. The power dissipation will be $(0.0134A)^2 \cdot 5600\Omega$ = 1W. So R3 should be rated at 1W[7].

The resistors used have a tolerance of ±5%. In the interest of accuracy it is easy enough to measure their actual resistance with a multimeter before building them into the circuit and use the measured values in the software. The current software has these values hardcoded, so the code must be edited to use the actual measured values. It would be a nice enhancement to have a user configuration file separate from the code to provide these custom values for each IV Swinger.

[7] The first IV Swinger has a ¼ W R3, so it cannot handle a reversed PV with V_{OC} > 37.5V. At least not for very long. Four 22kΩ ¼ W resistors in parallel would work as a replacement for a single 1W 5.6kΩ resistor.

4.2.3 Ammeter

There are two common ways to measure current:

- Hall-effect sensor
- Shunt resistor

A Hall-effect sensor measures the magnetic field created by the current and outputs a voltage proportional to the current. An ACS712 Hall-effect current sensor is very cheap and small and can measure up to 30A. The catch is that it only works if there are no other magnetic fields around. The IV Swinger has 16 relays, each containing an electromagnet. That's a deal-breaker.

That leaves us with the shunt resistor method. A shunt resistor is simply a very low resistance high precision resistor. By measuring the voltage drop across the shunt, the current through it can be calculated using Ohm's Law. Because of its low resistance it dissipates little power and therefore has a negligible effect on the values being measured.

The shunt resistor used in the IV Swinger is shown in Figure 4-2 below.

Figure 4-2: Shunt resistor

It cost $3.62 on Amazon. It is specified as 10A/75mV. This means that it can handle up to 10A of current, and the voltage across it at 10A is 75mV. Translating to resistance, this is R = V/I = 0.075V/10A = 0.0075 Ω = 7.5 mΩ. Physically this shunt resistor is rather large, but it conveniently provides a mechanically solid attachment point for the PV cables in the enclosure.

In order to measure the current through the load circuit, the shunt resistor must be part of that circuit. But where in the circuit should it go? It seems as if it shouldn't matter, since the current will be the same regardless of where it is. But since we want to measure the voltage across the shunt, it simplifies things if one end of the shunt is at the ground point in the circuit. This is known as "low-side" current sensing.

We now have the opposite problem that we had with the voltmeter: the voltage range is too small for the ADC inputs. With the ADC's PGA programmed to its highest gain of 16, the range is ±256mV and that is still a lot larger than 75mV so we'd lose a lot of resolution. We need to multiply the voltage across the shunt resistor before feeding it to the ADC inputs. This is done with a simple non-inverting op amp circuit.

The non-inverting op amp circuit is shown in the schematic view in Figure 4-3 on page 54 and the breadboard view in Figure 4-4 on page 55. The op amp is a TLV2462 ($2.95 from Adafruit). The TLV2462 contains two op amps but we only use one. Resistor R_g is 1.5 kΩ and is connected from the (–) input of the op amp to ground (which is also the low side of the shunt). Resistor R_f is 82 kΩ and is connected from the op amp output to its (–) input. The high side of the shunt resistor is connected to the (+) input of the op amp. The output of the op amp is connected to the A2 input of the ADC and the A3 input of the ADC is connected to ground. The gain of the amplifier is:

$$Gain = 1 + \frac{R_f}{R_g} = 1 + \frac{82}{1.5} = 55.67$$

A current of 10A will produce a voltage across the shunt of 75mV, which will be amplified to 0.075V * 55.67 = 4.18V which is approximately equal to the maximum value in the ADC range with the PGA gain set at 1.

The TLV2462 is a "rail-to-rail" op amp, meaning it can generate output voltages very close to 0V at the low end and +5V at the high end. Other op amps can bottom out at values over one volt above the ground rail and can max out at values over one volt below the VDD rail (+5V in our case). This would cause inaccurate current measurements.

The values of the resistors were chosen as follows: Figure 38 on p.27 of the ADS1115 data sheet shows a low-side current monitor where the value of R_g is 1kΩ. We'll assume that's a reasonable value. We want a ratio of 0.075:4.096, which is 1:54.6. If R_g is 1kΩ, the closest standard value for R_f would be 56 kΩ for a ratio of 1:56 (gain of 57). Bumping up R_g to 1.5 kΩ and using 82 kΩ for R_f gets us 1:54.7, which is almost perfect.

The power dissipated by the shunt resistor at 10A is $I^2R = (10A)^2 \cdot 0.0075\Omega = 0.75$ W which is negligible compared to the power generated by the PV panel.

4.2.4 Schematic View

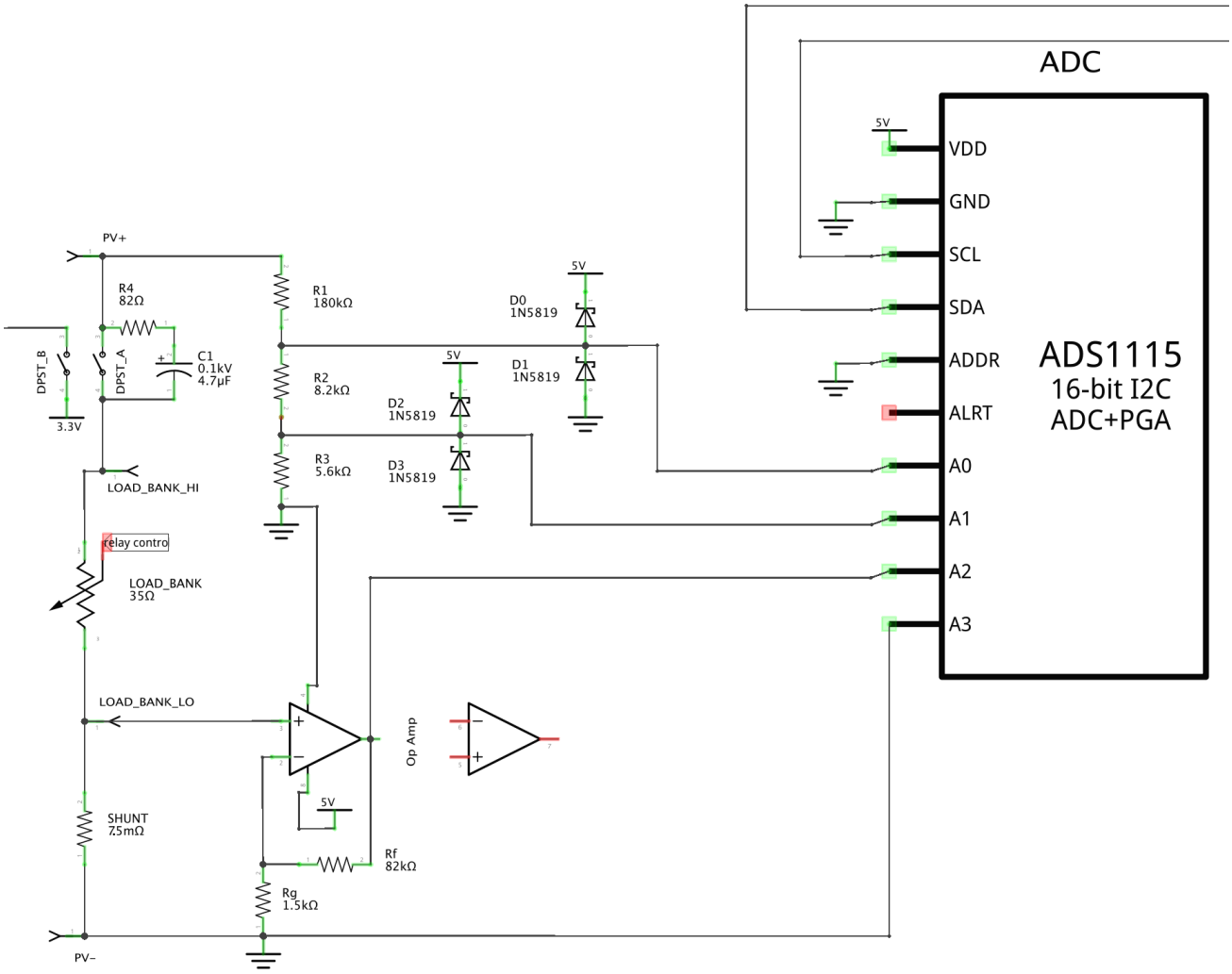


Figure 4-3: Meters schematic

4.2.5 Breadboard View

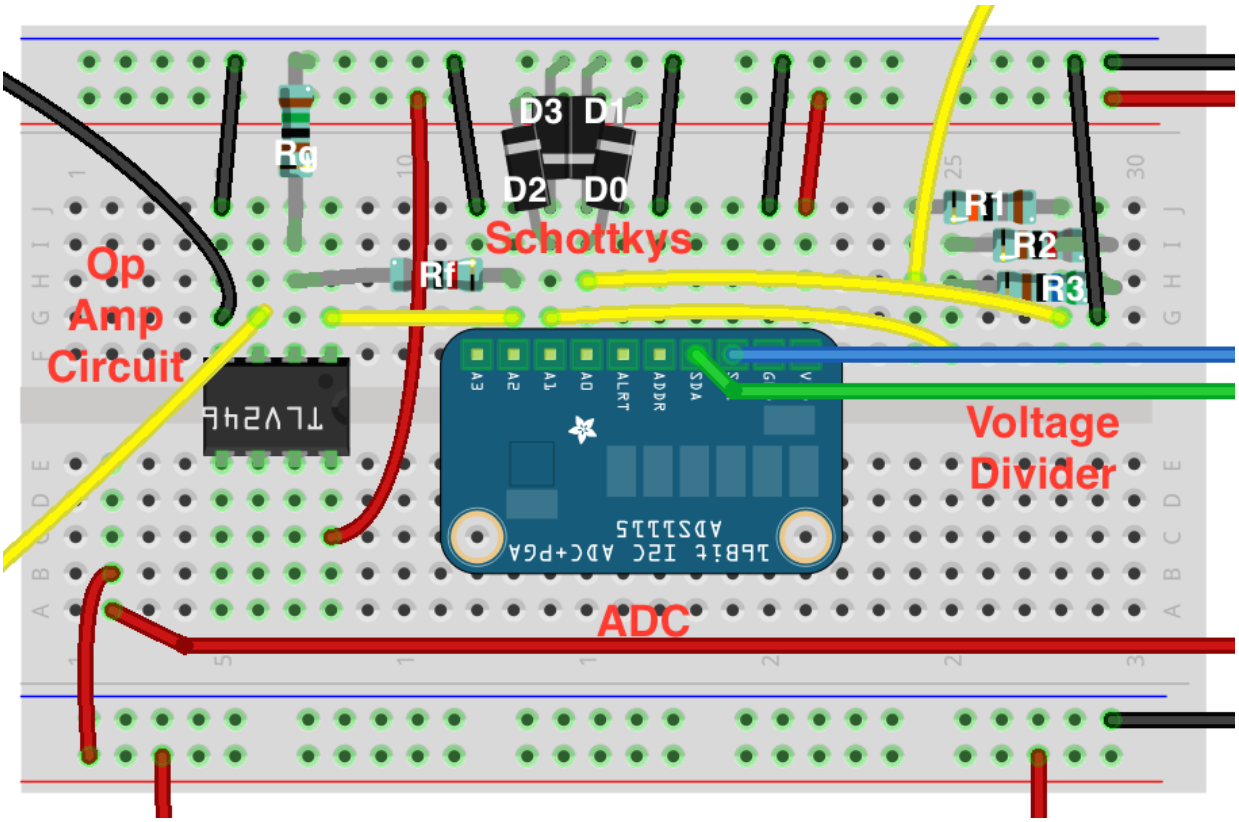


Figure 4-4: Meters breadboard

5 Computer and Other Electronics

5.1 Raspberry Pi

It would not be possible to build the IV Swinger at a low cost if it were not for the availability of a small inexpensive single-board computer (SBC) with General Purpose I/O (GPIO) pins that can be used to control and observe things. The Raspberry Pi was not the first low cost SBC to hit the market, but it was the one that started the "revolution" and is by far the most popular (5M sold as of 2/2015), with a rich ecosystem of software and hardware and a strong support community.

The truth is that the Raspberry Pi was not chosen for the IV Swinger; no other alternatives were considered. I had a Raspberry Pi before I had the idea of the IV Swinger! I wanted to build something with the Raspberry Pi, and the IV Swinger came out of that brainstorming[8].

The first (and only, so far) IV Swinger uses the Raspberry Pi Generation 1 Model B+. That was the top model at the time I was designing and building it. At time I am writing this, the Generation 2 Model B has superseded the Generation 1 Model B+, but the Gen 1 B+ is still available and is now discounted ($29.95). Certainly as time marches on, the older models will be discontinued and newer models will come out. The Gen 2 Model B should work fine in place of the Gen 1 Model B+. Presumably future models will also work fine in the IV Swinger, but that is hard to say with certainty at this point. It seems hard to imagine anything they could do that would preclude using a future generation Raspberry Pi but it is certainly possible that modifications to the hardware and software could be required.

5.1.1 Gen 1 Model B+ features

- Physical dimensions: 85.60 mm × 56.5 mm (3.370 in × 2.224 in) – not including protruding connectors
- Weight: 45 g (1.6 oz.)
- 700 MHz ARM CPU
- 512MB memory (SDRAM)
- MicroSD storage
- 4 USB ports
- 17 General Purpose I/O (GPIO) pins
- I^2C bus support
- HDMI
- Power consumption: 3.0W (600mA)
- Power supply: 5VDC (MicroUSB)

5.1.2 Why not Arduino?

The Arduino is a single-board microcontroller. It has been around longer than the Raspberry Pi and is also wildly popular. The Arduino is also very inexpensive, even smaller, and has GPIO pins that can be used to control and observe the physical environment under the control of software. It even has a built-in ADC. There's a major difference between the two, however: the Arduino is a microcontroller, not a computer. There's no operating system; the software is developed on an external computer and

[8] My son Ryan was a TA for the CEE176B course at Stanford in spring of 2014 and I had sat in on the class.

downloaded to the Arduino. The Raspberry Pi is a full-fledged computer that runs the Linux operating system. A major advantage of the Arduino for many applications is that it is inherently real-time. However, that is not important for the IV Swinger.

The fact that the Raspberry Pi is a full computer makes it possible and pretty easy to do things that would be impossible or at least very difficult with an Arduino, such as:

- Use multithreaded software (Python in our case)
- Generate PDF graphs of the IV curves
- Store the results on a user's USB thumb drive

Although it would have been possible to build an Arduino-based IV curve tracer, it would have been quite different from the IV Swinger in many ways. The Arduino was never considered for the IV Swinger because, as mentioned earlier, the idea of using the Raspberry Pi to build something useful came before the idea of the IV Swinger!

5.2 MicroSD card

The Raspberry Pi uses a MicroSD card for its "disk". This is not included with the Raspberry Pi unless it is purchased in a "bundle". A 16GB MicroSD card is more than adequate. In fact an 8GB card should be more than adequate. In addition to writing the output files to the USB flash drive(s), all files are also written to the SD card, and there are also log files written to the SD card for every run. Over time these will take up more and more space, but they are not big enough to make much of a dent in the free space on cards with this much storage, even after thousands of runs.

5.3 HDMI extension

When the Raspberry Pi is mounted inside the enclosure, the HDMI port is not directly accessible because it is on the bottom edge. In normal usage there is no need for the HDMI port, but it can be useful to have a directly connected monitor for debugging, software updates, etc. The other option is to use a USB wi-fi adapter and configure the Raspberry Pi as a wi-fi access point; this makes it possible to use a laptop to connect to the Raspberry Pi (using VNC) without a keyboard, mouse or monitor. But that configuration can be tricky and is not covered in this document. So it is recommended to have access to the HDMI port from outside the enclosure.

For this purpose, a "6-inch Right Angle Short HDMI 1.4 Male to Female Extension Cable Adapter" as shown in Figure 5-1 below is used in the current design. The right angle connector on the male end is necessary for clearance over the RELAY_LO module. Unfortunately, this cable is difficult to find. It is not available on Amazon. It is available on EBay, but only from Hong Kong or China – and is around $10. It would probably work to use a right angle male-to-female HDMI adapter and a straight cable.

Zip ties are used to strap the female end of the cable to the Raspberry Pi, below the Ethernet port. See Figure 2-7 and Figure 2-8 on page 16.

Figure 5-1: HDMI extension cable

5.4 I²C Bus

The I²C bus was mentioned earlier in Section 4.2.1 (in the discussion of the ADS1115 ADC). I²C has been around for over 30 years, and you can read all about it on Wikipedia if you want to know more about how it works. All you really need to know is that with only two GPIO pins, the Raspberry Pi is able to communicate with multiple other devices, and the software to abstract away all of the low level detail is freely available. With just these two pins and the free software, it is very easy to instruct I²C devices to do things for us and to request information from them.

The IV Swinger uses I²C to communicate with the following devices:

- MCP23017 I/O expander (Section 5.6 below)
- ADS1115 ADC (Section 4.2.1 above)
- DS1307 Real Time Clock (Section 5.9 below)

The Raspberry Pi is the I²C master and the other devices are I²C slaves. This just means that all transactions are initiated by the Raspberry Pi and completed by the other devices. The two I²C signals are called Serial Data (SDA) and Serial Clock (SCL), and GPIO pin 2 and GPIO pin 3 respectively are used for these signals.

5.5 Perma-Proto boards

Adafruit sells what they call "Perma-Proto" breadboards. These are sort of a "poor man's PCB". The IV Swinger uses two 1/2 Sized Perma-Proto boards to mount and connect most of the supporting electronics[9]. These cost $4.50 each from Adafruit. Figure 5-2 and Figure 5-3 below show the front and back of a 1/2 sized Perma-Proto board. It is basically just like a standard breadboard except you solder components to it (hence the "Perma"). You can see the connectivity from the back view: the two rows of holes on the top and bottom are power and ground rails, and the other holes are organized in columns of five. Unlike a standard breadboard, it is possible to put wires and components on the back of the board.

[9] A single full-size Perma-Proto could be used instead of the two 1/2 size boards, but there would be some slight differences (due to the continuous power rails). I bought the smaller boards long before most of the design details were worked out so I just used what I had.

One nice thing about using the Perma-Proto boards is that Fritzing's breadboard view represents exactly how the components and wires are laid out and connected.

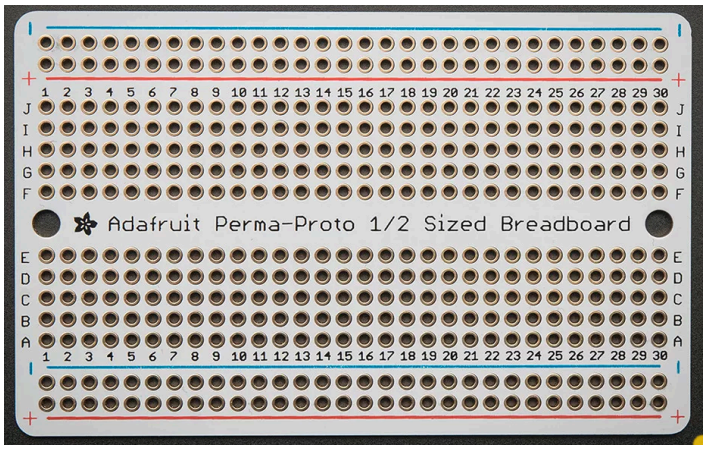


Figure 5-2: Perma-Proto (front)

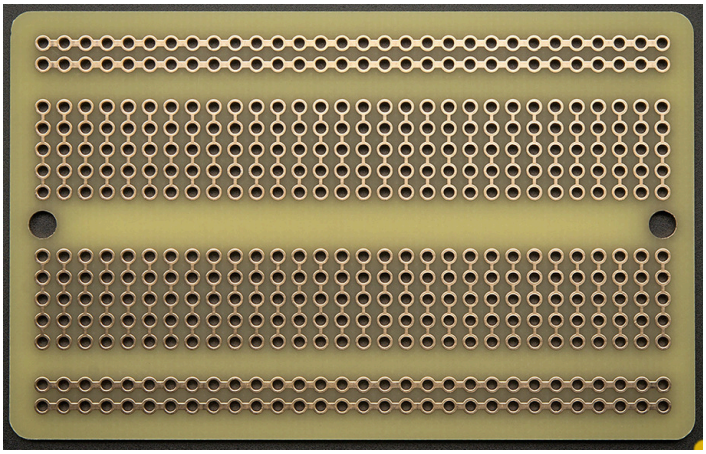


Figure 5-3: Perma-Proto (back)

For the purpose of distinguishing between the two, the remainder of this document will use "Perma-Proto A" for the one on the left (as seen from the front), and "Perma-Proto B" for the one on the right, as shown below in Figure 5-4.

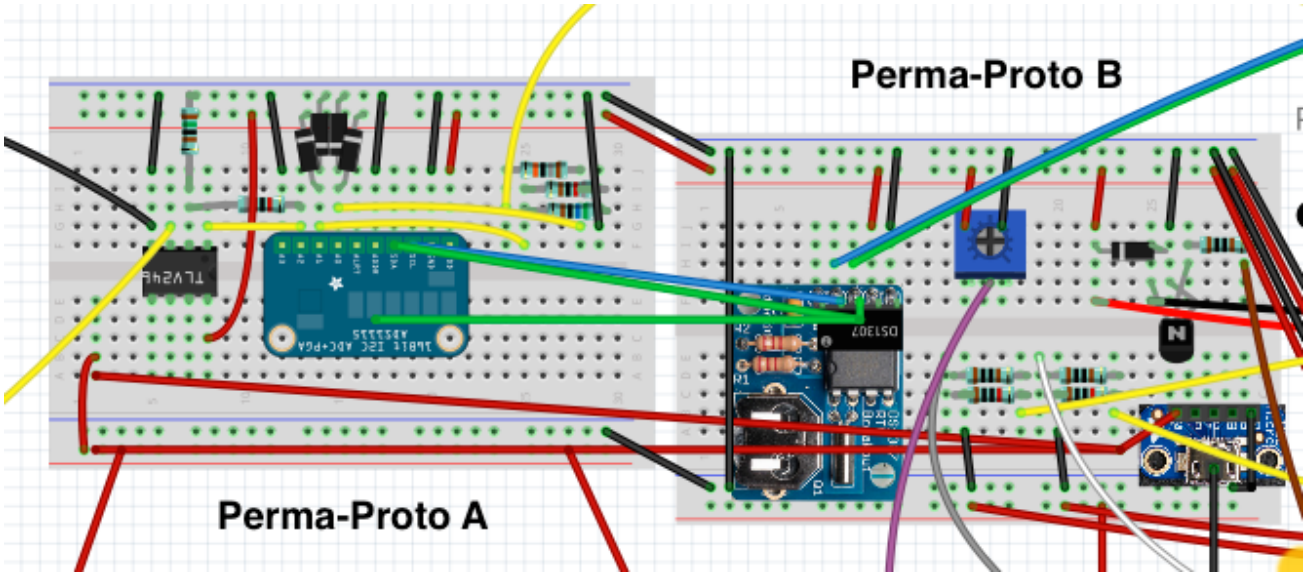


Figure 5-4: Perma-Protos A and B

5.6 MCP23017 and "Slice of PI/O" expansion board

The Raspberry Pi Model B+ has 17 GPIO pins. Each of the 16 relays has one pin that controls whether it is activated or not. If one GPIO pin were used for each relay, there would only be one left for anything else.

One solution to this would be to implement external decode logic so that a small number of GPIO pins could control the 16 relays. This would be possible because only about 33 out of all 2^{16} combinations of the relay pins are actually used. 6 GPIO pins would be enough to encode up to 64 values. Implementing the decode logic would require discrete logic gates or a lookup table (ROM/PROM).

A simpler and more flexible solution is to use the MCP23017 "port expander" chip to effectively add 16 more GPIO pins. The Raspberry Pi communicates to the MCP23017 using I^2C, which uses only two pins (and those same two pins are also used to communicate with other I^2C devices). The MCP23017 supports I^2C commands that tell it what value (0 or 1) to place on each of the 16 output pins[10]. By computer standards, this process of sending commands to change the values of the pins is quite slow, but it's virtually instantaneous relative to human perception, and plenty fast enough for our needs.

The IV Swinger uses a small expansion board called the "Slice of PI/O" (shown in Figure 5-5 below) that includes the MCP23017 and is designed to piggyback on the Raspberry Pi. It comes as a kit, so you have to solder the parts on yourself.

[10] The pins can also be used as inputs, but the IV Swinger uses them all strictly as outputs.

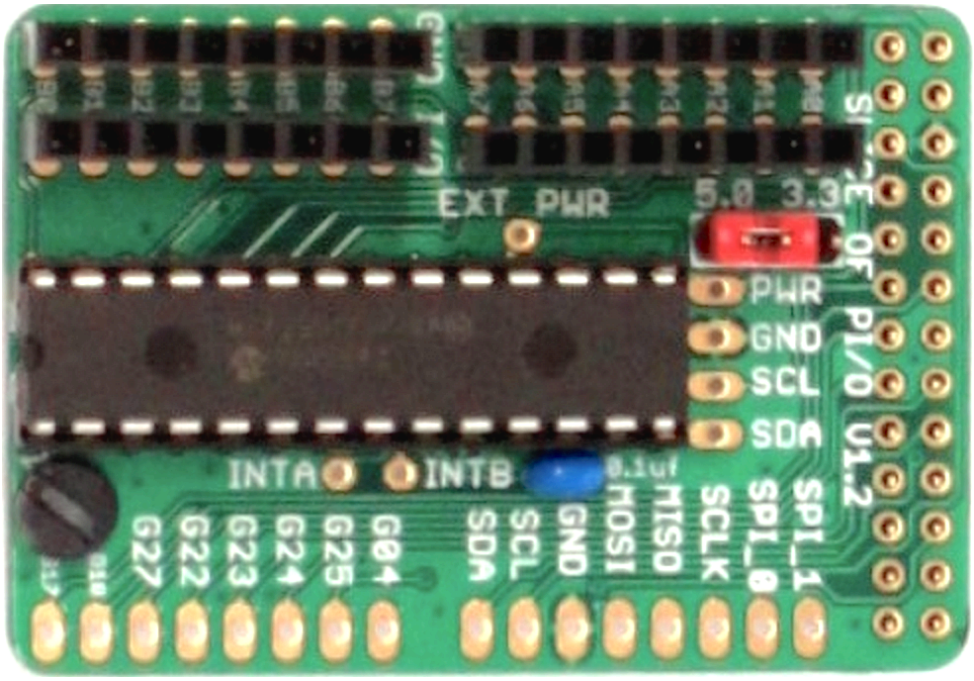


Figure 5-5: Slice of PI/O with MCP23017

The MCP23017 is in a socket in the middle of the board. The two rows of female jumper headers at the top are ground pins (upper row) and the extended I/O pins (lower row). The extended I/O pins are (from left to right in the photo): B0:7, A7:0. It is important to note that the order of the "B" pins is opposite from the "A" pins (this was actually marked incorrectly on the v1.0 version of the board). The pins along the right side of the board are soldered to a female connector underneath the board, and this connector is plugged onto the Raspberry Pi GPIO pins. The holes along the bottom edge extend the GPIO pins so they are still accessible when the Slice of PI/O board is mated to the Raspberry Pi. This photo does not show male pin headers soldered into these holes, but adding these makes it possible to connect to the GPIO pins with standard jumper wires without soldering. The short ends of the pins go through the holes and are soldered on the back so the long end is facing out/up. IV Swinger only uses the G17..G04 pins in the left hand group of eight, so it is not necessary to solder pins to the other eight. There are also four holes to the right of the MCP23017, and these also need four pins of male header[11]. The PWR pin is used to connect +5V to the VCC pin on each of the relay modules. The GND pin is used to connect to the ground rail on the Perma-Proto boards. The SCL and SDA pins are the I^2C bus pins (connected by traces to GPIO pins 3 and 2, and to the MCP23017 pins), and are used to extend the I^2C bus to the ADC and RTC modules on the Perma-Proto boards. Note the red jumper just above these four pins. It must be in the position shown in the photo, connecting the middle pin to the one labeled "5.0". This drives the extension pins at 5V, which is what the relays require. We use the pin labeled "3.3" to connect to the lower power rail of Perma-Proto B. This is the +3.3V rail that is used by the DPST sensing circuit and the shutdown button sensing circuit.

The I^2C address of the MCP23017 must also be configured by soldering the A0, A1, and A2 pins to GND on the back of the Slice of PI/O board as shown below in Figure 5-6. This places the MCP23017 at I^2C address 0x20.

[11] The 8-pin male header and the 4-pin male header are not included in the Slice of PI/O kit and must be purchased separately. The LCD display kit has 16 extra pins of header that can be used for this.

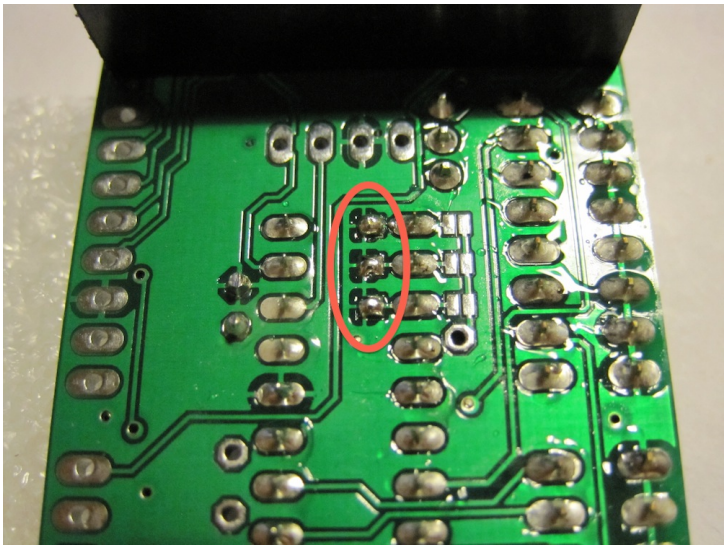


Figure 5-6: Slice of PI/O address configuration

The connections to/from the Slice of PI/O board on its top side are as follows:

- A0 (I/O): connects to RELAY_HI pin IN8
- A1 (I/O): connects to RELAY_HI pin IN7
- A2 (I/O): connects to RELAY_HI pin IN6
- A3 (I/O): connects to RELAY_HI pin IN5
- A4 (I/O): connects to RELAY_HI pin IN4
- A5 (I/O): connects to RELAY_HI pin IN3
- A6 (I/O): connects to RELAY_HI pin IN2
- A7 (I/O): connects to RELAY_HI pin IN1
- B7 (I/O): connects to RELAY_LO pin IN8
- B6 (I/O): connects to RELAY_LO pin IN7
- B5 (I/O): connects to RELAY_LO pin IN6
- B4 (I/O): connects to RELAY_LO pin IN5
- B3 (I/O): connects to RELAY_LO pin IN4
- B2 (I/O): connects to RELAY_LO pin IN3
- B1 (I/O): connects to RELAY_LO pin IN2
- B0 (I/O): connects to RELAY_LO pin IN1
- B0 (GND): connects to RELAY_LO pin GND
- G04: connects to DPST sensing circuit resistor R5
- G25: connect to LCD pin 4
- G24: connects to LCD pin 6
- G23: connects to LCD pin 11
- G22: connects to LCD pin 14
- G27: connects to LCD pin 13
- G18: connects to piezo buzzer circuit resistor R4
- G17: connects to LCD pin 12
- PWR: connects to RELAY_HI and RELAY_LO pin VCC
- GND: connects to top ground rail of Perma-Proto B
- SCL: connects to RTC module on Perma-Proto B
- SDA: connects to RTC module on Perma-Proto B

RELAY_LO is the 8-relay module on the right when looking from the front; it is connected to the lower numbered immersion coil loads (HALF..FIVE). RELAY_HI is the 8-relay module on the left when looking from the front; it is connected to the 6Ω power resistors and the higher numbered immersion coil loads (SIX..ELEVEN). Figure 5-7 below shows the connections between the Slice of PI/O and the relays. The I/O wires are all female on the relay end and male on the Slice of PI/O end. For neatness, these are ribbons of 8 wires each. The figure also shows which pins on the Raspberry Pi the Slice of PI/O plugs onto.

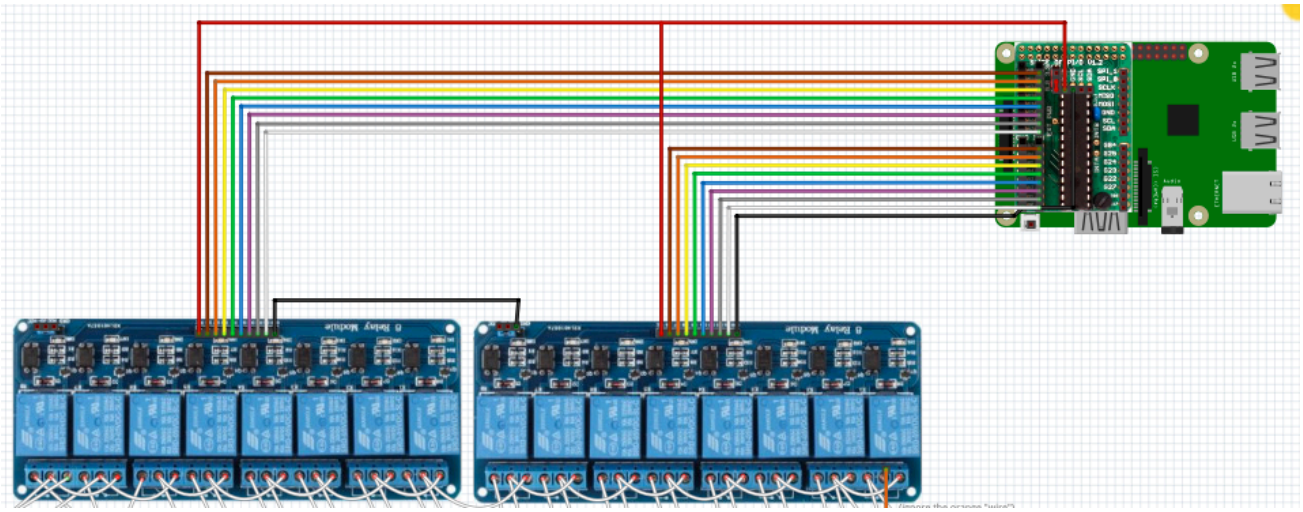


Figure 5-7: Slice of PI/O connections to relays

5.7 DPST sensing circuit

The double-pole single-throw (DPST) switch was discussed earlier in Section 3.2.5. The A-B side (DPST_A in the schematic) is used to open or close the load circuit itself. The C-D side (DPST_B in the schematic) is connected to a GPIO pin on the Raspberry Pi so the software can detect if the load circuit is open or closed. To be more specific: the C terminal of the DPST (Figure 3-16, p.42) is connected to +3.3V; the D terminal is connected to one end of a 10kΩ resistor to ground and through a 1kΩ resistor to GPIO pin 4. When the switch is open (OFF) the 10kΩ pulldown resistor keeps the voltage seen by the GPIO pin low. When the switch is closed (ON) the +3.3V keeps the voltage seen by the GPIO pin high. The 1kΩ resistor limits the current into the GPIO pin. This is illustrated (in context) in the schematic in Figure 2-3 on page 13 and (by itself) in Figure 5-8 below.

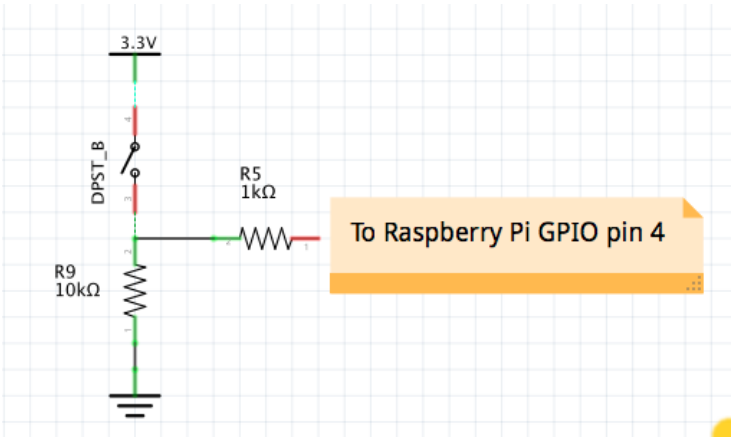


Figure 5-8: DPST sensing circuit

Physically this circuit is just to the right of the DS1307 module on Perma-Proto B as shown in Figure 5-9 below.

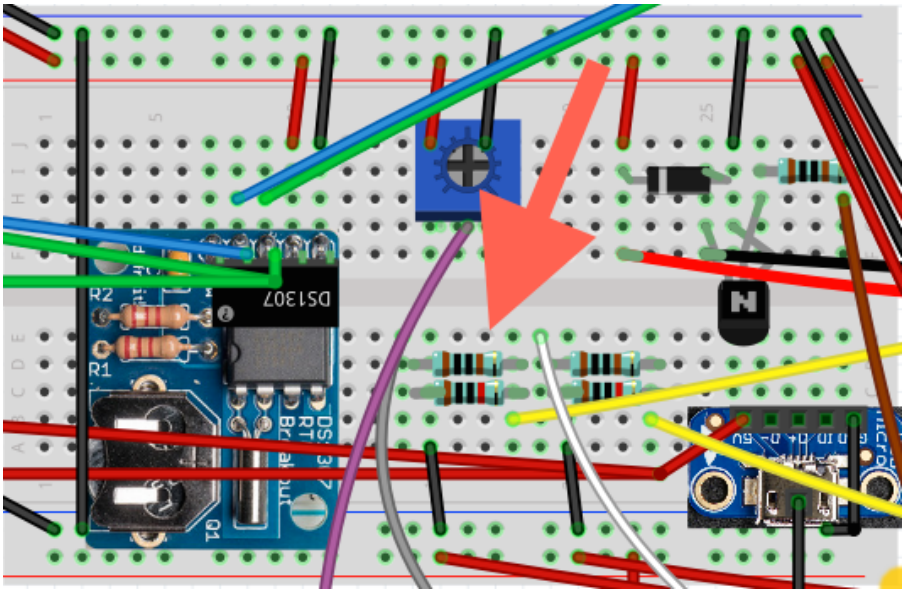


Figure 5-9: DPST sensing circuit on Perma-Proto B

The yellow wire is from the D terminal of the DPST and the gray wire goes to GPIO pin 4 on the Raspberry Pi (actually the Slice of PI/O board). The upper resistor is R5 (1kΩ) and the lower resistor is R9 (10kΩ). The black wire below the resistors connects the left end of R9 to the lower ground rail. The red wire below that connects the C terminal of the DPST to the lower power rail (+3.3V).

5.8 LCD display

The display is used to communicate information to the user. A standard 16x2 character LCD display (1602A module with HD44780 controller) was chosen because it is adequate for the purpose, low power, well supported in terms of software and information about how to use it, and is inexpensive ($9.95 from Adafruit; $1.90 from China on EBay). The 16x2 means it has 2 rows of 16 characters each. The background color is blue, and the characters are white. It is reasonably easy to read in sunlight.

Figure 5-10 below shows the front view of the LCD module. This picture also shows the pin header and 10kΩ contrast adjustment potentiometer that are included from Adafruit (not included in the China/EBay option, but required). 16 of the pins are soldered into the holes along the top edge (small ends poking through the holes from the back, solder on the front). As seen from the top, pin #1 is on the left, and pin #16 is on the right. The excess pin header can be used for the Slice of PI/O.

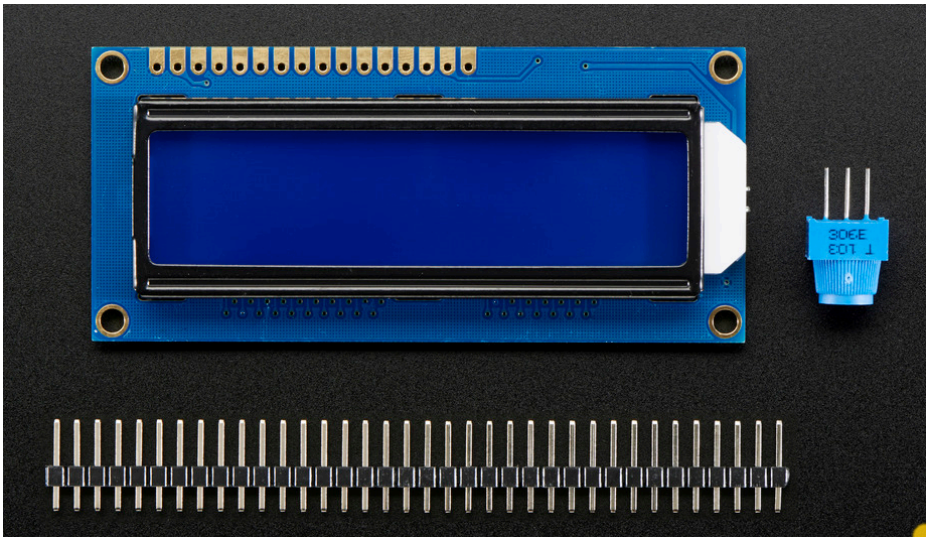


Figure 5-10: LCD front, with pin header and potentiometer

Figure 5-11 below shows the back view of the LCD module. You can see that holes #16 and #1 are labeled, and there is a printed table of the pin names.

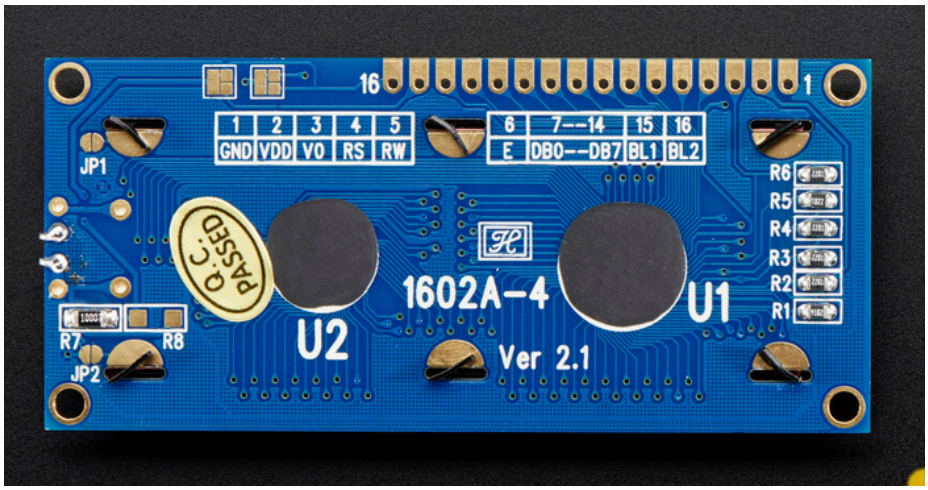


Figure 5-11: LCD back

The LCD display is connected as follows:

- Pin #1: (GND) connects to ground (black wire)
- Pin #2: (VDD) connects to +5V (red wire)
- Pin #3: (Vo) connects to the middle of the potentiometer (purple wire)
- Pin #4 :(RS) connects to GPIO 25 (yellow wire)
- Pin #5: (RW) connects to ground (black wire)
- Pin #6: (EN) connects to GPIO 24 (green wire)
- Pin #7: (D0) is left unconnected
- Pin #8: (D1) is left unconnected
- Pin #9: (D2) is left unconnected
- Pin #10: (D3) is left unconnected
- Pin #11 (D4) connects to GPIO 23 (blue wire)
- Pin #12 (D5) connects to GPIO 17 (purple wire)
- Pin #13 (D6) connects to GPIO 27 (gray wire)
- Pin #14 (D7) connects to GPIO 22 (white wire)
- Pin #15: (LED +) connects to +5V (red wire)
- Pin #16: (LED -) connects to ground (black wire)

The GPIO connections go to the header on the Slice of PI/O board as shown below in Figure 5-12.

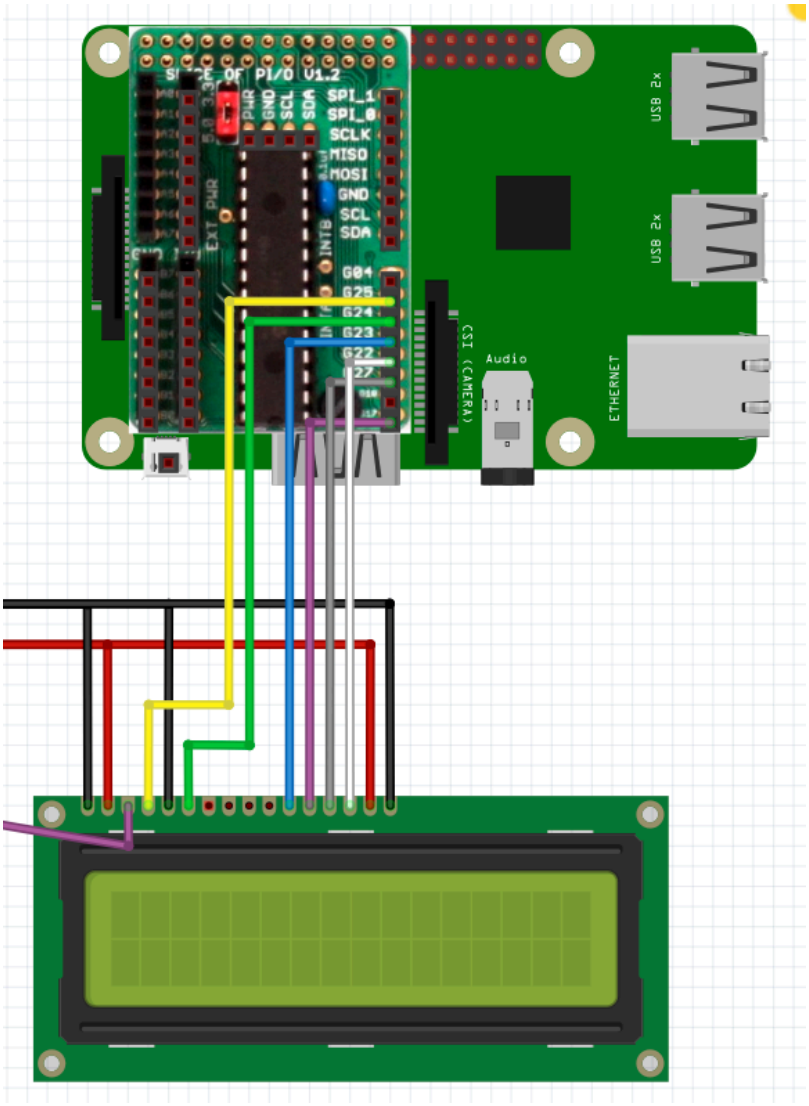


Figure 5-12: Slice of PI/O connections to LCD

The power and ground wires are connected to the upper power rail on Perma-Proto B. The 10kΩ potentiometer is used to adjust the contrast of the display. It is soldered onto the back side of Perma-Proto B. A hole in the acrylic case provides access to it for adjustment. Its three pins are connected as follows:

- Pin #1: connects to +5V (red wire)
- Pin #2: connects to pin #3 of the LCD (purple wire)
- Pin #3: connects to ground (black wire)

Pin #2 is the middle one. The other two are #1 and #3, but it isn't really important which is which since the only difference it makes is which direction the knob is turned to increase/decrease contrast. The blue potentiometer can be seen in the photograph of the back of the IV Swinger (Figure 2-8 on page 16). It is shown as if it were on the front of Perma-Proto A on the Fritzing breadboard views in Figure 2-2 on page 12 and Figure 5-4 on page 60.

5.9 Real Time Clock

The Raspberry Pi does not have a built-in battery-backed real-time clock (RTC). Instead, it updates the time-of-day from the Internet. This works fine for most users, and keeps the cost of the Raspberry Pi low for everyone by not including a feature that is not necessary in the vast majority of cases.

Although it is possible to connect the IV Swinger to a local network (and therefore to the Internet) with an Ethernet cable or wi-fi adapter, this is not generally going to be the case. Without a battery-backed RTC, the time-of-day would be set at boot time to 00:00:00, 1/1/1970.

Why does this matter? The IV Swinger software names the output folders and files based on the current date/time. While it would have been possible to use a different naming convention such as a simple incrementing number, it seemed to be worth the extra cost to be able to use timestamps so it wouldn't be possible to lose track of when a trace was taken. In addition to the timestamp-based names, the Linux file system timestamps are also dependent on knowing the correct date and time. Without the RTC, it would have been necessary for the users to manually keep track of when each test was run.

Fortunately it is easy to add a battery-backed RTC. Adafruit sells a RTC module based on the DS1307 chip for $9 (including the battery). This is sold as a simple kit that requires soldering the components onto the breakout board. The Raspberry Pi communicates with the DS1307 using the I^2C bus. Software can set the clock (e.g. when it is connected to the Internet) and it can read the value of the clock when the Internet is not available at boot time.

The DS1307 module sits on Perma-Proto B as seen in Figure 5-4 on page 60 and Figure 5-9 on page 65. Its only connections are power (5V), ground, and I^2C (SCL and SDA). As recommended in the Adafruit instructions, the 2.2kΩ resistors (R1 and R2) are not used, allowing the Raspberry Pi to communicate with the DS1307 using 3.3V signaling.

5.10 Piezo buzzer

The IV Swinger needs a way to get the user's attention. There are some fairly urgent conditions where simply displaying a message on the LCD might not be noticed and damage could occur. The piezo buzzer serves this purpose by generating a loud BEEP under software control.

There are two types of piezo buzzer: passive and active. The passive type is more flexible because it is possible to control both the tone and volume it generates, but it requires an externally generated square (or sine) wave signal. The active type has a built-in oscillator to generate the signal. There's no choice of tone, but it is much simpler to use and the built-in oscillator results in a very pure tone at the resonant frequency of the buzzer (i.e. the one that makes it the loudest). An active piezo buzzer is used for the IV Swinger. Adafruit sells an active piezo buzzer for $0.95 (it calls it a "breadboard friendly" buzzer).

Figure 5-13: Active piezo buzzer

The piezo buzzer is controlled by GPIO pin 18. The GPIO pin cannot source enough current for a loud beep, so a transistor circuit is used to turn the buzzer on and off. The GPIO pin is connected to the base of a 2N2222a NPN transistor. The collector of the transistor is connected to one side of the piezo buzzer and the emitter of the transistor is connected to ground. The other side of the piezo buzzer is connected to +5V. A Schottky diode is placed across the piezo buzzer to prevent voltage spikes from damaging the transistor[12]. When software puts a high voltage on GPIO pin 18 the transistor turns on and the buzzer sounds at maximum volume. When software puts a low voltage on GPIO pin 18 the transistor is turned off, and the buzzer is quiet. The circuit is illustrated (in context) in the schematic in Figure 2-3 on page 13 and (by itself) in Figure 5-14 below.

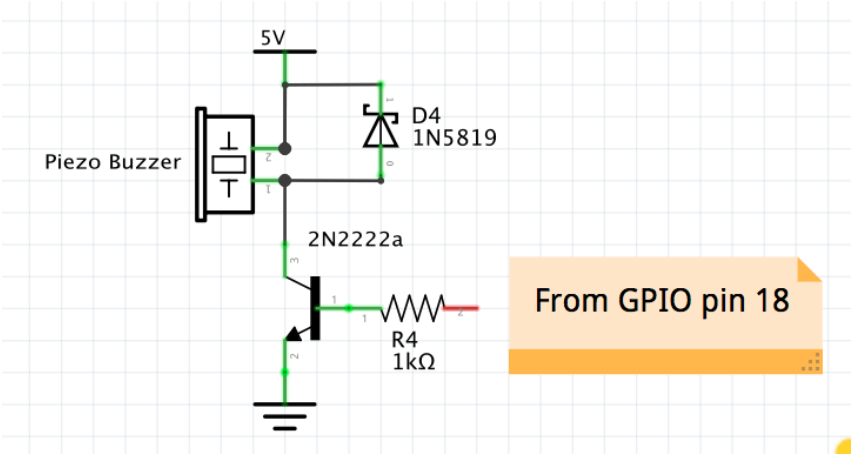


Figure 5-14: Piezo buzzer driving circuit

The piezo buzzer is attached to the back of Perma-Proto B. It is aligned with a hole in the acrylic case so the sound is not muffled by the case.

5.11 Shutdown button and sensing circuit

As with any computer, ungracefully powering the Raspberry Pi off potentially can cause problems such as file system corruption. Raspberry Pi users know that the *shutdown* command should be run before disconnecting power (there's an icon on the desktop that can be used for this too). But the IV Swinger has no keyboard for the user to type this command. This is the purpose of the shutdown button. It is monitored by the software, which issues the *shutdown* command to the operating system when the button is pressed.

The shutdown button is a "momentary" panel mount pushbutton that costs $0.95 from Sparkfun.

[12] I'm not 100% sure this is the reason for the diode. The circuit was based on the one here: http://www.instructables.com/id/PIR-Motion-Sensor-Security-Circuit-Duration-Adju/step3/The-Piezo-Buzzer-Driver-Circuit/. Unless the buzzer has a high inductance there shouldn't be voltage spikes. It certainly doesn't hurt anything, though, and we have Schottkys for the ADC input protection.

Figure 5-15: Shutdown pushbutton

The “momentary” designation distinguishes it from a “toggle” type switch that would flip its state each time it is pushed. It’s very simple – as long as the button is held down, the switch is closed. When it is released, the switch is open. This is the desired behavior because we want to be able to ignore accidental presses of the button by requiring that it be held down for at least 3 seconds before initiating the shutdown.

The pushbutton is connected to GPIO pin 5 using a circuit identical to the one used for the DPST sensing circuit described earlier in Section 5.7 on page 64. This is illustrated (in context) in the schematic in Figure 2-3 on page 13 and (by itself) in Figure 5-16 below.

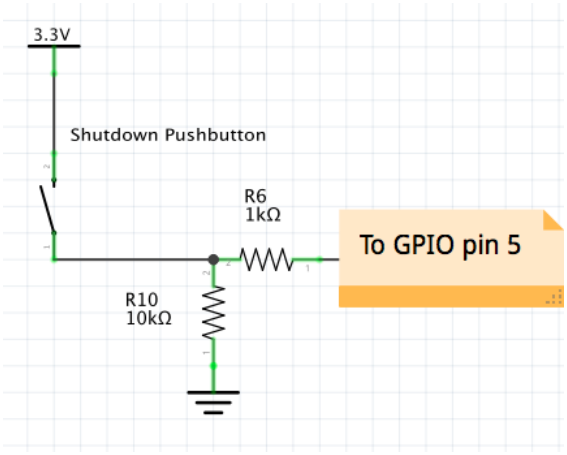


Figure 5-16: Shutdown pushbutton sensing circuit

Physically this circuit is just to the right of the DPST sensing circuit on Perma-Proto B as shown in Figure 5-17 below.

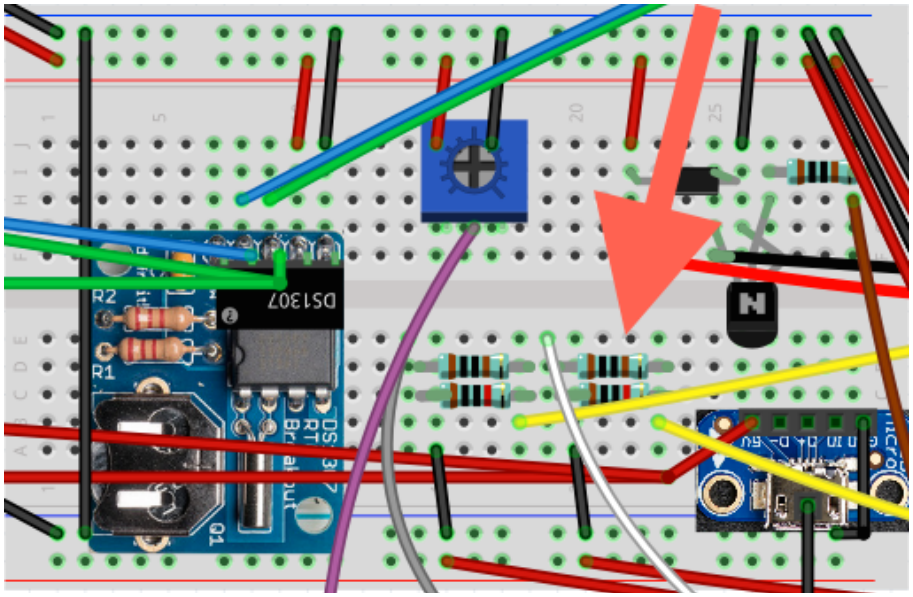


Figure 5-17: Shutdown pushbutton sensing circuit on Perma-Proto B

The yellow wire is from one of the pushbutton terminals and the white wire goes to GPIO pin 5 on the Raspberry Pi (actually the Slice of PI/O board). The upper resistor is R6 (1kΩ) and the lower resistor is R10 (10kΩ). The black wire below the resistors connects the left end of R10 to the lower ground rail. The red wire below that connects the other terminal of the pushbutton to the lower power rail (+3.3V).

6 Power

The two major consumers of power in the IV Swinger are the Raspberry Pi and the relay coils. Minor consumers of power are the MCP23017, ADS1115, LCD display, DS1307, piezo buzzer and relays (control current); but these are negligible in comparison to the Raspberry Pi and relay coils.

Since the Raspberry Pi and the relays are 5V devices (as are the minor consumers of power), we can do the accounting on the basis of current draw (mA or A) rather than Watts.

The Raspberry Pi current draw depends on how hard it is working and what is plugged into it. The raspberrypi.org website recommends a 2.5A power supply for a Model B+ that is using all 4 of its USB ports. On the other hand, it also says that the typical bare-board active current consumption is only 330mA and each USB thumb drive should draw around 150 mA, so we'd expect the current draw with 4 USB drives to be around 1000 mA (1A).

In Section 3.2.4.3 on page 38 the current required by the relay coils (electromagnets) was calculated to be 1.43A when all 16 relays are energized.

6.1 Battery pack

The battery pack used for the first IV Swinger is the EasyAcc® 12000mAh Power Bank with 4 USB outputs (model PB12000A).

Figure 6-1: Battery pack

The 4 USB outputs are rated at 0.5A, 1.3A, 1.0A, and 2.1A. The maximum total output current is 3.5A.

One very nice feature is that when the battery pack detects that no current is being drawn, it shuts off.

NOTE THAT THIS BATTERY PACK MODEL HAS BEEN DISCONTINUED. There is a newer model that is rated at a capacity of 15000mAh and has its 4 USB outputs rated at 1.0A, 1.3A, 2.0A, and 2.4A. The maximum total output current is 3.1A. This newer model should provide a bit more "headroom" if the 2.0A and 2.4A outputs are used – with the caveat that the maximum total output current actually decreased by 400mA. One catch: it is (at this writing) unavailable on Amazon, but it is still listed on the EasyAcc manufacturer's website. It is unlikely that either model will still be available

by the time you are reading this. But there are many similar products and the trend is toward higher capacities and current outputs.

Since the relays are only active for a brief time, the battery life is mostly determined by the power used by the Raspberry Pi. Assuming the total idle current draw is 500mA, the 12000mAh battery should last 24 hours and the 15000mAh battery should last 30 hours. This should be more than adequate since the software will power down the IV Swinger if it has been idle for 10 minutes, and it is expected that the battery will be recharged frequently.

6.2 Battery pack connections

The 1.3A output is used to drive the Raspberry Pi through one USB-to-Micro-USB cable. And the 2.1A output is used to drive the relay coils through a second identical cable. These cables are 1 foot long and have ends as shown below in Figure 6-2. This cable is made by StarTech and is sold by Amazon for about $4 each (search for UUSBHAUB1RA). The right angle connector on the Micro USB end is required for space reasons.

Figure 6-2: USB power cable

The cable from the 1.3A battery output connects directly to the Raspberry Pi power connector. The cable from the 2.1A output connects to a Micro USB breakout board that is soldered onto Perma-Proto B in the lower right corner (see Figure 2-2 on page 12 and/or Figure 5-17 on page 72). The Micro USB breakout is sold by Adafruit for $1.50 and is shown below in Figure 6-3.

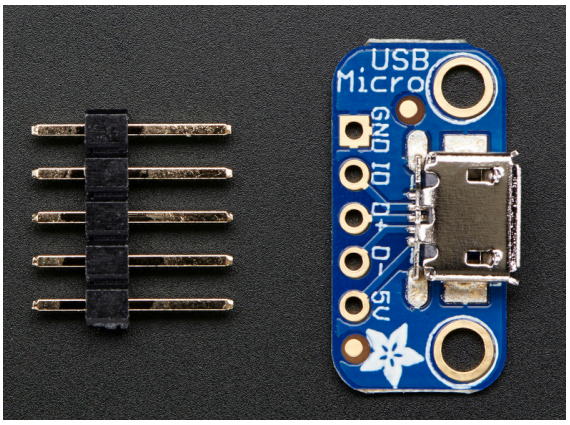


Figure 6-3: Micro USB breakout

The 5-pin header is soldered onto the breakout board, and the pins are soldered to the Perma-Proto. The ground pin is wired to the lower ground rail of Perma-Proto B, and the 5V pin is wired to the lower

power rail of Perma-Proto A. The JD-VCC pins of the relay modules are then connected to the lower power rail of Perma-Proto A. See Figure 2-2 on page 12 for details of these connections.

The battery pack itself has a Micro USB connector (marked “IN” in Figure 6-1 above) that is used for charging the battery. An extender cable is needed to make this accessible from the outside of the case. Such a cable can be purchased on EBay or Amazon for less than $2 and is shown below in Figure 6-4. In Figure 2-9 on page 17 this cable is visible from the bottom of the IV Swinger.

Figure 6-4: Battery charging extension cable

7 Enclosure

The enclosure chosen for the IV Swinger is an acrylic case sold for the purpose of displaying small model airplanes.

Figure 7-1: Enclosure (intended use)

It is manufactured by FlyteLine Models and is sold by several vendors on EBay for around $20.

This enclosure was chosen primarily for aesthetic reasons ("form over function" as they say). There are no practical reasons for using a transparent case; the IV Swinger would function fine with an opaque case. In fact there are many reasons why this was an impractical choice:

- It is breakable
- Drilling holes in acrylic is tricky (and all it takes is one mistake and the case is ruined)
- It is difficult getting all of the internal components connected and installed in the case, and it is equally difficult to perform any maintenance

However, given that the primary purpose of the IV Swinger is as a teaching tool, the visibility of all the innards has its benefits. Perhaps more than anything, it simply makes it more interesting looking, and anything that captures students' attention will foster better learning. Furthermore, it is important that students understand the basics of how the IV Swinger works so they understand the IV curves that it generates. This is one reason why it is highly recommended to go through the process of generating an IV curve manually at least once before using the IV Swinger. But being able to relate that manual process to the automated process implemented by the IV Swinger is also important, and being able to see the internal components helps in that regard.

The FlyteLine case does not come with a bottom panel. A flat piece of acrylic must be purchased separately and cut to size. Home Depot sells an 11x14" sheet of 0.093" thick clear acrylic for less than $5. It needs to be cut to 6x12".

There are a number of holes that need to be drilled in the case and bottom:

- Top: Holes for LCD standoff screws
- Top: Hole for DPST
- Front: Holes for heat sink fastening screws
- Front: Holes for button head Allen bolts
- Front: Hole for shutdown pushbutton
- Front: Hole for battery power button access
- Left: Holes for shunt resistor mounting screws
- Left: Holes for PV cables
- Left: Finger hole
- Right: Finger hole
- Right: Hole for battery charger cable
- Right: Rectangular hole for access to Raspberry Pi USB, HDMI, and Ethernet ports
- Back: Hole for piezo buzzer
- Back: Hole for LCD contrast pot adjustment
- Back: Holes for standoff screws (Raspberry Pi, Perma-Protos, relay modules)
- Back: Holes for heat sink fastening screws
- Bottom: Holes for screws for rubber feet

See Figure 2-4 through Figure 2-9 on pages 14 through 17 for the locations of these holes. HOWEVER, the following two MODIFICATIONS are recommended:

1. The LCD should be moved toward the front of the case as far as possible without the front holes being on the curved part. This will provide more clearance for the connections to the Slice of PI/O behind it.
2. The two Perma-Protos should be moved toward the left – away from the Raspberry Pi and another finger hole drilled to allow the SD card to be removed from and inserted into the Raspberry Pi.

Cutting and drilling acrylic without it cracking or shattering can be tricky. There are tips that can be found online; here's one very thorough discussion: http://www.bcae1.com/plexi.htm For the cutting (bottom and rectangular Raspberry Pi access hole) I used a jigsaw and a Bosch T101AOF laminated flooring blade. For drilling the larger holes I used Forstner bits (Ryobi 8 bit set is $20 at Home Depot). For the smaller holes (< ¼") I used normal bits. The secret is to keep the acrylic cool by stopping frequently while drilling and spraying the hole with water. Since the 11x14" sheet of acrylic for the bottom is quite a bit larger than needed, there's enough to practice both cutting and drilling before doing the real thing.

The rubber feet are "Small extra tall round rubber feet" available on EBay. The hole fits a #6 machine screw. 2" long 1/4" hex 6-32 spacer standoffs are cut in half and Gorilla glued into the two right corners of the case, and between the fins of the heat sink on the left end. 6-32 machine screws are then used to screw the feet and the bottom to the standoffs. This can be seen in Figure 2-5 through Figure 2-9 on pages 15 through 17.

8 Software

The IV Swinger software consists of the following:

- Operating system
- Utilities
- Externally developed Python library code
- IV Swinger Python code

8.1 Operating system

The Raspberry Pi operating system used for IV Swinger is the standard Raspbian distribution. This is a Debian Linux variant tailored for the Raspberry Pi. New Raspbian updates are released frequently so it is possible that there will be compatibility issues at some point, but this is unlikely. It is still recommended that the latest stable release be used for new IV Swingers.

The remainder of this section assumes that the default userid "pi" is used.

8.2 Utilities

The following utilities must be installed:

8.2.1 Python 2.x

Python is installed by default, but it should be noted that the IV Swinger Python code assumes Python 2.x (and has only been tested with Python 2.7). There may be compatibility issues if it is run with Python 3.x.

8.2.2 Gnuplot

Gnuplot is a plotting utility with a nearly 30-year history. The IV Swinger software uses gnuplot to generate the PDF graphs of the IV curves. It might have made more sense to use the matplotlib Python library for this purpose but I've used gnuplot for years and had no experience with matplotlib.

Gnuplot is installed with:

```
% sudo apt-get install gnuplot
```

8.2.3 USBmount

USBmount is a utility that automatically mounts USB drives when they are inserted and unmounts them when they are removed.

When the 'startx' command is used to bring up a desktop environment, USB drives are automatically mounted and unmounted as `/media/<LABEL>` when they are inserted and removed. But this is not the case when there is no desktop environment (as is the case for IV Swinger). It is important that USB drives are mounted because they are where IV Swinger writes its results.

There may be a better solution, but installing USBmount solves the problem. With USBmount installed, the mount points are named `/media/usb0`, `/media/usb1`, etc. It should be noted that the Python code doesn't depend on the USBmount nomenclature, so if a better solution is found, no changes to the code would be necessary. The software just assumes any mount points it finds under `/media` are USB drives, regardless of their names.

USBmount is installed with:

```
% sudo apt-get install usbmount
```

8.2.4 I²C Tools

The I²C Tools utility is not used by IV Swinger per se, but is very useful for bring-up and debugging. It is installed with:

```
% sudo apt-get install i2c-tools
% sudo adduser i2c pi
```

8.3 Python library code

This section lists the Python library code that must be installed in support of the IV Swinger Python module.

8.3.1 SMBUS (I²C)

The python-smbus module is required for Python support of I²C and is used by the Adafruit Python modules that control the I²C devices.

```
% sudo apt-get install python-smbus
% sudo adduser i2c pi
```

8.3.2 RPi.GPIO

The RPi.GPIO module is required for Python access to the GPIO pins. RPi.GPIO is used by the Adafruit modules as well as directly by the IV Swinger Python module.

```
% sudo apt-get install python-rpi.gpio
```

Note that the RPi.GPIO module is installed by default in current Raspbian distributions so it may not be necessary to install it with the command above.

8.3.3 NumPy

The IV Swinger Python module uses the NumPy library for some mathematical computations.

```
% sudo apt-get install python-numpy
```

8.3.4 Adafruit modules

The IV Swinger Python module uses library code from Adafruit for controlling the MCP23017, ADS1115, and LCD display devices. The Adafruit code is posted on GitHub for anyone to use. The entire Adafruit Raspberry Pi Python library can be downloaded with the following command:

```
 % git clone https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code.git
```

That will create a directory called Adafruit-Raspberry-Pi-Python-Code, and under that will be subdirectories for each of the modules. The required modules will be included as will be many others that are not needed for IV Swinger.

HOWEVER, the IV Swinger repository on GitHub includes the required Adafruit modules, so it is not necessary to download them with the command above. It is possible, however, that the versions of these modules included in the IV Swinger distribution are not the latest and greatest. On the other hand, they are the ones that have been tested with the IV Swinger code and hardware so there's some argument for using the potentially older versions.

The required Adafruit modules are:

8.3.4.1 Adafruit_I2C.py

This module contains the *Adafruit_I2C* class, which is used by the Adafruit_MCP230xx.py module and the Adafruit_*ADS1x15* .py module.

8.3.4.2 Adafruit_MCP230xx.py

This module contains the *Adafruit_MCP230XX* class, which is used by the IV Swinger Python module to control the MCP23017 port expander.

8.3.4.3 Adafruit_*ADS1x15* .py

This module contains the *ADS1x15* class, which is used by the IV Swinger Python module to access the ADS1115 ADC.

8.3.4.4 Adafruit_CharLCD.py

This module contains the *Adafruit_CharLCD* class. It is used by the IV Swinger Python code to control the 16x2 LCD display.

IMPORTANT: A modification to this file is necessary. This modification has already been made to the file included in the IV Swinger distribution. The change is to the following line:

```
  def __init__(self, pin_rs=25, pin_e=24, pins_db=[23, 17, 27, 22],
GPIO=None):
```

The third pin in the pins_db list must be 27, not 21 (this is due to a pin numbering change between the earlier Raspberry Pi models and the B+ and later models).

8.4 IV Swinger Python module

The IV_Swinger.py module controls the IV Swinger hardware, captures measurements, and displays the results graphically.

The IV_Swinger.py file contains extensive comments and docstrings to make it as "self-documenting" as possible. While this section of this document attempts to describe the software in a fair amount of depth and detail, the code itself is the ultimate documentation.

8.4.1 Importable object-oriented design

The first version of the IV_Swinger.py module was not object-oriented and could not be imported by external code. The code has now been refactored and is completely object-oriented. The main class is called *IV_Swinger*. External Python code can now import the IV_Swinger module and do all the nice things that OOP allows, such as overriding default property values, extending methods, overriding methods, etc. Now included in the GitHub distribution is a module called IV_Swinger_test.py, which shows how the IV_Swinger module can be imported and customized without altering any of the actual code.

8.4.1.1 Classes

The classes in the IV_Swinger module are:

- *PrintAndLog*: Provides printing and logging methods
- *BeepGenerator*: Generates beeps from the piezo buzzer
- *StoppableThread*; Extends *threading.Thread* to add an event for cleanly stopping the thread
- *ScrollingMessage*: Continuously scrolls a message until signaled to stop
- *SoundWarning*: Continuously sounds a warning until signaled to stop
- *Interpolator*: Creates an interpolated curve from a set of data points
- *IV_Swinger*: Main class

Most of these are discussed in more detail in subsequent sections below.

8.4.1.2 Properties

The module now makes extensive use of properties. If you are not familiar with Python properties you might want to read up on them (a Google search for "python property" will do it). Property definitions look like class method definitions, but are "decorated" with `@property` and/or `@<name>.setter`. This provides "managed access" to class attributes. In the simplest case they are just "getters" and "setters", but they can have more complex actions associated with them than just getting and setting class attributes. For example, the *Interpolator* class has properties that actually run the interpolation algorithm. From the user's point of view they just look like a class variable, i.e. they are accessed without the () that would be used for a method call. When a property is used on the left side of an equation, the "setter" functionality is invoked. When a property is used on the right side of an equation, the "getter" functionality is invoked. A typical "value added" of property "setters" is to check the provided value for proper type and other constraints.

Most of the *IV_Swinger* class properties can be thought of as "knobs" that have default values. These defaults can be overridden in any code that instantiates an *IV_Swinger* object. Many of these properties are mentioned in the following sections. In some cases the description just assumes the default value and doesn't mention the fact that it is actually a property. Section 8.4.7 lists all of the *IV_Swinger* class properties.

8.4.2 Exception handling

If an exception is detected during execution of the code, the hardware may be in a state that would be bad to leave it in. To deal with this, the ***run*** method does a minimal amount of setup and then uses a try/except with a method named ***run_meat*** called in the "try" and a method called ***clean_up*** called in the "except" branches. The ***clean_up*** method turns all of the relays off, prompts the user to turn the DPST switch off (if it is not already), displays an exit message on the LCD, and then after 10 seconds it shuts down the Raspberry Pi. It also logs the exception information in the log file. The setup code before the "try" only does what is necessary to perform the actions in the ***clean_up*** method. The ***run_meat*** method contains the rest of the program. Doing it this way (in addition to more specific exception handling sprinkled around the code) provides the most protection since there is no chance of uncaught exceptions after the setup.

The reason for the shutdown on an exception is because the users have been instructed never to shut down the IV Swinger with the power button on the battery pack – only the shutdown pushbutton. But if the program crashes, the shutdown button is no longer useful. The danger in shutting down on an exception is that a coding bug could result in a shutdown loop with no possibility of debugging or reverting to an older version of the software. The 10-second delay should be enough time to kill the process before the shutdown. An additional line of defense is that the shutdown will not occur if the pushbutton is pressed until the end of the 10-second warning period. And finally, if the pushbutton is pressed when IV_Swinger.py is starting up, it will immediately exit.

8.4.3 Multithreading

There are several things that have to be concurrent with the execution of other code:

- Monitoring the shutdown button
- Scrolling messages on the LCD display
- Sounding a warning on the piezo buzzer

These are accomplished with Python's multithreading support.

In the case of the shutdown button monitoring, the RPi.GPIO library makes this easy. The ***add_event_detect*** method registers a callback function that is invoked when a transition is seen on a GPIO pin. During the execution of the main program, there's a thread that is always "watching" the GPIO pin connected to the shutdown button so that no matter when the button is pressed, the ***pushbutton_callback*** method is called, and that method initiates the shutdown (if the button is held down for 3 seconds).

For the other two, the Python *threading* module is used. A *StoppableThread* class is borrowed from pibrella.py (https://github.com/pimoroni/pibrella). This class extends the *threading.Thread* class by adding methods to cleanly start and stop the thread. Two *StoppableThread* subclasses are defined: *ScrollingMessage* and *SoundWarning*. The former continuously scrolls a message on the LCD display

until it is signaled to stop. The latter pulses the piezo buzzer in a beeping warning pattern until signaled to stop.

The only resource shared between threads is the LCD display. The ***pushbutton_callback*** method needs to display messages on the LCD, but the main thread also displays messages on the LCD (both directly and via *ScrollingMessage* threads). To arbitrate for the LCD resource a *threading.Lock()* object is used.

8.4.4 Relay control

Each relay is controlled by one of the 16 outputs of the MCP23017 I/O extender chip on the Slice of PI/O board. The *Adafruit_MCP230XX* class makes this very easy; it hides all of the details of the I^2C transactions. The IV_Swinger code uses only two *Adafruit_MCP230XX* methods: ***config*** and ***write16***. The ***config*** method defines each of the extended I/O pins as an input or an output (and we define them all as outputs). The ***write16*** method applies a given 16-bit value to the extended I/O pins. The 16 bits correspond to the MCP23017 outputs as follows: B7:0,A7:0.

The relay control pins are active-low, so a value of 0 on an I/O extender output turns the associated relay ON and a value of 1 turns it OFF. The 16-bit value 1111111111111111 (hex 0xffff) turns all relays off.

Section 5.6 on page 60 describes the connections between the MCP23017 and the relays. Due to the layout of the Slice of PI/O board and the cabling, the B7:0 pins of the MCP23017 are connected to the RELAY_LO module pins in the opposite order from the way the A7:0 pins are connected to the RELAY_HI module pins (swizzled).

To make things less confusing for the development and debug of the main code, both the active-low nature of the relay controls and the swizzling of the RELAY_LO module connections are hidden by a wrapper method called ***set_relays_to_pattern***.

Before the inversion and swizzling, a *load_pattern* is a 16-bit quantity that has a 0 in the bit positions corresponding to relays that are to be OFF and a 1 in bit positions corresponding to relays that are to be ON. The most significant (leftmost) bit corresponds to the relay controlling the "HALF" (0.4Ω) load, and the least significant (rightmost) bit corresponds to the relay controlling the "FOURTEEN" (12Ω) load. For example, a *load_pattern* of 1000000000000000 (hex 0x8000) has only the HALF load activated and all others deactivated. A load pattern of 0000000000000001 (0x0001) has only the FOURTEEN load activated. This is pretty intuitive; the smallest load is on the left, and the largest is on the right[13].

The ***set_relays_to_pattern*** method takes a *load_pattern* as described in the previous paragraph and inverts all of the bits (to account for the active-low relay control) and swizzles the upper 8 bits (to account for the quirky wiring) and then calls the *Adafruit_MCP230XX* ***write16*** method with the resulting 16-bit value. So for example if the *load_pattern* value is 1000000000000000 (hex 0x8000), the value passed to ***write16*** is 1111111011111111 (hex 0xFEFF). This activates MCP23017 output B0, which is connected to RELAY_LO pin IN1, which is the HALF load relay. If the *load_pattern* value is

[13] On the other hand it is backwards from what you see looking at the box from the front (the HALF load is on the right and the FOURTEEN load is on the left). Another slightly confusing point is that the second relay (the one adjacent to the HALF relay) is not connected to a load, so when it is activated the circuit is OPEN (e.g. 0100000000000000 = 0x4000).

0000000000000001 (hex 0x0001), the value passed to ***write16*** is 1111111111111110 (hex 0xFFFE). This activates MCP23017 output A0, which is connected to RELAY_HI pin IN8, which is the FOURTEEN load relay.

8.4.5 Voltmeter and Ammeter measurements

As described in Section 4.2 (starting on page 49), the voltmeter and ammeter are both implemented using the ADS1115 ADC and some external circuitry. An Adafruit *ADS1x15* object is created and used to read values from the ADC. The *ADS1x15* class supports both the ADS1015 and ADS1115 parts, and this is specified at object creation. It hides all of the details of the I^2C transactions, register addresses, etc. Since the inputs are connected in the differential configuration, the *ADS1x15* ***readADCDifferential*** method is used. This method has parameters to specify the differential pin pair, the sample rate, and the gain of the programmable gain amplifier (PGA). The sample rate affects the accuracy of the measurements, but also affects the time it takes to take a measurement. The default value of 250 samples per second is currently used.

A wrapper method around ***readADCDifferential*** called ***read_adc*** is implemented in IV_Swinger.py. Its purpose is to experimentally determine the "best" value for the PGA gain and to use that value. The ***read_adc*** method first uses different PGA gain values until it finds the optimal range. The optimal range is the largest one where the reading is greater than 1/3 of the range (or the smallest range if none of the larger ranges meets this criterion). It also avoids using a range that is too small to accommodate the reading. The assumption is that the ADC is most accurate when the reading is near the middle of the range. If a reading is slightly less than 1/3 of the range, it will be less than 2/3 of the next smaller range and therefore not at risk for exceeding that range. Once the optimal range has been determined, the ***read_adc*** method takes several readings and returns the average value, in volts. This obviously exploits the assumption for this application that the voltage at the ADC is fairly stable at the times when measurements are taken.

Here's an example to help to understand the PGA range selection algorithm. Keep in mind that this occurs for every call to the ***read_adc*** wrapper method, i.e. for every measurement that is taken. In differential mode, the PGA supports ranges of: ±6144mV, ±4096mV, ±2048mV, ±1024mV, ±512mV, and ±256mV (from lowest to highest gain). Suppose the differential voltage being measured is +1234mV. The largest range, ±6144mV, is tried first. Since the measured value is less than 1/3 of the range (1234/6144 = 0.2), the next smaller range, ±4096mV, is tried. The measured value is still less than 1/3 of the range (1234/4096 = 0.3), so the next smaller range, ±2048mV, is tried. Now the measured value is greater than 1/3 of the range (1234/2048 = 0.6), so this range is deemed to be the optimal range. This is the range where the measured value is closest to the middle of the range. The two larger ranges would have worked, but their resolution would be 1/2 or 1/4 that of the chosen range. The smaller ranges are too small to accommodate the measured value.

The software then converts the voltage reading from the ADC to the actual value being measured based on the external circuitry. As described in Section 4.2.2 on page 50, the voltage between the PV cables (across the load) is divided by approximately 24 using a voltage divider circuit. This is what the ADC sees, so the software multiplies the ADC reading by 24 (the precise multiplier value is based on the actual resistor values), and that is the measured voltage for the point on the IV curve. For the current measurement, the voltage is measured across the shunt resistor. But as described in Section 4.2.3 on page 52, this voltage is too small for even the smallest PGA range (highest gain) of the ADC, so it is amplified externally with an op amp circuit before the ADC sees it. The gain of the external amplifier is approximately 56, so the software divides the ADC reading by 56 (the precise divisor value is based on

the actual resistor values), and that is the measured voltage across the shunt resistor. The software then uses Ohm's Law to translate this voltage to a current value, using the known resistance of the shunt (7.5 mΩ). This is the measured current for the point on the IV curve.

8.4.6 Program flow

Other than the exception handling and the concurrent threads described above, the code execution is fairly linear and is described sequentially in this section. As mentioned earlier, the code itself and its comments and docstrings should be consulted for the lowest level of details. Except where otherwise noted, the descriptions in this section refer to the *IV_Swinger* class properties and methods.

8.4.6.1 Initial setup

The initial setup code is in the ***run*** method before the call to ***run_meat***. As mentioned above in Section 8.4.2, the initial setup code only does what is necessary for the exception handling code to be able to run.

The first thing the initial setup code does is call the ***init_other_class_variables*** method to initialize some class variables in the supporting classes (*PrintAndLog*, *ScrollingMessage*, and *BeepGenerator*). Then it creates a logger (*PrintAndLog*) object, a beeper (*BeepGenerator*) object, the lock (*threading.Lock*) object used for inter-thread arbitration for the LCD display, and the LCD (*Adafruit_CharLCD*) object. Then the GPIO pins are initialized and the LCD is reset via calls to the ***set_up_gpio*** and ***reset_lcd*** methods. Next it creates the directory `/IV_Swinger/logs` (if it doesn't already exist), and it determines the name of the current log file based on the current date and time (e.g. `/IV_Swinger/logs/log_150401_17_39_46`). The "Welcome to IV Swinger" message is displayed on the LCD. Finally, the *Adafruit_MCP230XX* object (*io_extender*) is created and configured so that all pins are outputs.

At this point the software has control of the GPIO pins, the relays (via the I/O extender), and the LCD display and it has identified the log file. This allows it to perform the actions in the exception ***clean_up*** method as described in Section 8.4.2.

8.4.6.2 Additional setup

Once the initial setup is complete, the ***run_meat*** method is called with the protection of the try/except. Before entering the main code loop a few more setup actions are performed:

- The *ADS1x15* object (*adc*) is created
- The relays are all turned off
- The USB drives are identified

The USB drive identification deserves a bit of description. With the USBmount utility installed (see Section 8.2.3 on page 78), any USB drive that is inserted is mounted as `/media/usb<n>`. The ***find_usb_drives*** method finds all directories under `/media` and then uses *os.path.ismount* and *os.access* to filter out any that are not mount points or are write-protected. It is possible that none is found (e.g. if the user forgot to insert one or inserted one that is write-protected). In this case, a message is displayed on the LCD and the user is given 30 seconds to insert a USB drive. If and when one is inserted, it will be mounted by USBmount and will show up under `/media` and then the code will

proceed. If the 30 seconds expires, the code also proceeds but the results are written only to the SD card. In this case the software writes to the SD card a list of files that have not been copied to USB and these are copied to USB later when a USB drive is found (even if the IV Swinger is shut down in the meantime).

8.4.6.3 Main loop

The remaining steps are performed in an infinite loop (broken only by a shutdown due to idle timeout or via the shutdown pushbutton). Each iteration of the loop traces a single IV curve.

8.4.6.3.1 Make sure DPST switch is in OFF position

At the beginning of the loop, the DPST switch should be in the OFF position. The ***prompt_and_wait_for_dpst_off*** method checks that this is the case. If the switch is in the OFF position as expected, the code proceeds to the next step. If the switch in in the ON position, the piezo buzzer warning pattern is sounded and a message is displayed on the LCD instructing the user to turn the switch off. The code then sits in a loop polling the DPST switch until it is in the OFF position, at which point it proceeds.

8.4.6.3.2 Measure V_{OC} and wait for user to turn DPST switch on

When the DPST is in the OFF position the PV circuit is open. At this point the software enters a loop in which it uses the voltmeter to repeatedly measure the voltage between the PV+ and PV- cables (see Section 8.4.5 above). Since the circuit is open, this is the open-circuit voltage V_{OC}. This loop is implemented in the method called ***measure_voc***.

The ***measure_voc*** method does more than its name implies. The loop inside the method repeats while the DPST switch remains in the OFF position. Each time through the loop, the ***read_adc*** method is called to measure V_{OC}. It is also called to measure the current (which should always be zero when the DPST switch is OFF, but it is measured anyway). Since the differential mode of the ADC is used, both positive and negative voltages can be measured. If a negative V_{OC} is measured, the ***measure_voc*** method returns to the calling code immediately (see next section for more details). Otherwise, it continues to record the V_{OC} measurements, keeping a list of the most recent 5 (`voc_settle_count` property) readings. The standard deviation of these most recent readings is calculated, and if it is less than 0.01, it is concluded that the V_{OC} value is "settled", and a message is displayed on the LCD prompting the user that they may turn the DPST switch ON to begin IV curve tracing. There is no urgency at this point for the user to proceed, however. So it is possible that a long time passes before the user does in fact turn the switch on. The loop continues taking approximately ten V_{OC} measurements per second. It also continues calculating the standard deviation of the most recent five measurements. If the standard deviation is greater than 0.01 when the switch is turned on, a message is displayed on the LCD warning that the results may be unreliable, but it proceeds. This loop in the ***measure_voc*** method also performs the idle timeout detection. If the user has finished taking measurements, but hasn't pressed the shutdown button, this is where the code will be executing. In order to preserve the battery, a shutdown is performed if this loop executes for 600 seconds (`idle_timeout_seconds` property) without the DPST switch being turned ON. But 30 seconds (`idle_timeout_warning_seconds` property) before the timeout expires, the user is warned with beeping and a countdown on the LCD display. There is no way for the user to extend the timeout other than to turn the DPST switch ON and generate an IV

curve. Even if the user isn't ready for the next measurement, at least he/she doesn't have to wait through another reboot.

When the DPST switch is turned on, the loop in the ***measure_voc*** method terminates, and the last measured V_{OC} value is returned to the calling code (in ***run_meat***).

8.4.6.3.3 Check for incorrect PV connection

If the ***measure_voc*** method returns a negative V_{OC} value, a message is displayed on the LCD informing the user that the PV cables are connected backwards and must be disconnected immediately to avoid damage. The much simpler ***read_voc*** method is called in a loop until the voltage is no longer negative; the piezo buzzer is used to generate a loud beep each time through the loop to alert the user to the urgent situation. When the voltage is no longer negative, the loop exits and the code goes back to the beginning of the main loop.

If the ***measure_voc*** method returns a V_{OC} value of 0 volts, it is an indication that no PV is connected. In this case, the code in ***run_meat*** displays a message to this effect on the LCD display along with piezo warning beeps to get the user's attention. It then enters a loop polling the voltage with ***read_voc*** until it sees a non-zero V_{OC} value. At that point the code goes back to the beginning of the main loop.

In the case where the PV cables are connected backwards, the user obviously cannot instantaneously reverse them, so they will see both of the above warnings with the latter occurring when they are in the process of swapping the cable connections.

8.4.6.3.4 Swing the IV curve

When the measured V_{OC} is positive and the DPST switch is turned ON, the process of swinging the IV curve begins. At this instant, the ***run_meat*** method captures the current date and time and calls the ***swing_iv_curve*** method.

The ***swing_iv_curve*** method cycles through the load values using the relays, taking a current and voltage measurement at each point. The results are returned in a list of 4-entry tuples (amps, volts, ohms, watts).

The basic algorithm is very simple. As described in Section 8.4.4 on page 83, a *load_pattern* is a 16-bit vector that maps to the 16 relays with a 0 meaning the corresponding relay is inactive and a 1 meaning the corresponding relay is active. There are three hardcoded lists of *load_pattern* values that are used by the ***swing_iv_curve*** method: `DIAG_LOAD_LIST`, `FINE_LOAD_LIST`, and `COARSE_LOAD_LIST`. Depending on property values, one of these is chosen and stepped through from beginning to end. The *load_pattern* is applied to the relays, the measurements are taken, and then this is repeated for the next *load_pattern* in the chosen list until the end of the list is reached.

If the `diag_mode` property is set, the `DIAG_LOAD_LIST` is used. Each of the *load_pattern* vectors in `DIAG_LOAD_LIST` is "one hot", i.e. only one bit is a 1 and all others are zeros. Therefore each relay is activated individually and the I and V values are captured with just that one relay active and all the others inactive. This mode is not for generating an IV curve, but is useful for diagnosing problems such as bad relays, burnt out loads, etc. The `diag_mode` property defaults to `False` - the only way to run

in this mode is to override the default either by editing the code in IV_Swinger.py or by importing the module and overriding it externally; there is no way to do it from the standalone box.

If the `fine_mode` property is set, the `FINE_LOAD_LIST` is used. This one contains all of the half and full steps. This wears out the HALF relay faster than the others, but skipping the half steps on parts of the curve that are relatively straight lines mitigates this. If the `fine_mode` property is not set, the `COARSE_LOAD_LIST` is used. This one contains only full steps (starting with HALF). The `fine_mode` property is currently set to `True` in IV_Swinger.py; the only way to run in coarse mode is to override the default either by editing the code in IV_Swinger.py or by importing the module and overriding it externally. However, there is another property, `fine_mode_toggle` that can be set to `True` to enable toggling between fine and coarse modes at run time with the shutdown button. But since this property defaults to `False`, a software change is still required to enable the toggling feature. The `fine_mode` and `fine_mode_toggle` properties predate the algorithm to skip the half steps on the straight parts of the curve, which achieved a very good compromise between the curve resolution and relay wear. There is probably no real need for coarse mode anymore but the code to support it remains.

In all three modes, the first *load_pattern* in the list is the value NONE, which is a vector of all zeros (all relays inactive). This is the closest to a short circuit that is possible since all of the loads are bypassed. However there is still a small amount of resistance even when all of the relays are inactive; this is the resistance of the wires, solder joints, printed traces on the relay modules, and the relay contacts. Therefore the voltage measured with the NONE *load_pattern* is small but not zero, and the current is slightly less than the actual I_{SC}. Section 8.4.6.3.6 below describes how the I_{SC} value is extrapolated from this point and the point after it.

The algorithm to skip half steps in `fine_mode` requires more explanation. This algorithm is based on the observation that most of a typical IV curve consists of straight lines: the nearly horizontal one before the knee and the nearly vertical one after the knee. An IV curve generated using the coarse mode looks nearly as good as one generated using fine mode because the straight lines get no added value from the half steps. It is only the knee of the curve itself that benefits from the half steps. Since the half steps wear out the HALF relay, we would like to only use half steps when they add value. The solution is to dynamically determine when a half step will add value, and only do it if it does. The `FINE_LOAD_LIST` contains all of the full and half steps. Instead of using each *load_pattern* in the list sequentially, the code skips every other one. When it has taken a measurement, it uses some math to determine if the curve is inflecting. It needs three points to determine this: the current one and the previous two. The details of the math are in the code, but the gist of it is that the angle between the two lines connecting the two pairs of points is calculated, and the distance between the two points is calculated. The product of the angle and the distance is compared with a threshold and if it is smaller than that threshold, the half step is skipped. If the three points are literally in a straight line, the angle is 0, and the half step is skipped regardless of the distance between the two points. If the points are very close together already, adding the half step between them doesn't add value even if the angle is somewhat large, so the half step is also skipped in that case. The catch is that once we've determined that a half step would be beneficial, we've already gone past that point on the curve, so to get that point we have to go backwards. This is achieved by putting the *load_pattern* vectors in the `FINE_LOAD_LIST` out of order. For example, the FOUR *load_pattern* comes after the FOUR_AND_A_HALF *load_pattern*. If after the FOUR_AND_A_HALF measurement is taken, the curve is determined not to be inflecting, then the FOUR measurement is skipped. But if the curve is inflecting, then the FOUR measurement is taken after the FOUR_AND_HALF measurement, effectively backing up a half step. Since the measurements are taken out of order, they need to be sorted (by load resistance) before the results are returned to the caller. This algorithm works very well. A typical IV

curve only requires two or three half step data points on its knee. But for more unusual curves such as shading cases, the algorithm does not break down; it applies the half steps in all the places where there are inflections in the curve, even if there are multiple knees (see Figure 13-1 on page 117).

There is one more enhancement to the basic algorithm. This is needed for the case where the IV curve is very "wide and low". This is the case for any panel when there isn't much sun. The voltage at the MPP (knee) is the same as in full sun, but the current at the MPP is less - in proportion to the insolation. The load resistance at the MPP in 1/2 sun is double the load resistance at the MPP in full sun. This problem was described way back on page 21 and illustrated in Figure 3-3. That figure shows how the 1/2 sun case has no points over the knee of the curve. We'd like to be able to generate decent IV curves for low insolation cases. If the maximum load doesn't reach over the knee, there's nothing that can be done. This is why the IV Swinger load bank has the 3Ω, 6Ω, and 12Ω power resistor loads at the end of the chain; these loads have more resistance than all of the other loads combined. This means we will indeed have points past the knee even for curves that are "wide and low". That's good, but there is still a problem. These last loads are very "coarse", so we'll end up with very poor resolution on the knee with the fine resolution loads "wasted" on the earlier flat top of the curve.

The solution to this is to use one or more of the big loads early, and then add the finer grained loads later. It doesn't matter that the big loads are physically at the end of the chain. An adaptive algorithm is used to determine a 'base load', i.e. one or more of the power resistors. The goal is to center the fine-grained part of the load chain at the knee of the curve. With the 3Ω, 6Ω, and 12Ω resistors the following base load values are possible: 0Ω, 3Ω, 6Ω, 9Ω, 12Ω, 15Ω, 18Ω, and 21Ω. The ***`get_base_loads`*** method determines the optimal base load value. This method is called immediately after the first measurement is taken when the DPST is turned on. That first measurement is for the NONE *load_pattern*, which as described above is the closest to a short circuit that is possible. The current measured with the NONE *load_pattern* is approximately I_{SC}. This is used with the V_{OC} value to estimate the load resistance at the knee of the curve: V_{OC}/I_{SC}.[14] The ideal base load is the estimated knee load minus the resistance of half of the unit load chain. Half of the unit load chain is SIX+HALF, which (from empirical data) is 5.6 ohms (average 0.86 ohms per load). The base load that is closest to the ideal base load is chosen. There's one additional condition, however. Since the power resistors are only rated at 50W it is possible to burn them out if the current is too high. This is not an issue if they are added to the chain at the end of the IV curve because the current is low by that point in the curve. But when they are used as a "base load" they are added first, when the current is at its highest (close to I_{SC}). To protect from overdriving the power resistors, the I_{SC} value is checked against upper limits. The code assumes that the 50W resistors can handle at least 100W for a couple seconds. The 3Ω load is two in parallel, so it can handle 200W. The 12Ω load is two in series, so it can also handle 200W. Since power is I^2R, here are the current limits for each of these loads:

- 3Ω: $\sqrt{\frac{200W}{3\Omega}} \approx 8A$
- 6Ω: $\sqrt{\frac{100W}{6\Omega}} \approx 4A$
- 12Ω: $\sqrt{\frac{200W}{12\Omega}} \approx 4A$

If I_{SC} is greater than 8A, the code chooses not to use a base load even if the ideal base load would otherwise be 3Ω or more. And if I_{SC} is between 4A and 8A, only the 3Ω base load is used even if the

[14] Note that this assumes a typical unshaded IV curve. If there is shading, it is possible that the fine grain resolution would be more valuable near the beginning of the curve. You can't win 'em all.

ideal base load would otherwise be 6Ω or more. But remember that the whole purpose of the base load algorithm is to handle the case where the IV curve is "wide and low", which means I_{SC} is low. So it is unlikely that these current limits will come into play in the cases where a base load is advantageous.

The ***get_base_loads*** method returns a list of 1-hot load vectors that correspond to the power resistor(s) that comprise the selected base load (or possibly the NONE vector if no base load is to be used). The base load can consist of 0, 1, 2, or 3 of the power resistor loads. In the cases where 2 or 3 are used, the code in ***swing_iv_curve*** activates them one at a time and takes a measurement at each of these points. Then it starts taking measurements using the *load_pattern* values in the normal list (e.g. `FINE_LOAD_LIST`), but ORing them with the base load vector. Since this will result in some redundant measurements, it checks each vector first to see if it has already taken a measurement with that *load_pattern* and skips it if it has.

The last thing that the ***swing_iv_curve*** method does is call the ***open_circuit*** method. The second relay from the right on RELAY_LO (controlled by pin IN2, connected to Slice of PI/O pin B1) is not connected to any load, and is designated the OPEN relay. This was a late design change to improve the lifetime of the relay contacts; as shown in Figure 3-13 on page 39 there is actually an immersion coil associated with this relay that is simply not connected to the relay. When this relay is activated, the load circuit is opened, even though the DPST switch is still in the ON position. The ***open_circuit*** method first activates the OPEN relay to open the circuit, while keeping the other relays as they were. Once the OPEN relay is activated (and after a short delay), the other relays are inactivated. The purpose of this is to avoid inactivating the relays while current is flowing. This completely eliminates any arcing on that side of the relay (the Normally Open, "NO" side), which is the side without a snubber. It also means that there is no current flowing when the DPST is turned OFF (opened), so there will also be no arcing across the DPST contacts.

8.4.6.3.4.1 Taking the measurements

For each data point being measured, the ***swing_iv_curve*** method calls the ***get_data_values_for_load_pattern*** method, passing it the *load_pattern*. This method sets the MCP23017 outputs (controlling the relays) to the appropriate values based on the specified load pattern and reads the current and voltage at that point. Ohms and watts are calculated and the four values are returned in a tuple.

The ***set_relays_to_pattern*** method is called to activate the appropriate relays based on the *load_pattern* (see Section 8.4.4 on page 83).

After a short delay, the current is measured first, and then the voltage is measured. See Section 8.4.5 on page 84.

The resistance in ohms and power in watts are calculated using the measured I and V values (V/I and V*I respectively).

All four values are written to the log file and are displayed on the LCD. It is necessary to reset the LCD before the values are displayed because the switching of the relays causes transients that mess up the LCD display.

8.4.6.3.5 Prompt and wait for user to turn DPST off

After the ***swing_iv_curve*** has opened the circuit and returned the sorted list of measured data points, the ***run_meat*** method calls the ***prompt_and_wait_for_dpst_off*** method, which was also called at the beginning of the main loop and is described in Section 8.4.6.3.1 on page 86.

8.4.6.3.6 Turn relays off and clean up LCD

The ***prompt_and_wait_for_dpst_off*** method returns when it detects that the user has turned the DPST switch off. At this point the circuit is open in two places: the OPEN relay and the DPST switch. In preparation for the next run (i.e. iteration of the main loop) all of the relays are inactivated[15]. The LCD display is also reset to clean it up from the relay-switching transients.

8.4.6.3.7 Analyze and format data

Some analysis of the measured data is necessary at this point.

The first step is to find the measured point with the highest power. The ***get_max_watt_point_number*** method does this. It simply steps through the list of measured data points and identifies the one with the highest "watts" value. The actual Maximum Power Point (MPP) is most likely not exactly at this point, but somewhere between this point and one of its neighbors and will be found later via interpolation. The ***get_max_watt_point_number*** method does not return the power value at the data point; it returns the index of the point in the list of measured data points.

Next the I_{SC} value is extrapolated. As discussed earlier, the voltage measured with the NONE *load_pattern* is small but not zero, and the current is slightly less than the actual I_{SC}. The ***extrapolate_isc*** method operates on the assumption that a straight line between the NONE data point and the next data point will intersect the I-axis at the true I_{SC} point. Simple high school math (y=mx+b, or in this case i=mv+b) is used to calculate the extrapolated I_{SC} value. A typical IV curve slopes down only slightly at the beginning of the curve so the extrapolated I_{SC} value should be no more than 2% greater than the measured current of the NONE data point. If the calculated value is greater than this, it indicates something wrong in the assumptions. One such case would be where a base load is used but there is also shading. This could cause the first data point after NONE to already be past the first knee, and therefore not in the same line as the true I_{SC} point and the NONE point. In this case, ***extrapolate_isc*** will return a value that is 2% greater than the NONE point current. Another indication that the assumptions were violated is if the measured point with the highest power is one of the first three measured points. That also indicates that the point after the NONE point may already be heading over the knee. In that case, the method returns the current value of the NONE point.

The extrapolated I_{SC} data point is added to the beginning of the list of measured data points. More accurately, it is written over a placeholder data point that the ***swing_iv_curve*** method left in that position.

Next, the V_{OC} data point (that was captured before the DPST switch was turned on) is appended to the end of the data point list.

[15] If you're reading carefully, you'll note that all of the relays except for the OPEN relay were already inactivated at the end of ***swing_iv_curve***, so all this really does is inactivate the OPEN relay.

8.4.6.3.8 Write data to SD card

The ***run_meat*** method next creates a directory for the output files under `/IV_Swinger` on the Raspberry Pi SD card. This directory is named for the date and time at which the DPST switch was turned ON by the user for this run (e.g. `150401_12_08_56`). It then creates names for the four files that it will write to this directory: the CSV data points file, the PDF file, a gnuplot command file, and a gnuplot data points file. The file names contain the same timestamp string as the directory name. This may seem redundant, but it is useful when the file is copied somewhere else.

The CSV file is written first. The ***write_csv_data_points_to_file*** method does this. The CSV file is very simple. The first line contains the column headers: Volts, Amps, Watts, Ohms. Each subsequent line contains the comma-separated values for each data point in order from the I_{SC} point to the V_{OC} point.

The gnuplot data points file is written next. This is the file provided as input to the gnuplot utility to generate the PDF graph. The ***write_gp_data_points_to_file*** method does this. There are no comments or column headers in this file. Starting on the first line, each line consists of three numerical values: volts, amps, and watts. A single space character separates the values. The values are written with 3 places after the decimal point.

8.4.6.3.9 Generate interpolated data and find MPP

As mentioned above in Section 8.4.6.3.7, the data point with the highest I*V product is not necessarily the actual Maximum Power Point (MPP) of the IV curve. The actual MPP is most likely somewhere between the data points preceding and following that one (but could be at a different "knee" in shading cases where there are multiple knees). To find the true MPP, we need to interpolate the curve as it passes through all the measured points and find where the power is highest on that curve. Since the MPP is at a knee, the IV curve really is curving at this point and it would be nice to find an interpolation function for that curve rather than using a straight line. Furthermore, once such a function has been found, it can be used to draw the graph more accurately. Stated another way, we want to interpolate and graph similar to the way Excel does when a chart is generated using "Smooth Marked Scatter" rather than using "Straight Marked Scatter".

I spent a disproportionate amount of time trying to get this to work, but ended up falling back to linear interpolation in the first version of the code (and for the first year of the IV Swinger's existence). Cubic spline interpolation had seemed to be the way to go, and it does work perfectly in many cases. But when it goes wrong, it goes very wrong. The failure is usually of the nature that the interpolated curve "oscillates" through the data points, i.e. there are places where the curve goes too low and has to go upward to pass through the next point. One thing we know about IV curves is that each point has an I value that is less than or equal to its predecessor's I value and a V value that is greater than or equal to its predecessor's V value; i.e. as you move from left to right, the curve should never go "up", only down or at most, flat. The technical term for this is "monotonicity", and cubic splines are not guaranteed to preserve the monotonicity of the input data set. There are, however, splines that do preserve monotonicity. See https://en.wikipedia.org/wiki/Monotone_cubic_interpolation for more info (and a graph that is a perfect illustration of the problem).

According to some sources on the web, Excel uses a 3rd order Bézier spline for its "Smooth" curve fitting, but others suggested Catmull-Rom. It turns out that Catmull-Rom is excellent for this (more specifically "centripetal Catmull-Rom"). It is not monotonic (without modification), but it does not oscillate, and the overshoot is negligible. The *scipy.interpolate* module does not have support for

Catmull-Rom interpolation. But the Wikipedia article on Catmull-Rom (https://en.wikipedia.org/wiki/Centripetal_Catmull-Rom_spline) has a full Python implementation from which I borrowed liberally. The IV Swinger code no longer uses *scipy.interpolate* at all.

The *Interpolator* class performs the interpolation services for, and is instantiated by, the *IV_Swinger* class. It supports both linear and Catmull-Rom interpolation, although the linear interpolation is currently unused. It operates on a given set of data points in the form of a list of (I, V, R, P) tuples provided at object creation time. The measured data points (plus the extrapolated I_{SC} data point) are contained in this format in the `data_points` list in the *IV_Swinger* class, and this is the list that is passed when it instantiates the *Interpolator* object. The *Interpolator* class provides two basic services: it generates a set of interpolated data points that include the given data points, and it identifies the data point with the maximum power (MPP). No interpolation is performed until one of the *Interpolator* properties is accessed. The *Interpolator* properties are:

- The ***linear_interpolated_curve*** property returns a list of tuples containing the initial set of points and all of the linearly-interpolated points
- The ***spline_interpolated_curve*** property returns a list of tuples containing the initial set of points and all of the spline-interpolated points
- The ***linear_interpolated_mpp*** property returns the (I, V, R, P) tuple of the maximum power point on the linearly-interpolated curve
- The ***spline_interpolated_mpp*** property returns the (I, V, R, P) tuple of the maximum power point on the spline-interpolated curve
- The ***num_interp_points*** property can be used to set/get the number of points that will be interpolated between each of the given points. Its default is 100.

The interpolated curve properties use only the I and V values from the given points to perform the interpolation but calculate the R and P values for each interpolated point to complete the (I, V, R, P) tuple. The interpolated MPP properties search through the interpolated points looking for the one with the highest P value.

The ***linear_interpolated_curve*** property (currently not used by default by the *IV_Swinger* class) simply uses *numpy.linspace* to separately interpolate the I values and the V values between each pair of the given points. It then combines the interpolated I values with the interpolated V values and calculates the R and P values to generate each interpolated (I, V, R, P) tuple. The resulting list is stored in an instance attribute and a copy of that attribute is returned when the property is referenced. If the property is referenced again, the interpolation is not re-run; the existing populated attribute is used. The ***linear_interpolated_mpp*** property searches through the list of linear interpolated points and it identifies the one with the highest power, which it stores in an instance attribute and a copy of that attribute is returned when the property is referenced. If the property is referenced again, the search is not re-run; the existing populated attribute is used.

For the spline interpolation, two methods are based heavily on the Wikipedia code: ***catmull_rom_spline*** and ***catmull_rom_chain***.

The ***catmull_rom_spline*** method takes four points as input. These points are [x, y] coordinate pairs (which in our case are [V, I] pairs). It also has a parameter to specify how many interpolated points to generate. The method performs the Catmull-Rom math and returns the list of interpolated [x, y] points starting from the second given point and ending at the third given point (inclusive of both). The value of "alpha" is 0.5 for a "centripetal" Catmull-Rom spline, and that is what is used initially. The code is

modified from the Wikipedia code, however, to improve on the results following a sudden inflection such as the one found in a shading case. The centripetal Catmull-Rom curve tends to overshoot after the sudden inflection and then overcorrect before settling at the next point. There can be other cases where the centripetal Catmull-Rom curve overshoots or undershoots between the two end points. The code modification is to detect these cases, and rerun the interpolation with a value of 0.1 for alpha. After the alpha=0.5 run, the interpolated points are analyzed to see if there are any adjacent points where either the x values or the y values are changing in the opposite direction from the change in the x or y values of the endpoints. i.e. it is looking for cases where the interpolated curve is "going up" when the linear interpolation is "going down", etc. For a normal IV curve, the x values (voltage) should increase and the y values (current) should decrease at each point along the curve (i.e. the curve is always going "down and to the right"). But the algorithm is generalized to detect any difference in direction (up/down/left/right). In the shading case there is a sudden inflection where the curve switches from going nearly straight down, to going nearly straight to the right (see Figure 13-1 on page 117). The interpolated curve after this inflection starts out going down (as expected), but then reaches a bottom, and starts going back up. This is detected as a difference in direction from the linear interpolation between the two points. In this case, the method is called recursively with a `rerun_with_low_alpha` parameter and the interpolation is performed with alpha=0.1. This does not completely eliminate the overshoot and overcorrection, but it dampens it. Empirically, the results with this modification appear to be more true to what we know about IV curves. Future versions of the code may improve on this. In fact, it is possible that reverting to linear interpolation for segments such as this would be the best solution. But it is just a bit "ugly" mixing linear interpolation with the spline interpolation.

The ***catmull_rom_chain*** method is directly ported from the Wikipedia code. It simply calls the ***catmull_rom_spline*** method for each group of four points in the input list to generate a combined curve. The "catch" is that this combined curve begins at the second point in the input list and ends at the second-to-last point.

The **spline_interpolated_curve** property calls the ***catmull_rom_chain*** method to generate the interpolated curve from the given points. But since the curve returned by the ***catmull_rom_chain*** method does not include the first or last points, the **spline_interpolated_curve** property inserts "dummy" points in the list that it passes to the ***catmull_rom_chain*** method. These dummy points are nearly identical to the first and last given points. The first dummy point is 1/1000 of the way between the first and second given point and the other dummy point is 999/1000 of the way between the second-to-last and last given point. The given points in the *Interpolator* class are (I, V, R, P) tuples, but the ***catmull_rom_chain*** method takes [x, y] pairs, so the **spline_interpolated_curve** property creates a `list_with_dummies` of [V, I] pairs from the given points with the [V, I] pairs for the two dummy points added in the second and second-to-last positions. It calls the ***catmull_rom_chain*** method with `list_with_dummies` and then it takes the much larger list of [V, I] pairs returned by that method and calculates the R and P values for each point to generate the complete interpolated list of (I, V, R, P) tuples. This complete list of interpolated points is stored in an instance attribute and a copy of that attribute is returned when the **spline_interpolated_curve** property is referenced. If the property is referenced again, the interpolation is not re-run; the existing populated attribute is used. The **spline_interpolated_mpp** property searches through the list of spline interpolated points and it identifies the one with the highest power, which it stores in an instance attribute and a copy of that attribute is returned when the property is referenced. If the property is referenced again, the search is not re-run; the existing populated attribute is used.

The *IV_Swinger* class (in the **`main_meat`** method) uses the Interpolator class as follows:

- Instantiates an *Interpolator* object using `data_points` as the given points
- Gets the list of interpolated points from its **`spline_interpolated_curve`** property
- Gets the MPP point from its **`spline_interpolated_mpp`** property

The *IV_Swinger* class has a **`use_spline_interpolation`** property that defaults to *True*. If its value is set to *False*, the linear *Interpolator* properties will be used instead.

The interpolated points are then appended to the gnuplot data points file. This is the file provided as input to the gnuplot utility to generate the PDF graph, and the original `data_points` list was already written to it as mentioned in section 8.4.6.3.8. The interpolated points are added as a second data set in the file. The ***`write_gp_data_points_to_file`*** method is called with the `new_data_set` parameter set to *True*. The interpolated data set is appended to the gnuplot data points file after two blank lines (which is how data sets are delimited in a gnuplot data input file).

The interpolated MPP value is displayed in a message on the LCD.

8.4.6.3.10 Generate PDF graph with gnuplot

The gnuplot utility can be run interactively or in batch mode. We obviously need to use the latter, in which case it takes a file with commands as its command line argument. The method ***`plot_with_gnuplot`*** calls a method ***`write_gnuplot_file`*** to create the command file and then invokes gnuplot. The ***`write_gnuplot_file`*** method creates a specific command file for the particular run's plot. It includes commands to set the output mode to PDF (and specify the output file name), set up the titles, set the axis scales and configure grid lines. It also adds commands to put labels on the I_{SC}, MPP, and V_{OC} points. Finally, it adds the "plot" command that generates the actual graph using the data points file for this run. The resulting PDF file will be in the run's directory under `/IV_Swinger` on the SD card, along with the CSV file and the gnuplot data and command files.

8.4.6.3.11 Copy CSV and PDF files in flat directories

Next, the CSV and PDF files are copied to the `/IV_Swinger/csv` and `/IV_Swinger/pdf` directories. Since the files contain the date and time in their names, it is still possible to identify them in this flat organization. The purpose of this is simply as a convenience. For example, it is easy to flip through all of the PDFs with a viewer if they are in one directory (and potentially even see them as thumbnails).

8.4.6.3.12 Copy files to USB drive(s)

At this point, the result files and log file have been written to the SD card under `/IV_Swinger` but they have not been written to the USB drive(s). The ***`run_meat`*** method calls the ***`copy_files_to_usb`*** method to do this. The ***`find_usb_drives`*** method is called at the beginning of the ***`copy_files_to_usb`*** method even though it was previously called before the beginning of the main loop (see Section 8.4.6.2 on page 85). This allows for the possibility that one or more USB drives have been inserted (or removed) since that time. Each found USB drive is also checked to make sure it has at least 1 million bytes of free space available. If no USB drive with enough space is found, the date/time string is added to the file `/IV_Swinger/pending_usb_copy`. If one

or more USB drives (with enough space) are found this time, any files from previous runs that were never copied to USB (i.e. those listed in `/IV_Swinger/pending_usb_copy`) are copied now - in addition to the files for the current run.

The ***`copy_files_to_usb`*** method creates the `/IV_Swinger` directory on the USB drive and creates `logs`, `csv`, and `pdf` directories under `/IV_Swinger`. It copies the current run's log file to `/IV_Swinger/logs`. It then copies the whole directory with the output files to a directory of the same name under `/IV_Swinger` on the USB drive. Finally, it copies the CSV and PDF files from the run to the `/IV_Swinger/csv` and `/IV_Swinger/pdf` directories on the USB drive. This is done for each USB drive that was identified and had enough space.

8.4.6.3.13 Display final messages and repeat main loop

The last thing the main loop of the ***`run_meat`*** method does is display two alternating messages on the LCD display. The first of these messages informs the user of the directory name containing the results for the just-completed iteration (i.e. the date/time string). The other message instructs the user to turn the DPST switch ON to begin the next test.

At this point, the main loop iteration is complete, and the steps described starting in Section 8.4.6.3.1 on page 86 begin again. The messages described in the previous paragraph continue to be displayed on the LCD while the V_{OC} value is measured and the code waits for the user to turn the DPST switch on again.

8.4.7 *IV_Swinger* class properties

Table 1 below lists the properties of the *IV_Swinger* class, their descriptions, and their default values. The default behavior of an *IV_Swinger* object can be changed by assigning different values to one or more of these properties after the object is instantiated. A property may be read by assigning its value to a variable. The properties labeled READ ONLY are derived from other properties' values and may not be set directly.

Property name	Property Description	Property Default
shutdown_on_exit	If True, shut down the Raspberry Pi on an exception	True
fine_mode	Fine mode produces better results but wears out the HALF relay faster. However, with the adaptive algorithm that only adds the half steps where the curve is bending, most runs only add two extra points in fine mode.	True
fine_mode_toggle	If *fine_mode_toggle* is True, the pushbutton can be used to toggle the mode.	False

Property name	Property Description	Property Default
loads_to_skip	If *loads_to_skip* is non-zero, the first N load_pattern values in COARSE_LOAD_LIST or FINE_LOAD_LIST are skipped. This mode is for testing with small power supplies that have short-circuit protection that triggers when small load values are used.	0
root_dir	Root directory on SD card where results and log files are written	"/IV/Swinger"
logs_dir *(READ ONLY)*	Directory on SD card where log files are written	*<root_dir>/logs*
diag_mode	When true, this property causes the test to run in diagnostic mode	False
headless_mode	When true, this property causes the test to run in headless mode	True
idle_timeout_seconds	Amount of idle time in seconds before a shutdown is initiated	600
idle_timeout_warning_seconds	Number of seconds before the idle timeout that a warning is issued	30
dpst_gpio	GPIO pin (BCM numbering) that the DPST is connected to	4
button_gpio	GPIO pin (BCM numbering) that the pushbutton is connected to	5
buzzer_gpio	GPIO pin (BCM numbering) that the piezo buzzer is connected to	18
button_time_for_shutdown	Amount of time in seconds that the pushbutton must be held down in order for a shutdown to be initiated	3
lcd_lines	Number of lines on the LCD	2
lcd_disp_chars_per_line	Number of characters per line on the LCD display	16
lcd_mem_chars_per_line	Number of characters per LCD line that can be held in memory	40
lcd_chars_per_scroll	Number of characters to scroll on the LCD when displaying a scrolling message. Must be integer divisor of 24 (1,2,3,4,6,8,12)	4
lcd_scroll_delay	Time in seconds to delay between each LCD scroll	1.2
mcp23017_i2c_addr	Hex I2C address that the MCP23017 is jumpered to	0x20
time_between_measurements	Time in seconds to delay between taking measurements	0.05
samples_per_measurement	Number of samples to take at each measurement point	2

Property name	Property Description	Property Default
max_retries	Number of times to retry failed measurements	0
voc_settle_count	Number of Voc measurements to keep when determining if the value has settled	5
sps	Samples per second with which to program the ADC. Must be one of: 8, 16, 64, 128, 250, 475, 860	250
vdiv_r1	Resistance in ohms of voltage divider resistor R1	180000.0
vdiv_r2	Resistance in ohms of voltage divider resistor R2	8200.0
vdiv_r3	Resistance in ohms of voltage divider resistor R3	5600.0
vdiv_mult (READ ONLY)	Amount that voltage divider Vout is multiplied to determine Vin	*(vdiv_r1 + vdiv_r2 + vdiv_r3) / vdiv_r2*
vdiv_chp	ADC channel connected to positive side of the voltage divider	0
vdiv_chn	ADC channel connected to negative side of the voltage divider	1
amm_op_amp_rf	Resistance in ohms of ammeter op amp resistor Rf	82000.0
amm_op_amp_rg	Resistance in ohms of ammeter op amp resistor Rg	1500.0
amm_op_amp_gain (READ ONLY)	Gain of ammeter op amp circuit	1 + (*amm_op_amp_rf / amm_op_amp_rg*)
amm_shunt_max_volts	Maximum voltage across ammeter shunt resistor	0.075
amm_shunt_max_amps	Maximum current through ammeter shunt resistor	10
amm_shunt_resistance (READ ONLY)	Resistance of shunt resistor	*amm_shunt_max_volts/ amm_shunt_max_amps*
amm_chp	ADC channel connected to positive side of the ammeter shunt resistor	2
amm_chn	ADC channel connected to negative side of the ammeter shunt resistor	3
plot_ideal_curve	If True, an ideal IV curve based on the Isc and Voc is plotted	False
plot_interpolated_curve	If True, the interpolated IV curve is plotted	True
use_spline_interpolation	If True, spline interpolation is used. If False, linear interpolation is used.	True

Table 1: IV_Swinger class properties

The main() function in IV_Swinger.py is a simple example of instantiating an IV_Swinger object and overriding some of its property defaults:

```
def main():
    """Main function"""
    ivs = IV_Swinger()
    # Override default property values
    ivs.vdiv_r1 = 178900.0  # tweaked based on DMM measurement
    ivs.vdiv_r2 = 8200.0    # tweaked based on DMM measurement
    ivs.vdiv_r3 = 5595.0    # tweaked based on DMM measurement
    ivs.amm_op_amp_rf = 82100.0  # tweaked based on DMM measurement
    ivs.amm_op_amp_rg = 1499.0   # tweaked based on DMM measurement
    ivs.run()
```

9 Raspberry Pi Configuration

The IV Swinger and Adafruit Python code must be installed manually (or with git). Sections 8.2, and 8.3 on pages 78-80 include instructions on how to install the supporting utilities and libraries. This section describes where to put the code from the IV Swinger distribution. Additionally there are some system files that need to be modified in order for the IV Swinger to use the I^2C bus and to run without a display, keyboard, mouse and Internet connection.

NOTE: It is assumed that there is a non-root userid "pi" on the Raspberry Pi, as is customary.

9.1 IV Swinger code

The IV Swinger and Adafruit code must be installed in the following locations:

```
/home/pi/IV_Swinger/python/IV_Swinger.py
/home/pi/IV_Swinger/python/Adafruit_ADS1x15.py
/home/pi/IV_Swinger/python/Adafruit_CharLCD.py
/home/pi/IV_Swinger/python/Adafruit_I2C.py
/home/pi/IV_Swinger/python/Adafruit_MCP230xx.py
```

No copying is necessary if the IV_Swinger GitHub repo is cloned on the Raspberry Pi as follows:

```
% cd /home/pi
% git clone https://github.com/csatt/IV_Swinger.git
```

9.2 System file modifications

Some system files must be modified as specified in this section.

9.2.1 /etc/modules

The `/etc/modules` file must be edited to add any of the following three lines that it does not already include:

```
i2c-bcm2708
i2c-dev
rtc-ds1307
```

Use `sudoedit` (or other editor under `sudo`) to edit this file.

9.2.2 /etc/modprobe.d/raspi-blacklist.conf

The `/etc/modprobe.d/raspi-blacklist.conf` file (if it exists) must be edited to comment out the blacklisting of spi-bcm2708 and i2c-bcm2708. Just add the # character at the beginning of each line:

```
#blacklist spi-bcm2708
#blacklist i2c-bcm2708
```

Use `sudoedit` (or other editor under `sudo`) to edit this file.

9.2.3 /etc/rc.local

The `/etc/rc.local` file must be edited to add the following lines BEFORE the existing "`exit 0`" line at the end of the file:

```
# Initialize time of day from DS1307 RTC
echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-1/new_device
hwclock -s

(python /home/pi/IV_Swinger/python/IV_Swinger.py)&
```

Use `sudoedit` (or other editor under `sudo`) to edit this file.

This file is executed automatically when the Raspberry Pi boots (it is run by the "root" superuser). The first added lines are to set the system time from the DS1307 battery-backed real-time clock (see Section 5.9 on page 69).

The last line is to run the IV Swinger python code. If the IV_Swinger module is imported by another module (e.g. to override properties such as the resistor values), this line should point to that file instead.

9.2.4 /boot/config.txt

According to https://learn.adafruit.com/adafruits-raspberry-pi-lesson-4-gpio-setup/configuring-i2c the following lines must be added to `/boot/config.txt` if the kernel is 3.18 or higher (which was not the case for me):

```
dtparam=i2c1=on
dtparam=i2c_arm=on
```

Use `sudoedit` (or other editor under `sudo`) to edit this file.

10 Development Testing

Testing during the IV Swinger hardware and software development was performed incrementally and much of it could well be considered "experimentation" rather than "testing". It made sense to continuously try things out along the way and work the issues out at each step. Even though there was a pretty clear vision of the design before the testing started, many of the details were worked out through this experimentation process. Some of the same testing could and probably should be performed by anyone building an IV Swinger. It would be very difficult to debug if no testing is done until after the whole thing is put together.

All of the circuitry that ended up on the Perma-Proto boards was initially tested on an ordinary breadboard. It didn't always look as nice and pretty as it ended up, as can be seen from Figure 10-1 below!

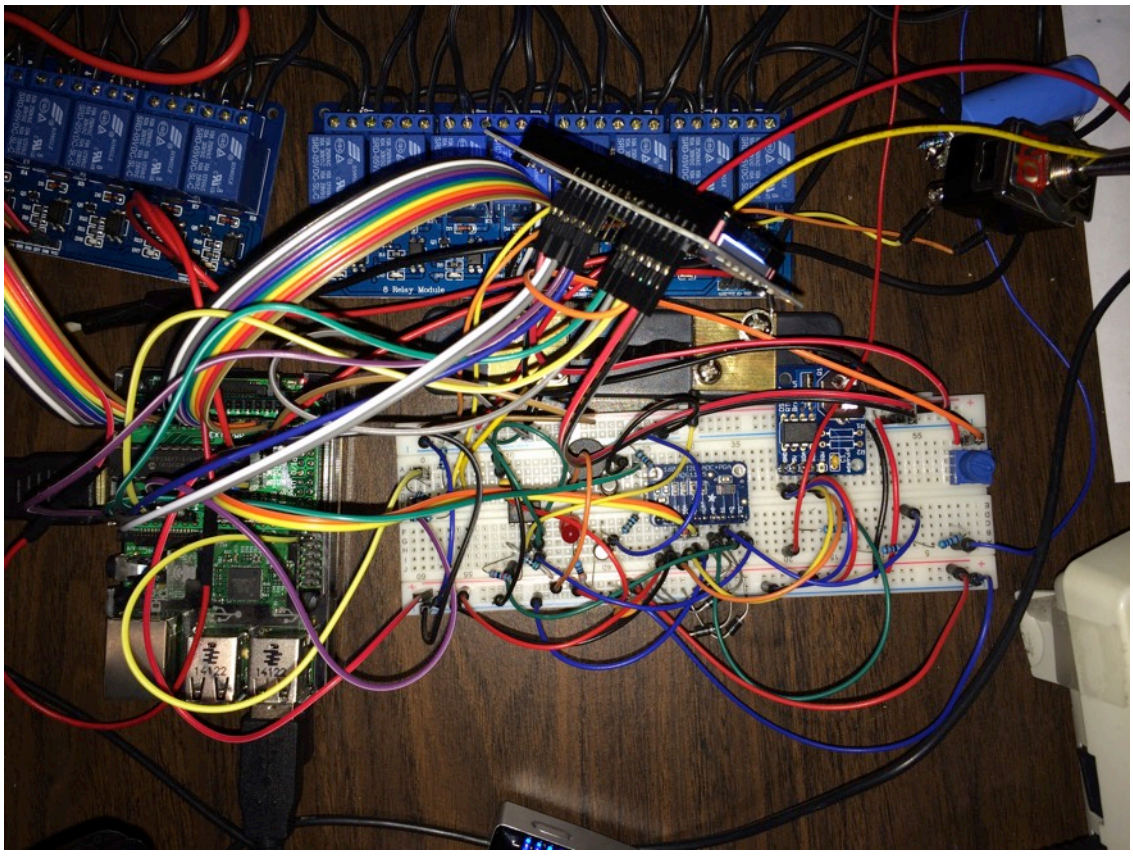


Figure 10-1: Breadboard testing

10.1 Relay control testing

Some of the testing was possible without anything driving the load circuit; some even before there was a load circuit. The earliest testing was of the relay control functionality.

The Slice of PI/O kit was assembled/soldered and piggybacked onto the Raspberry Pi. The Raspberry Pi was powered by a normal plug-in power supply (no battery yet), and connected to a keyboard, monitor and mouse. The first step was to test the I^2C communication between the Raspberry Pi and the MCP23017 I/O extender. All of the necessary installations and configurations were performed as described in Sections 8.2.4, 9.2.1, 9.2.2 and 9.2.4. This includes the i2c-tools utility. The most basic test

was to verify that the MCP23017 was identified as a device on the I^2C bus. This was accomplished with the `i2cdetect` command as follows:

```
pi@raspberrypi ~ $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: 20 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- UU -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

The hex value `20` in the 0 column is the MCP23017. This is where we expect to see it because of the solder connections shown in Figure 5-6 on page 62.

Next, the `i2cset` command was used to manually write to registers in the MCP23017:

```
pi@raspberrypi ~ $ i2cset -y 1 0x20 0x00 0x00
pi@raspberrypi ~ $ i2cset -y 1 0x20 0x14 0x01
```

The first command writes all zeros to the register at address 0x00. This is the IODIRA register, and writing a value of 0x00 to it sets the A7:0 pins on the MCP23017 to output mode. The second command writes a value of 0x01 to address 0x14. This is the OLATA register, and writing a value of 0x01 to it sets pin A0 to a value of 1 (high) and pins A7:1 to a value of 0 (low). It was easy to confirm that this worked by monitoring the A0 pin with a digital multimeter (DMM) and watching its voltage go from 0V to +5V when the second command was run.

Next, the relays were connected to the Slice of PI/O outputs as shown in Figure 5-7 on page 64. The relay coil power (JD-VCC) was connected to its own independent +5V power supply. The `i2cset` command was then used to set the relays to different combinations of active and inactive, keeping in mind that the relay controls are active-low (meaning a value of 0 turns the relay on). The following sequence turns all relays off and then turns them on one-by-one until all 16 are on:

```
pi@raspberrypi ~ $ i2cset -y 1 0x20 0x00 0x00
pi@raspberrypi ~ $ i2cset -y 1 0x20 0x01 0x00
pi@raspberrypi ~ $ i2cset -y 1 0x20 0x14 0xFF
pi@raspberrypi ~ $ i2cset -y 1 0x20 0x15 0xFF
pi@raspberrypi ~ $ i2cset -y 1 0x20 0x14 0xFE
pi@raspberrypi ~ $ i2cset -y 1 0x20 0x14 0xFC
pi@raspberrypi ~ $ i2cset -y 1 0x20 0x14 0xF8
pi@raspberrypi ~ $ i2cset -y 1 0x20 0x14 0xF0
pi@raspberrypi ~ $ i2cset -y 1 0x20 0x14 0xE0
pi@raspberrypi ~ $ i2cset -y 1 0x20 0x14 0xC0
pi@raspberrypi ~ $ i2cset -y 1 0x20 0x14 0x80
pi@raspberrypi ~ $ i2cset -y 1 0x20 0x14 0x00
pi@raspberrypi ~ $ i2cset -y 1 0x20 0x15 0xFE
pi@raspberrypi ~ $ i2cset -y 1 0x20 0x15 0xFC
pi@raspberrypi ~ $ i2cset -y 1 0x20 0x15 0xF8
```

```
pi@raspberrypi ~ $ i2cset -y 1 0x20 0x15 0xF0
pi@raspberrypi ~ $ i2cset -y 1 0x20 0x15 0xE0
pi@raspberrypi ~ $ i2cset -y 1 0x20 0x15 0xC0
pi@raspberrypi ~ $ i2cset -y 1 0x20 0x15 0x80
pi@raspberrypi ~ $ i2cset -y 1 0x20 0x15 0x00
```

The registers at addresses 0x00 and 0x01 are ODIRA and ODIRB and the registers at addresses 0x14 and 0x15 are OLATA and OLATB. It is easy to tell when a relay is activated because it has a red LED that lights up and there is a "click". The order that the MCP23017 outputs are set to zero above is A0..A7, B0..B7. The eight relays on the left-hand module will activate in order from left to right, but the relays on the right-hand module will activate in order from right to left, illustrating the reverse order of the B outputs on the Slice of PI/O that has been mentioned several times in this document.

In addition to seeing the red LEDs light up, it was useful to verify at this point which relay terminals were connected to each other when the relay was inactive and which were connected to each other when the relay was active. With the relays oriented with the connection terminals downward as they are in Figure 5-7 on page 64, the terminal on the left is the NO terminal, the terminal in the middle is the C terminal and the terminal on the right is the NC terminal. When a relay was in the inactive state (LED off), a DMM was used to verify that there was continuity between the C (middle) and NC (right) terminals and no continuity between the NO (left) and either of the others. And when a relay was in the active state (LED on), the DMM was used to verify that there was continuity between the C (middle) and NO (left) terminals and no continuity between the NC (right) and either of the others.

Next, the ability to control the relays from Python code was tested. The necessary installations and configurations were performed as described in Sections 8.3.1, 8.3.2, 8.3.4.1 and 8.3.4.2. A small Python program was then written to use the Adafruit library code to turn the relays on and off in different patterns. This Python program was in fact the genesis of what eventually became IV_Swinger.py. This program no longer exists, but can easily be re-created by copying and pasting pieces of IV_Swinger.py, namely:

- *Adafruit_MCP230xx* import
- MCP23017_I2C_ADDR and MCP23017_PIN_COUNT definitions
- RELAY_OFF, RELAY_ON and ALL_RELAYS_OFF definitions
- Load pattern defines (NONE, etc)
- Functions: ***turn_off_all_relays***, ***swizzle_byte***, ***swizzle_msb***, ***set_relays_to_pattern***
- 3 lines from ***run*** to create MCP23017 I/O extender instance and set all pins as outputs

With the above, code can be written that calls the ***turn_off_all_relays*** and ***set_relays_to_pattern*** functions to set the relays to different patterns. The load pattern defines can be used with the latter function to set the relays to the patterns that are actually used by IV_Swinger.py. Before changing the pattern, the code will have to sleep for some amount of time or else the relays won't have enough time to switch. You must import the *time* module in order to be able to use the ***sleep*** function.

10.2 Load circuit testing

Without a way to actually drive current through the load circuit, testing would be extremely limited. Using an actual PV module has some obvious drawbacks:

- PV modules are large and unwieldy
- Testing with a PV module pretty much has to be done outside with the sun shining

10.2.1 Using a bench power supply

Fortunately, a bench DC power supply can be used in place of the PV module for testing the load circuit. In fact, a DC power supply has an IV curve that is very similar to the IV curve for a PV module, so it is useful as a stand-in for the PV module for all testing.

A DC power supply capable of driving up to 10A at 30V makes an excellent surrogate PV module. It is not necessary to use a high quality power supply since none of what you pay for in a high quality supply is relevant for this usage. There are many Chinese-made 30V/10A power supplies available on EBay for around $80. There are several different "brands", but they are all probably manufactured in the same place. Most have a model number of 3010D. Figure 10-2 below is the Yihua PS-3010D that I bought and used for the IV Swinger testing.

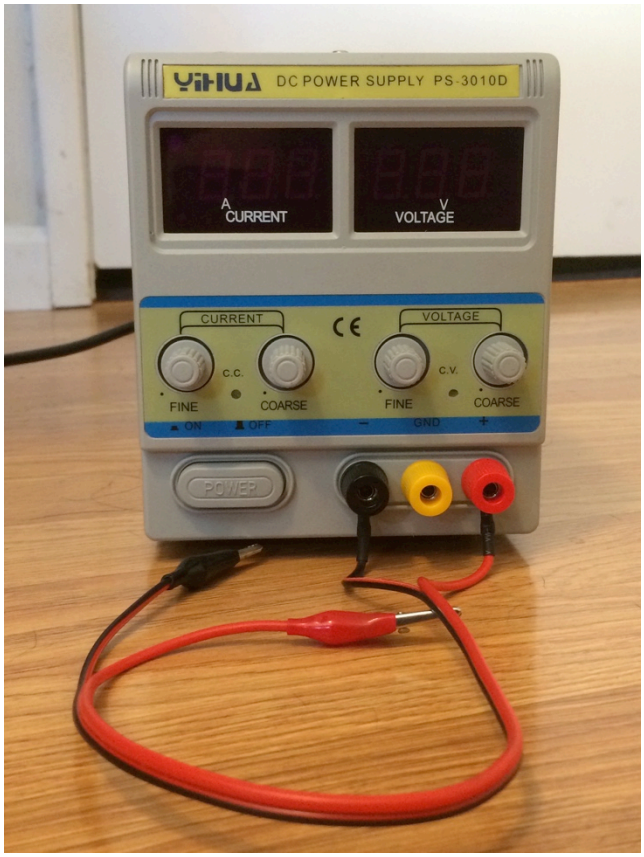


Figure 10-2: Cheap 10A/30V power supply

The CURRENT knobs allow the user to set the maximum current (I_{SC}), and the VOLTAGE knobs allow the user to set the maximum voltage (V_{OC}). Similar to a PV panel, when a low-resistance load is placed between the positive and negative terminals, the current maxes out at a value at or close to the I_{SC} value set by the CURRENT knobs. In this case, the LED labeled "C.C." between those knobs lights up because the power supply is in "constant current" mode. In constant-current mode, the resistance of the load determines the voltage (V=IR). When a high-resistance load is placed between the positive and negative terminals, the voltage maxes out at a value at or close to the V_{OC} value set by the VOLTAGE

knobs. In this case, the LED labeled "C.V." between those knobs lights up because the power supply is in "constant voltage" mode. In constant-voltage mode, the resistance of the load determines the current (I=V/R). There will be some load resistance value that is right at the value where the mode switches from constant-current to constant-voltage. This is the knee of the power supply's IV curve.

One characteristic that a cheap bench power supply and a PV module do not share is their responsiveness to changes in the load resistance. This is probably due to internal inductance and/or capacitance in the power supply that prevent its output current and/or voltage from changing quickly. Whatever the reason, when testing the IV Swinger with a power supply it is necessary to add a software delay between changing the load value and taking the I and V measurements[16]. This delay can be very small or zero when a real PV module is used because it can nearly instantaneously change its output current and voltage when the load changes.

10.3 Meter testing and calibration

The first step in bringing up the voltmeter and ammeter was to assemble/solder the Adafruit ADS1115 ADC breakout board kit and make the following connections:

- VDD to +5V
- GND to ground
- ADDR to ground
- SCL and SDA to Slice of PI/O (I^2C)
- A0:A3 unconnected (for now)

At this point, the `i2cdetect` utility was run to verify that the device was found:

```
pi@raspberrypi ~ $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: 20 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- UU -- -- -- --
40: -- -- -- -- -- -- -- -- 48 -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

The ADS1115 can be seen at address 0x48.

Next, the voltage divider resistor circuit (R1, R2, R3) was built and tested, but not connected to the ADC yet. The DMM was used to measure each resistor's resistance to confirm that it was as marked. I used 1% tolerance resistors, so the measured values were pretty close to the expected values, but the exact measured values were noted for use by the software. The power supply was used to apply a positive voltage across the resistors and the DMM was used to measure the input voltage and the output voltage to confirm that the division was as expected. The Schottky diode clamp circuits were then added, and the division test was repeated to verify that the diodes have no effect when the voltage is positive. Then

[16] To be clearer, the effect of not adding this delay is only that the IV curve looks bad when the IV Swinger is used with a power supply. It doesn't really hurt anything.

negative voltages were applied and the DMM was used to verify that the output voltage was clamped at about -5.3V.

Next, the ammeter voltage multiplier circuit (op amp and resistors R_f and R_g) was built and tested with the shunt resistor but not connected to the ADC yet. The DMM was used to measure each resistor's resistance to confirm that it was as marked, and the exact measured values were noted for use by the software. The power supply was used to generate a current of about 5A directly through the shunt resistor. The DMM was used (in series) to measure the current, but then removed from the circuit. Then the DMM was used to measure the output voltage of the op amp to verify that it was the expected value (~2.1V).

Next, the ability to read the ADC from Python code was tested using the Adafruit *ADS1x15* library. The Adafruit GitHub repository includes a couple simple test programs: `ads1x15_ex_singleended.py` and `ads1x15_ex_differential.py`. These programs just print out the voltages read at the ADC inputs in single-ended and differential modes respectively. The voltage divider and ammeter circuits were connected to the A0:3 inputs of the ADC and the above experiments were repeated, but using the Python programs to read the output voltage values in addition to the DMM.

The Adafruit code was used as the starting point for the `IV_Swinger.py` code described in Section 8.4.5 on page 84, and that code was tested in the same way.

The accuracy of the meter readings was verified by using the DMM to measure the I_{SC} and V_{OC} values and comparing them with the values measured by the meter circuitry and software. The current control knob of the bench power supply was adjusted to different I_{SC} values from near 0A up to 10A to verify that the accuracy was good across the whole current range. And the voltage control knob of the bench power supply was adjusted to different V_{OC} values from near 0V up to 30V to verify that the accuracy was good across the whole voltage range.

10.4 Other electronics testing

10.4.1 DPST testing

The DPST sensing circuit was built and connected to GPIO pin 4 of the Raspberry Pi (via the Slice of PI/O) as described in Section 5.7 on page 64. Then a small Python program was used to import the *RPi.GPIO* library and use it to set GPIO pin 4 as an input pin, reads its value and print the value. This program was run with the switch in the OFF position and the ON position to verify that the correct value was printed.

10.4.2 LCD display testing

The 16x2 LCD was connected to power, ground, the contrast potentiometer, and the Raspberry Pi GPIO pins (via the Slice of PI/O) as described in Section 5.8 on page 65. Then a small Python program was used to import the *Adafruit_CharLCD* library module and use it to display a message on the LCD. Once this was working, the Python code was written to display a scrolling message and to use a thread to keep it scrolling while doing other things. This code was incrementally tested as it was written.

10.4.3 Real-Time Clock testing

Before adding the RTC, it was useful to observe the problem that it solves. The Raspberry Pi was booted without a network connection and the "date" command was run from the CLI.

The Adafruit DS1307 RTC breakout board kit was constructed and the battery was inserted. The following connections were then made:

- 5V (VCC) to +5V
- GND to ground
- SCL and SDA to Slice of PI/O (I^2C)
- SQW unconnected

At this point, the `i2cdetect` utility was run to verify that the device was found:

```
pi@raspberrypi ~ $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: 20 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- UU -- -- -- --
40: -- -- -- -- -- -- -- -- 48 -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

The DS1307 can be seen at address 0x68.

Additional manual testing was performed as described in the Adafruit tutorial:
https://learn.adafruit.com/adding-a-real-time-clock-to-raspberry-pi

The Adafruit tutorial also describes the file modifications needed to automatically set the time from the RTC at boot time. These are mentioned above in Section 9.2.1 on page 100 and Section 9.2.3 on page 101.

With the file modifications in place and the DS1307 connected, the Raspberry Pi was booted without a network connection and the "date" command was run to verify that the time was now correct.

10.4.4 Piezo buzzer testing

The piezo buzzer circuit was built and connected to GPIO pin 18 of the Raspberry Pi (via the Slice of PI/O) as described in Section 5.10 on page 69. Then a small Python program was used to import the *RPi.GPIO* library and use it to set GPIO pin 18 as an output pin, and then to set the pin to a high value to sound the buzzer.

Later the multithreading code to sound the buzzer in a warning pattern while other code is executing was written and tested (see Section 8.4.3 on page 82).

10.4.5 Shutdown button testing

The shutdown button sensing circuit was built and connected to GPIO pin 5 of the Raspberry Pi (via the Slice of PI/O) as described in Section 5.11 on page 70. Then a small Python program was used to import the *RPi.GPIO* library and use it to set GPIO pin 5 as an input pin, read its value and print the value[17]. This program was run with the button not pressed and then pressed, to verify that the correct value was printed.

Later the multithreading code to handle the shutdown button press was also written and tested (see Section 8.4.3 on page 82). This was after the main program loop was already implemented. It was verified that the shutdown button press caused the main program flow to be interrupted and the ***`pushbutton_callback`*** function to be called. The case where the button was pressed only briefly was tested to verify that the shutdown was not performed (and the appropriate message was displayed on the LCD). And the case where it was held down for 3 seconds or longer was tested to verify that the shutdown was performed.

10.5 Arc control testing

Section 3.2.6 on page 43 discusses the measures taken to reduce arcing across the contacts of the DPST and the relays. Verifying that those measures are effective is difficult.

In the case of the DPST, the final software assures that the switch is never opened under load (unless the user disobeys instructions). A "correct by construction" argument can be used.

In the case of the relays there is no direct way to observe whether arcing is occurring, and if it is, how strong/damaging it is. This is because the relay cases are sealed. It might be possible to carefully cut open the case of a relay and visually compare the arcing with and without the snubber circuit, but this would be destructive and was not done. The only real way to know if the snubbers help or not would be to perform relay lifetime tests with and without the snubbers. This would be very time consuming and would also destroy the relays.

The bottom line is that no arc control testing was performed. Only time will tell how long the relays last.

10.6 Power testing

The ability of the battery pack to supply adequate power was not explicitly tested. The symptoms of insufficient power would be one of the following:

- Failure of the relays to switch or stay switched
- Crashes of the Raspberry Pi not attributable to other root causes

Neither of these was observed during the extensive system testing that was performed. However, none of the system testing was performed with all 4 USB drives installed. Doing so would be a good stress test on the battery pack's ability to supply adequate power.

Also not explicitly tested was the isolation between the relay coil power and the power to the Raspberry Pi and other electronics. As described in Section 3.2.4 on page 34, a separate USB output from the

[17] The same program used for the DPST testing was used, with the GPIO pin changed from 4 to 5.

battery pack is used for the relay coil power specifically to keep noise from the coils from interfering with the power of the Raspberry Pi. Although this was not specifically measured or tested, the likely symptom if it were not working would be Raspberry Pi crashes, and that was not observed during system testing.

10.7 Thermal testing

No quantitative thermal testing was performed. Some qualitative thermal testing was performed, however. Namely, the software was modified to delay for several seconds between each measurement and the bench power supply was used with the voltage set to 30V and the current set to 10A. At the completion of the IV curve tracing, the coils were felt by hand to see how hot they were. The hottest one was the ONE load and it was uncomfortably hot to touch, but not hot enough to cause a burn. The others were all cooler, with the higher numbered coils and the power resistors either cool or barely warm. The aluminum rod with the ONE load got warm but not hot at the end closest to the ONE load coil and was fairly cool at the other end. Without the artificial delay between measurements, even the ONE coil didn't get very warm unless many successive runs were performed back-to-back.

10.8 System/software testing

Most of the testing that was actually performed was not focused on a particular part of the design but was testing the system as a whole, including the software. The majority of the system testing used the bench power supply in place of the PV module and was done before all of the components were packed into the acrylic case. The Raspberry Pi was connected to a keyboard, mouse, and monitor to facilitate debugging and making changes to the Python code.

Here is a list (not necessarily exhaustive) of things that were tested at the system level[18]:

- Different combinations of I_{SC} and V_{OC} (high/high, high/low, low/high, low/low, etc.)
- Alarm when V_{OC} is zero
- Alarm when V_{OC} is negative
- No USB drive inserted
    - But one is inserted before timeout
    - Timeout expires
- I_{SC} extrapolation
- MPP interpolation
- LCD messages (multithreading and susceptibility to relay noise)
- PDF generation
- CSV file generation
- Copying files to USB drive(s)
- Idle shutdown

Looking at the generated IV curves was the primary way to confirm correctness of the results. The ***xpdf*** utility was run directly on the Raspberry Pi to view the PDF generated with gnuplot immediately after a run without having to move the USB drive to a different computer.

[18] This list is a combination of conditions that were tested for and functionality that was verified.

10.8.1 Using real PV modules

Although about 90% of the system testing was possible using the bench power supply in place of a PV module, some interesting things were not discovered until the IV Swinger was tested with real PV modules and real sunshine.

One such discovery was that I_{SC} can fluctuate significantly and quickly when there is any amount of haze moving across the sky. The implication of this is that it is important to swing the curve as quickly as possible. When the bench power supply was used, the software had to insert a delay each time it changed the load value to allow the power supply output current and voltage to "settle" (0.2 seconds worked well). Such is not the case for a PV module, so this delay was changed to a much smaller value (0.05 seconds). Even with essentially no delay between measurements, it was found that it doesn't work to swing the IV curve in two "passes". This was attempted as a way to avoid switching the HALF relay on every other measurement – one pass was with the HALF relay activated and the other was with it deactivated. But when these two passes were merged to plot the IV curve, the curve was very jagged due to the change in I_{SC} between the two passes, even though there was only about two seconds between the passes.

The problem with "wide and low" IV curves had been observed with the bench power supply (low I_{SC}/high V_{OC} case), but it wasn't clear that it was important to address until testing was performed with a real PV module and low insolation. The "base load" algorithm described in Section 8.4.6.3.4 on page 87 (putting the power resistors at the beginning of the curve) was implemented only after some testing with a real PV module had been done.

It was also soon after the initial testing with the real PV modules that the HALF relay began to fail (see Figure 3-17 on page 44). This was after close to 1000 IV curves had been generated with the bench power supply. The fact that the failure happened at this point may have been because all of that testing with the power supply had done most of the damage and it was just a coincidence that the PV testing happened to begin just as the relay was about to fail. It also could be because the original steel cooling rods were only inserted through the heating coils for a fairly small number of those ~1000 runs (near the end) and that is really what did the relay in. But it is also possible/probable that the real PV module was a lot rougher on the relays than the power supply. One reason would be the fact that the PV module had a higher V_{OC} than 30V. Another could be that the PV module may be more capable of sustaining the current required for an arc. In any case, it wasn't until the initial PV module testing and the relay failure that the whole issue of arc avoidance and suppression was studied and addressed. Both relay modules were replaced, the snubbers were added, the cooling rods were changed from steel to aluminum, and the software was modified as described in Section 3.2.6.3 on page 47 and Section 8.4.6.3.4 on page 87.

And finally, testing the IV Swinger's handling of shading cases was only possible with real PV modules. The added challenge is due to the fact that there are multiple "knees" when a PV (with bypass diodes) is partially shaded.

11 Bill of Materials / Cost

Below is a spreadsheet of all of the materials used for the original IV Swinger, including where they were purchased and how much they cost.

Description	Quantity	Unit Price	Total Price	Purchased From	Notes
Load circuit					
Portable 12V Car Immersion Heater Tea Coffee Water Auto Electric Heater	14	$2.67	$37.38	Ebay (China/HK)	120W, 10A, 2cm i.d.
Active Low 8 Channel Relay Module Board for Arduino PIC AVR MCU ARM 5V	2	$7.00	$14.00	Ebay	Sainsmart clone
50 ft. 18/2 Thermostat Wire - solid core 210-1002BR	1	$10.07	$10.07	Home Depot	Just need white conductor; have to cut off brown sheathing
Aluminum Heatsink Cooling Cooler Heat Spreader for 5 x 3W Aquarium LED Light	1	$8.63	$8.63	Ebay (China/HK)	Size (L x W x H): Approx. 9.5 x 3 x 0.8 inch / 240 x 76 x 21mm. Cut in half.
6 ohm 50W power resistors	5	$1.63	$8.16	Ebay	Only available in even numbers unfortunately
2 ohm / 2R - 1 watt 1W - 5% - Carbon Film Resistors	20	$0.39	$7.75	Ebay	For snubbers
6061 Aluminum rod - 0.75" x 11"	2	$2.82	$5.65	Ebay	Need to wrap with 39" of heavy duty aluminum foil to increase diameter to 2cm
Solar Panel Cable 16 AWG - MC4 PV Extension- 6FT	1	$5.50	$5.50	Ebay	Cut in half
Zinc Plated Split Ring Pipe Hangers ¾" - H72-075	4	$1.30	$5.20	Ebay	Support aluminum rods. Threaded hole is 3/8-16
1/4" OD x 10' refrigeration coil PCLE-250R010 (copper tube)	0.5	$9.98	$4.99	Home Depot	Cut 28 pieces, each 1.25" long - protection for heating coil filaments
Heavy Duty 20A 125V AC 15A 250V AC DPST On/Off 2 Position 4 Term Toggle Switch	1	$1.50	$1.50	Ebay	DPST switch
1/2 in. 36 in. Plain Steel Flat Bar with 1/8 in. Thick - Crown Bolt 43970	1	$3.57	$3.57	Home Depot	Cut into four 9" pieces, drill holes - each pair clamps one set of coil legs
Button Head Allen Bolts 3/8-16 X 1/2" Stainless Steel	4	$0.85	$3.40	Ebay	Connect split-ring hangers to case
4.7μF/250V Radial Electrolytic Capacitor 105C	20	$0.12	$2.30	Ebay	For snubbers
Heat shrink 7mm - 3m	0.5	$4.40	$2.20	Amazon	Fits around copper tubing and 18awg wire - strain relief
Single-Conductor #6 Stranded to 14 AWG Type BTC Copper Wire Connectors - Blackburn Model # BTC0614-B2-5	2	$1.09	$2.18	Home Depot	Lugs to connect solar panel cables. Screw onto shunt resistor.
#10-24 tpi Coarse Zinc-Plated Steel Wing Nut (4-Pack)	4	$0.30	$1.18	Home Depot	For coil clamps
1/4 in. x 3/4 in. x 10 ft. Black Sponge Rubber Foam Weatherstrip Tape - MD 06593	0.25	$3.76	$0.94	Home Depot	Goes around aluminum rods where clamped by split-ring hangers
#10-24 tpi x 3/4 in. Zinc-Plated Round-Head Combo Drive Machine Screw (8-Piece per Pack)	6	$0.15	$0.89	Home Depot	For coil clamps (4), and also shunt resistor to case (2)
Heat shrink 2.5mm - 6m	0.25	$3.01	$0.75	Amazon	For filament to 18awg wire connection
#10 Zinc-Plated Steel Flat Washers (30-Pack)	8	$0.04	$0.31	Home Depot	For coil clamps (4), and also shunt resistor to case (2)
#8 1/2 in. Phillips Pan-Head Self-Drilling Screws (300-Pack) - Teks 21350	14	$0.02	$0.26	Home Depot	Connect 6Ω power resistors to heat sink. Connect heat sink to acrylic case
Heavy Duty aluminum foil	1	$6.00	$6.00	Grocery store	Wrap aluminum rods to increase diameter from 3/4" to 2cm (39" each)
Meters					
ADS1115 ADC	1	$14.95	$14.95	Adafruit	ADS1015 is cheaper and would be OK, but needs minor S/W change.
10A/75mV Shunt Resistor	1	$3.62	$3.62	Amazon	Ammeter
Op amp – TLV2462	1	$2.95	$2.95	Adafruit	Ammeter
Schottky diodes	5	$0.14	$0.70	Sparkfun	ADC input protection(4). Other one is for piezo circuit.
180kΩ resistor - 1/4W	1	0.01	$0.01	Ebay	Voltmeter - divider
8.2kΩ resistor - 1/4W	1	0.01	$0.01	Ebay	Voltmeter - divider
5.6kΩ resistor - 1/4W	1	0.01	$0.01	Ebay	Voltmeter - divider
82kΩ resistor - 1/4W	1	0.01	$0.01	Ebay	Ammeter - op amp circuit
1.5kΩ resistor - 1/4W	1	0.01	$0.01	Ebay	Ammeter - op amp circuit
Computer and Electronics					
Raspberry Pi B+	1	$29.95	$29.95	Adafruit	$10 cheaper than Pi 2
Raspberry Pi - Slice of PI/O (kit)	1	$12.75	$12.75	Ebay	I/O expansion
16x2 LCD display	1	$9.95	$9.95	Adafruit	$1.90 from China
16GB MicroSD card	1	$9.64	$9.64	Amazon	8G would be fine, but only saves $1
DS1307 RTC module	1	$9.00	$9.00	Adafruit	Keeps time somewhat correct when not connected to internet
6 inch Right Angle Short HDMI 1.4 Male to Female Extension Cable Adapter	1	$8.95	$8.95	Ebay (China/HK)	Optional if RPi is set up as wifi AP
Permaproto half-size breadboard	2	$4.50	$9.00	Adafruit	2"x3.2". 2 mounting holes.
StarTech PC Mounting Computer Screws M3 x 1/4-Inches Long Standoff - 50 Pack SCREWM3	40	$0.08	$3.14	Amazon	2 per standoff
Piezo Buzzer 3Pcs 5V Active (YMD12095G-5V)	1	$1.48	$1.48	Ebay	Must be "active" type (Adafruit sells one called "breadboard friendly")
Female Thread Brass Pillar PCB Standoff Hexagonal Spacer M3x10mm	20	$0.07	$1.37	Amazon	Used for: permaprotos (4), RPi (4), LCD (4), relay modules (8). First IV Swinger used F-M type, but F-F is much easier.
40X Dupont Wire Male to Female Jumper Cable 20cm	1	$1.31	$1.31	Ebay (China/HK)	Slice of PI/O to relay modules and other
Momentary Button - Panel Mount (Black)	1	$0.95	$0.95	Sparkfun	Shutdown button
2n2222a transistor	1	$0.39	$0.39	Ebay	Piezo buzzer circuit
10kΩ resistor - 1/4W	2	0.01	$0.02	Ebay	DPST and shutdown pulldowns
1kΩ resistor - 1/4W	3	0.01	$0.03	Ebay	DPST, shutdown and piezo series
Hookup wire - red ($2.50/25ft)	5	0.1	$0.50	Adafruit	permaproto connections
Hookup wire - black ($2.50/25ft)	5	0.1	$0.50	Adafruit	permaproto connections
Hookup wire - yellow ($2.50/25ft)	5	0.1	$0.50	Adafruit	permaproto connections
Power					
EasyAcc 15000 mAh battery pack	1	$25.99	$25.99	Amazon	First IV Swinger used 12000 mAh model, now superseded by 15000 mAh.
Micro USB Cable - A to Right Angle Micro B (UUSBHAUB1RA) - 1ft	2	$3.99	$7.98	Amazon	One for RPi power. One for relay coil power (JD-VCC).
Micro USB 2.0 Type B Male To Female M/F Extension Extender	1	$1.99	$1.99	Ebay	Extension of battery charger port to case
Micro-USB breakout	1	$1.50	$1.50	Adafruit	Relay coil power (JD-VCC)
Enclosure					
Contoured Acrylic Model Display Case 12"x6"x6" (FlyteLine)	1	$19.99	$19.99	Ebay	Order without base
Acrylic sheet (11x14 .093 Clear)	1	$4.24	$4.24	Home Depot	Cut 12"x6" piece for base
Small extra tall round rubber feet	4	$0.56	$2.26	Ebay	.750" wide x .437" tall, #6 screw
2" long 1/4" hex 6-32 spacer standoff	2	$0.47	$0.94	Ebay	Cut in half. Glue into corners of case.
#6-32 tpi x 3/4 in. Zinc-Plated Round-Head Combo Drive Machine Screw (8-Piece) - Everbilt 803031	4	$0.15	$0.59	Home Depot	Screw feet into hex spacers glued in corners
Total			$319.99		

12 Future enhancements

The current IV Swinger design has several shortcomings that could potentially be addressed with future enhancements to the design. Among those shortcomings are:

- Voltage range, current range, and power limits constrain its use to a single PV module. It cannot handle multiple modern PV modules in series or parallel.
- Relays have limited life
- Acrylic case is fragile
- Electronics are fragile
- Unit is relatively bulky and heavy
- IV curve resolution is limited
- IV curve cannot be viewed/previewed immediately

12.1 Use 100W power resistors in place of immersion coils

Although the immersion heating coils seemed to be a good and inexpensive choice for the loads, there may be a better choice. The coils were only inexpensive when ordered directly from China on EBay, and it turns out that inexpensive 100W power resistors can also be purchased directly from China. These are available in low resistances such as 0.5Ω and 1Ω. Their unit cost is very close to the same as the coils (less than $3).

Since $P = I^2R$, the maximum current is:

$$I_{MAX} = \sqrt{\frac{100}{R}}$$

So the 0.5Ω can handle 14A and the 1Ω can handle 10A. This meets the requirements. However the 100W rating of the power resistors does assume adequate heat sinking.

The advantage of using the power resistors would be that the load bank would be much more compact and easier to construct than it is with the coils. They could all be screwed onto heat sinks in the same manner that the 50W power resistors are mounted in the current IV Swinger design. The immersion coils are somewhat labor-intensive because each one has to have all of its unneeded parts removed, and also must be modified to fit in the enclosure and to protect its heating element from being damaged.

An unknown is how inductive these resistors are. They are “wire wound”, which means they will be inductive, but their inductance isn’t specified.

12.2 Use DC/DC converter for variable load

One possible design change would be to replace the multiple discrete loads with a single fixed value load and a DC-to-DC converter to make that single load appear to be different values. This is how a Maximum Power Point Tracker (MPPT) works. A buck-boost converter is used to make the actual load that the PV panel (or string of panels) is driving “appear” to be the resistance value at the MPP, regardless of what the actual load resistance is. There is very little power lost through a buck-boost converter, but the voltage at the output is stepped up or down from the voltage at the input. If the output

voltage is higher than the input voltage, the output current is lower than the input current, and vice versa. This is because P = VI and $P_{IN} \approx P_{OUT}$, so $V_{IN}I_{IN} \approx V_{OUT}I_{OUT}$. Because V=IR, the resistance "seen" at the input of the converter is given by: $R_{IN} = R_{OUT}(V_{IN}/V_{OUT})^2$. Figure 12-1 below shows the relationships between the input and output voltage, current, and resistance of a buck-boost converter.

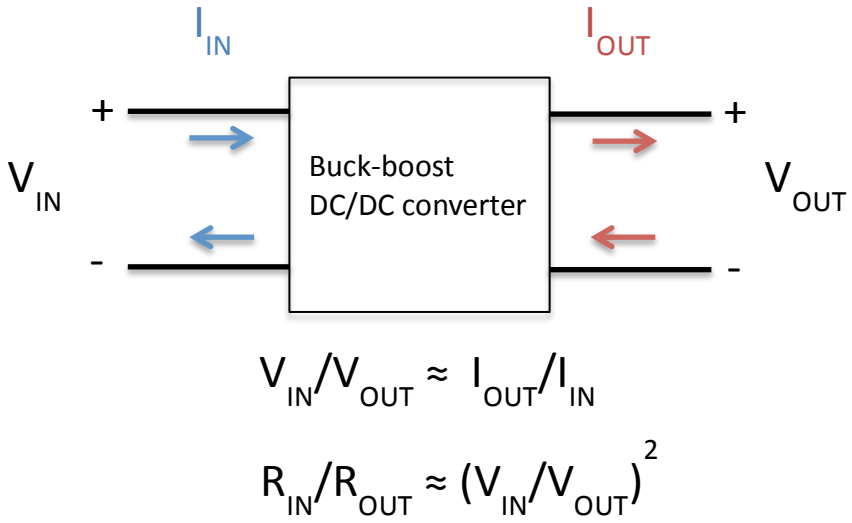


Figure 12-1: Buck-boost DC/DC converter

The V_{IN}/V_{OUT} ratio of the buck-boost converter is determined by the duty cycle of a rectangular wave control signal fed into the converter[19]. The duty cycle D of a rectangular wave is the ratio of the time that it is at a high voltage to the time that it is at a low voltage. The relationship between the duty cycle and V_{IN}/V_{OUT} ratio of an "inverting" buck-boost converter is given by:

$$\frac{V_{IN}}{V_{OUT}} = \frac{D-1}{D}$$

The PV module (or string of modules) is connected to the input side of the buck-boost converter and the load is connected to the output. The MPPT "hunts" for the MPP by varying the duty cycle of this signal, searching for the point where $V_{IN}I_{IN}$ has the greatest value. Figure 12-2 below shows a PV panel connected to a load through a buck-boost converter. This figure is generic enough to represent how a MPPT works, or how a redesigned IV Swinger could work.

[19] For more information on how a buck-boost converter works, see the Wikipedia article.

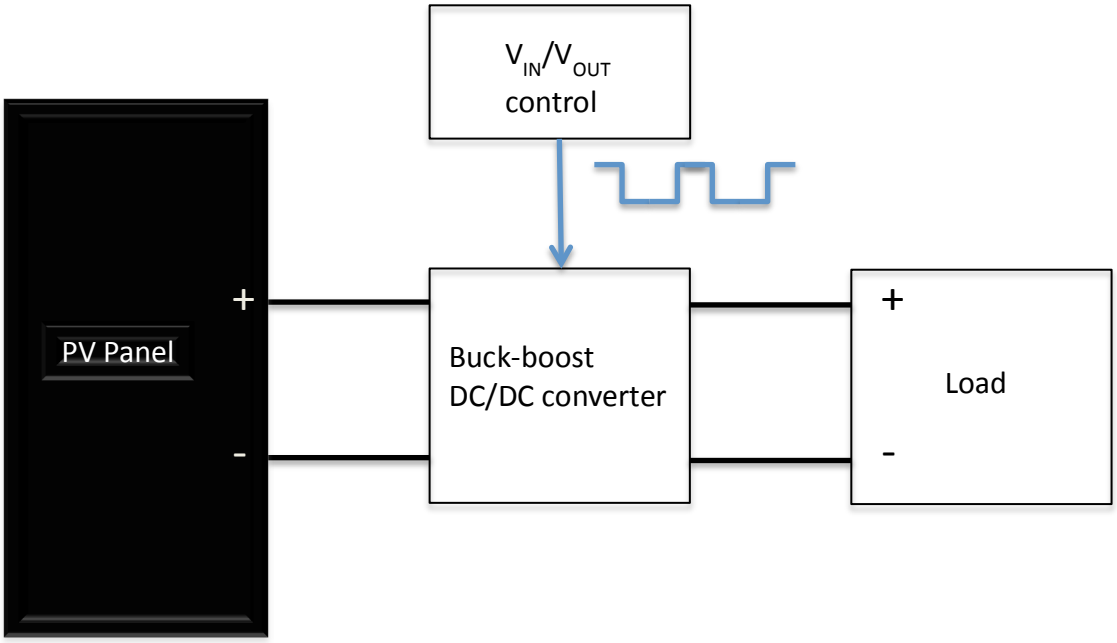


Figure 12-2: Variable load using a buck-boost converter

Instead of using the buck-boost converter to search for the MPP, the IV Swinger would use it to vary the load seen by the PV panel to swing out the IV curve. For the load to appear as a short circuit, the duty cycle D would be set to 1 (100%), so $V_{IN}/V_{OUT} = 0$, and $R_{IN} = 0$. For the load to appear as an open circuit, the duty cycle would be set to 0, so $V_{IN}/V_{OUT} = -\infty$, and $R_{IN} = \infty$. All duty cycle values less than 1 and greater than 0 would result in finite non-zero R_{IN} values. So with a single fixed-value load, the entire IV curve can be swung out by starting with D=1 and incrementally decreasing it to 0, measuring the current and voltage at each increment.

One advantage of using this scheme for the variable load is that many more measurement points could be taken since the duty cycle of the control signal could be varied nearly continuously. Especially around the knee(s) of the IV curve, this would generate a graph with much better resolution than the current design is capable of. It also would most likely be possible to take each measurement more quickly (although the ADC I^2C communication delays might start coming into play). It is also possible that this design could handle a whole string of PV modules rather than just the single module that the current design can handle. This would depend on the ability to buy or build a buck-boost converter that could handle the high voltage and power of multiple modules. This sounds possible since string inverters have built-in MPPTs. String inverters also cost several thousand dollars, however, so that may be an indication that high voltage/power buck-boost converters are expensive, and that would defeat one of the primary goals of the IV Swinger.

Another advantage could be a smaller, lighter design due to the fact that only one load is needed. Of course that single load must be able to absorb/dissipate the energy generated by the PV(s) during the swinging of the IV curve, and that could mean it has to be large. But if all of the measurements can be taken very quickly, then the load should be able to handle a lot more power than it is rated for.
A final advantage of this scheme would be the elimination of the relays, which despite the efforts to implement arc suppression and avoidance are almost certainly the limiting factor on the lifetime of the IV Swinger.

12.3 Ruggedization

The current IV Swinger design is fragile. The acrylic case and many of the electrical connections would not survive if the unit were dropped even a short distance (or if something were dropped on it). The rationale for the acrylic case was presented in Section 7 on page 76. But it may be desirable to have a "ruggedized" version of the IV Swinger. This would require the following changes:

- Replace the acrylic case with a metal one
- Eliminate all press-on jumper wire connections and solder all wires instead

12.4 On-board graphical display

One shortcoming of the current design is that the IV curve cannot be viewed without removing the USB drive, plugging it into another computer, and opening the PDF with a viewer utility. This is time-consuming enough that most users will only do it after a whole series of experiments has been performed. It would be useful to be able to display the IV curve immediately on the IV Swinger itself. This would provide more immediate identification of things such as inadvertent shading (from a passing cloud or person).

Small 128x64 LCD graphical displays are available for around $5 from China on EBay ($18 from Adafruit). This is enough resolution to generate an IV curve that is coarse, but adequate to show anything anomalous. Another possibility would be an e-ink display. Adafruit has a 2.7" 264x176 e-ink display for $40. Kindle replacement screens are available on EBay for about $22 and their resolution is 600x800.

The 128x64 LCD display is the cheapest and would probably be the easiest to get working, but still a significant amount of work. There doesn't appear to be the level of pre-written software support for it as there is for the other devices used by the current IV Swinger design. But it shouldn't be too difficult to get it connected to the Raspberry Pi and to push a 128x64 bitmap to the display. There would also be the task, however, of generating a graph in 128x64 resolution. This would probably have to be written from scratch, and that could take a fair amount of time.

Adafruit does have some Raspberry Pi support for the e-ink displays that they sell, so maybe that wouldn't be much harder to get working than the 128x64 LCD. Generating the bitmap graph would be the same degree of difficulty. $40 is pretty expensive though.

The Kindle display would be great since its resolution is good enough to produce a very nice graph. But the replacement screen is only the e-ink display itself, and doesn't include the electronics needed to drive it, so this is probably not a realistic option. It might be possible to hack an entire Kindle to be the display, but that would also be expensive unless an old used one could be found cheaply.

It isn't clear that the addition of an on-board graphical display is worth either the extra cost or development time. I actually bought both the 128x64 and e-paper displays from Adafruit, but decided to defer incorporating either of them into the design. I probably never will.

13 Appendix

13.1 IV curve for a partially shaded PV module

Multiple places in this document refer to the special case of an IV curve where there is shading on the module. Figure 13-1 shows what one such curve looks like.

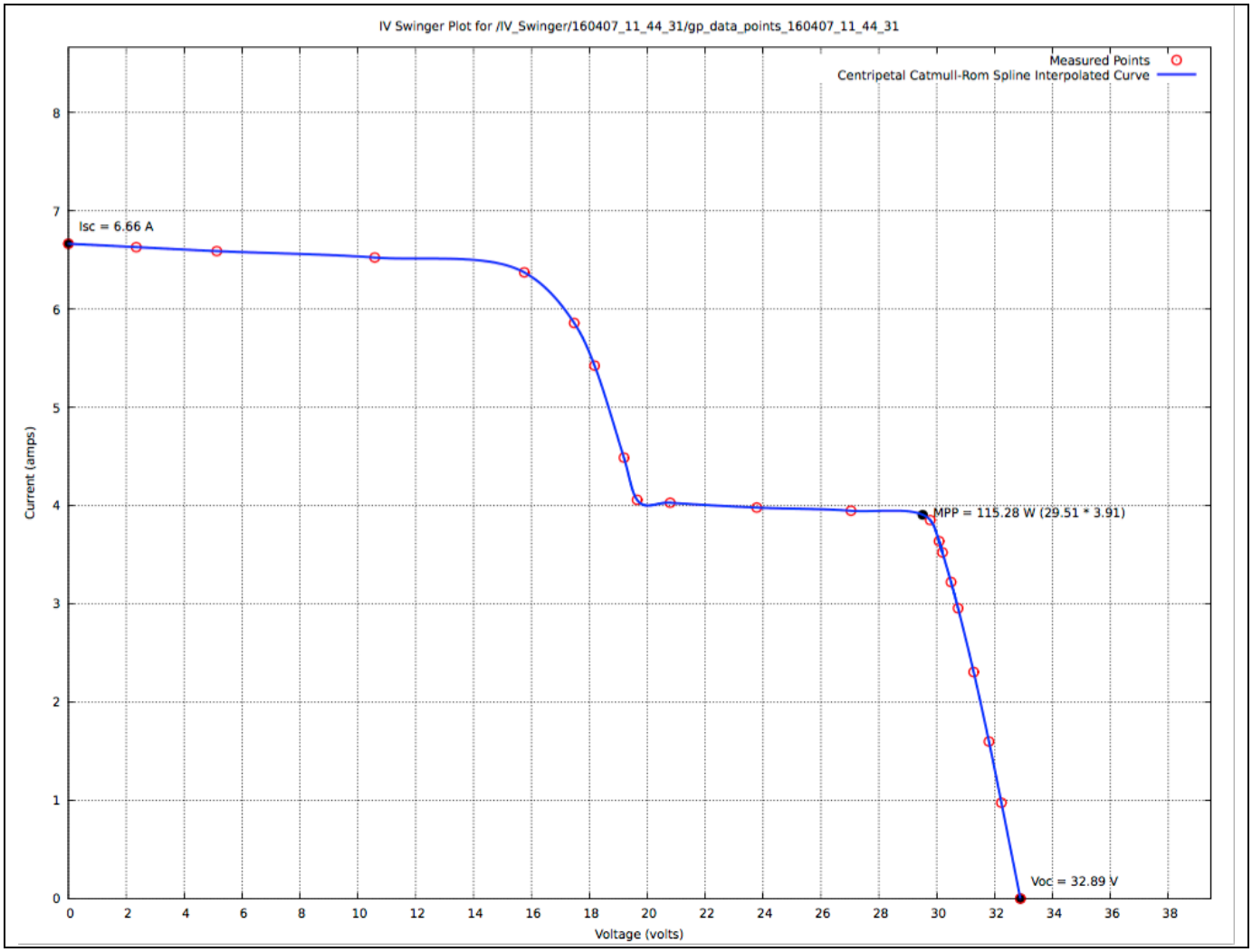


Figure 13-1: IV curve for a "shading case"

It looks like this because of the bypass diode feature that all modern PV modules have. This 60-cell module is divided into three groups of 20 cells, each of which has a bypass diode. In this case, a single cell was partially shaded, reducing its I_{sc} to about 4A. This causes the diode for that 1/3 of the panel to be forward biased, allowing current to bypass those 20 cells. This effectively turns the 60-cell module into a 40-cell module for the first part of the curve. I_{sc} is independent from the number of cells (the cells are all wired in series, so the current is the same through all of them). V_{oc} is directly proportional to the number of cells (series voltages add). The first part of the curve was on its way to a V_{oc} of about 21V (2/3 of the 60-cell Voc). But at approximately 4A and 20V the curve takes a sharp turn . This is because the voltage reaches the point where the diode becomes reverse biased and no longer bypasses the current around the 20 cells. Now it's back to a 60-cell module and the curve "rejoins" the curve for an unshaded panel. In this case, the bypass diode provided no benefit because the MPP is where it would have been

without the diode (i.e. the second knee). But if the cell had been just a bit more shaded, the first knee would have been the winner. This shows how a small amount of shade has a large effect on the MPP. Less than 1% of the module was shaded, but the power was reduced by about 1/3.