Lossless Digitally-Assisted Windowed Current Sensing for Solar Photovoltaic Applications

By

Benedict Foo

Senior Thesis in Electrical Engineering

University of Illinois at Urbana-Champaign

Advisor: R. C. N. Pilawa-Podgurski

May 2015

Abstract

Sensing current across a sense resistor adds to the total power loss and decreases efficiency. This work demonstrates the use of a current sense amplifier that measures current across a length of wire so as to minimize power loss. Current Sense Amplifiers for solar photovoltaic applications require measuring a wide range of current (e.g. 1-10A), yet a high resolution is needed for certain applications such as maximum power point tracking (MPPT) and differential power processing (DPP). The windowing technique has been adopted in this work, so as to measure the current with high resolution and yet to be able to cover the entire range of current as well. The use of PSOC 4 by Cypress offers a cost effective one chip solution to all current sensing requirements for solar photovoltaic applications. This is achieved by having a nearly $4 cost reduction from previous methods, and yet attaining under 1% error in current measurements.

Subject Keywords: Windowing, Photovoltaics, Current Sensing.

Acknowledgements

I would first like to thank Professor Robert Pilawa-Podgurski for his vision, technical insight and unwavering persistence for me to reach the end goal of this thesis. His guidance has helped prepare me for more advanced research as I am intending on pursuing a graduate degree. Huge thanks goes out to Christopher Barth who was always willing to lend a helping hand whenever I encountered problems with my project. Christopher's valuable experience in research has kept the project moving whenever I got stuck at a problem. I would like to thank Enver Candan for letting me use his bench to perform high current sweeps, Shibin and Yutian for being always willing to give me their opinions on my methods or intentions. To the rest of the Pilawa-Group, thank you, it was an honor being the only undergraduate to attend the weekly group meetings, and being amongst such brilliant yet humble students was a real eye-opener for me. I look up to all of you.

In the industry arena, I would like to thank Martin Krajci and Larry Sparling from Continental Automotive for imparting their knowledge and technical know-how onto me during my summer internship in 2014. I learnt so much from them during the summer that it has undoubtedly made me a technically strong undergraduate researcher.

I would like to thank my parents for allowing me to do my undergraduate studies in the University of Illinois at Urbana-Champaign. Without them, I would not even be aware of the opportunities that I have seized since I am here.

I would like to thank all my friends who have kept me sane and allowed me to de-stress whenever I was swamped with work. Lastly, I would like to thank Dana who has always given her undying support, college would have been utterly stressful without her.

List of Tables

List of Figures

Contents

1. Introduction

1.1 The Need for Current Measurement in Solar Photovoltaic Applications

Maximum power point tracking (MPPT) is dependent on accurate current and voltage measurements in order to effectively apply techniques such as the common perturb and observe, or dithering digital ripple correlation control [1], [2], [3]. They require high resolution power ripple measurements which involve high precision power processing components [1]. Also, in multilevel MPPT schemes that require partial processing dc-dc power converters, there is a need to detect and measure the monotocity in the current [4]. However, noise is a big factor in affecting the accuracy of current measurements [5]. More accurate current measurements in MPPT systems are made possible through various techniques such as current ripple minimization or system parameter optimization through noise analysis [5], [6]. This paves the way for the focus on the current sensing techniques themselves in this thesis. Another photovoltaic (PV) application that requires current measurements is differential power processing (DPP). In DPP for solar photovoltaic cells, current needs to be accurately measured in order to determine the power mismatch between PV cells and hence process that difference in power [7], [8].

1.2 Windowing Technique

PV current has a very wide range of current. For example, for a current range of 0 to 10A, the voltage range over a 1mΩ resistor would be 0 to 10mV. This wide range thus affects the resolution of the current measurement. Assuming an ADC resolution of 12 bits, then each bit, measures 2.44mA. However, if windowing were to be applied, then assuming a window of 2V, each bit measures 0.122mA, thus offering much higher (i.e. 20x) resolution in measurements.

Figures 1.1 to 1.3 gives a graphical overview of the benefits of the windowing technique. The sensed voltage ripple is very small since we are dealing with 0.125mV. Hence we need to scale this ripple. If we were to scale the ripple to fit half of a 3.3V ADC window (i.e. 1.65V) then the voltage ripple would be in the region of 6-7V which exceeds most ADC voltage ranges. Hence there is a need to shift this ripple back into the range of the ADC. This forms the crux of the need of the windowing technique as well as its merits where a high resolution on the voltage ripple can be attained.

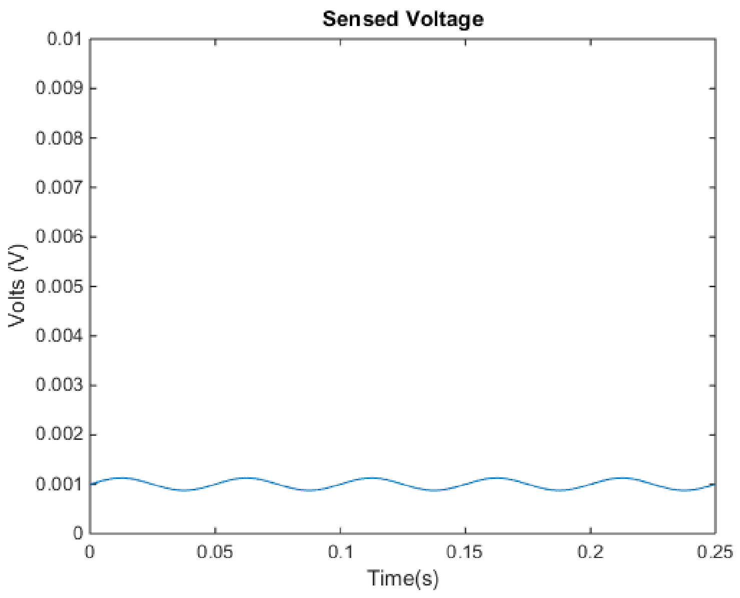


Figure 1.1. Voltage Ripple

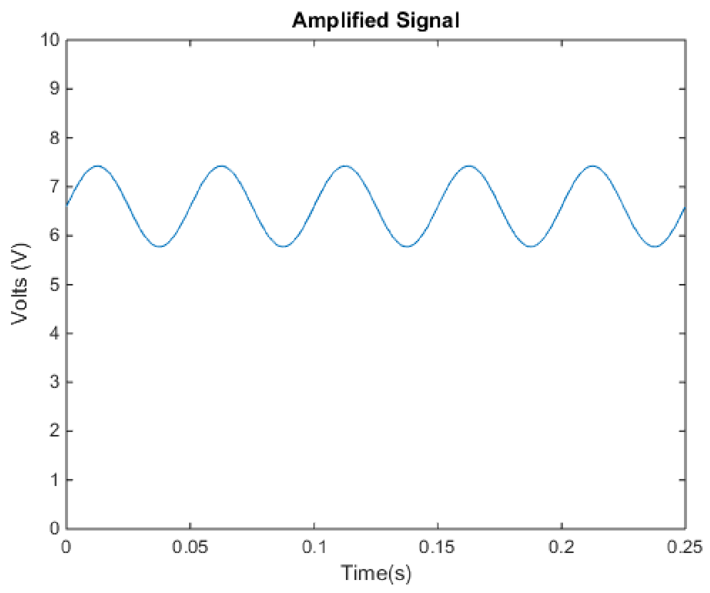


Figure 1.2. Amplified Signal

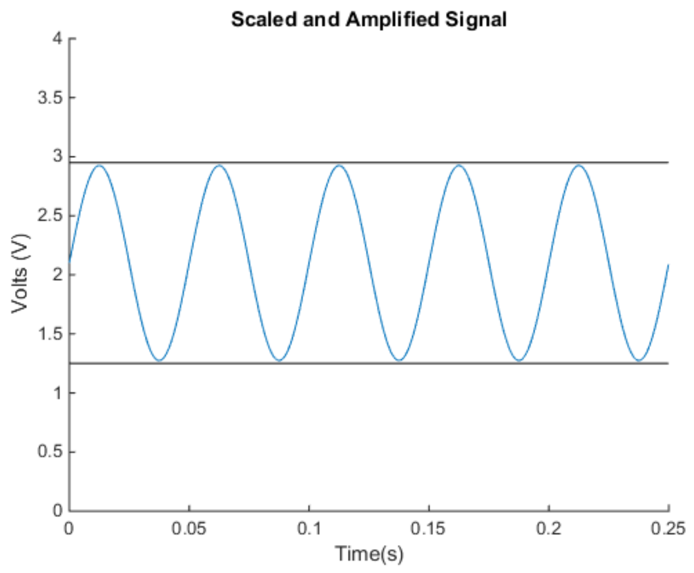


Figure 1.3. Scaled and Amplified Signal

1.3 Lossless Current Sensing

Sensing string current is needed in a variety of MPPT and DPP topologies [8], [9]. Windowing techniques have been implemented for PV MPPT techniques, however they introduce a sense resistor for low-side sensing [1]. This introduces series power loss into the design and hence affects the efficiency of the design. For example, a 10A current output would lead to a 1W power loss if a 10mΩ resistor is used. There have been numerous lossless current sensing techniques being introduced lately, but they are mostly implemented in the IC level and are not fully suitable to measure string current in photovoltaic systems [10], [11]. Another technique capitalizes on the parasitic resistance of an inductor, but is only suited for relative and not absolute current measurements [12]. Another direction is to reduce the number of current sensing circuits in a given system using a distributed control algorithm in DPP, but string current measurement is nevertheless needed [13]. Hence in this work, voltage is measured across a length of wire found on the back of PV panels to eliminate the introduction of additional losses through current sensing. This ensures that absolute current can be measured accurately and yet remain in the integrated system level. Sensing over a length of wire requires a high precision op-amp to handle the high gain bandwidth as well as the low sense voltage. However, high precision op-amps cost typically in the range of $2-3, which is considered expensive. The decrease in non-panel costs has become increasingly important as falling panel installation costs have been largely attributed to the decrease in module costs [14]. Hence there is a need to find a cheap, yet accurate current sensing solution.

1.4 One Chip Solution

The integration of maximum power point tracking systems with power electronics have been proven to increase overall performance of the portable power generation system [12], [15]. Hence there is a need to incorporate the current sense circuit with the power electronics of a MPPT system. Windowing techniques require the use of a microcontroller to adjust the window as it tracks the current levels. They also require op-amps to amplify the voltage signals to achieve the right resolution in the ADC. The use of PSOC4 by Cypress Semiconductor allows an integration of the two yet remains cost-effective at $1.50 per PSOC4 chip. The programmable op-amps incorporated into the PSOC4 microcontroller unit, allow the entire current sense amplifier to be based on one chip, with just external resistors to set the gains.

However, these op-amps are not built to make high-precision measurements. Hence this dissertation also delineates the various workarounds and measures performed to get the Cypress op-amps to work to an optimal level; that is to make precise current measurements with the aid of digital calibration. Hence the term digitally-assisted is used.

2. Initial Calculation

2.1 Circuit Design

Figure 2.1 shows the proposed circuit design for the current sense amplifier. It is similar to the circuit design in a previous windowed current sense amplifier by C. Barth *et al* [1].

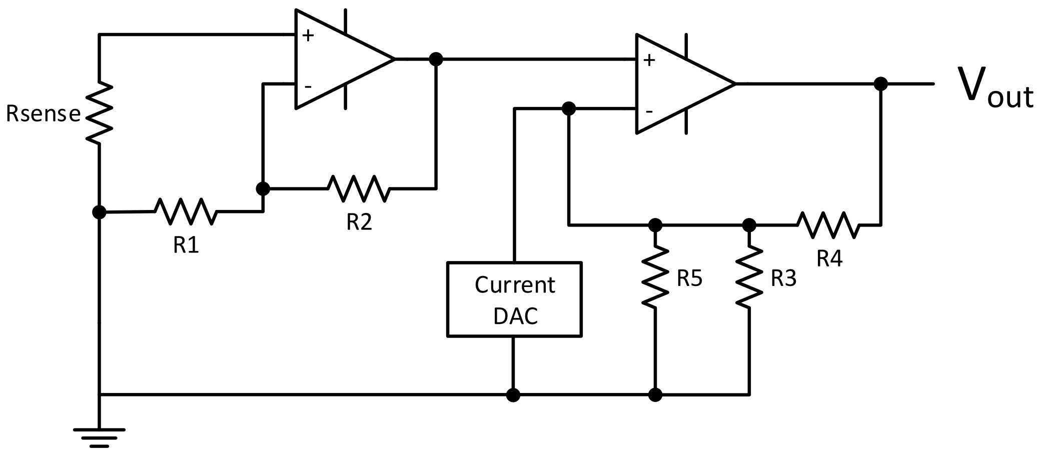


Figure 2.1. Initial Circuit Design

Note that Rsense, is not actually a resistor, but a figure representation of the length of wire that is sensed across. This value was measured to be 1mΩ using a 4-wire measurement at the return path at the back of the solar panel for low-side sensing. The first op-amp provides the initial gain stage and the second stage provides the biasing signal for windowing. Further details on this can be found in subsequent sub-sections and in previous windowed current sense amplifiers [1].

2.2 Parameter Calculations

The current ripple has to be scaled to fit the range of the ADC, with the DC value centered at the window. To determine the value of the resistors, the initial gain, as well as the bias gain need to be calculated. The following equations are taken from Equations 7 and 8 from [1].

$$K_{gain} = \frac{sr_{ADC}}{iR_{sense}} \tag{2.1}$$

$$K_{bias} = \frac{s r_{ADC} I_{max}}{\hat{\imath} V_{bias,max}} - \frac{r_{ADC}}{2 V_{bias,max}} \tag{2.2}$$

We configured the ADC to have an input range of 0 to 3.3V. A single-ended ADC was needed, since only positive current were to be measured. According to the Technical Reference Manual of the PSOC 4, this would pose the best signal-to-noise ratio for a single ended input. More on the ADC set up, as well as other software implementation will be explained in subsequent sections. The Table 2.1 below shows the value of the parameters to determine K_{gain} and K_{bias}.

Table 2.1. Initial Parameters

Parameter	Value
s	0.5
r_{ADC}	3.3V
I_{max}	10A
Rsense	1mΩ
$\hat{\imath}$	0.25A
$I_{bias,max}$	306µA

The parameter *s* refers to the proportion of the ADC range that the current ripple will be scaled to and *i* is the current ripple. As mentioned earlier, r_{ADC} is the range of ADC and is programmed to be 3.3V. The maximum current is the short circuit current as specified by the common solar PV panels. The maximum bias current, is programmed to be 306µA, which corresponds to the maximum bias current from the current DAC of the PSOC4.

2.3 Resistor Calculations

We calculated the values of the resistors using the parameters shown above, by first finding K_{gain} and K_{bias} through Equations 2.1 and 2.2. The values of K_{gain} and K_{bias} are shown in Table 2.2 below.

Table 2.2. Gain Values

K_{gain}	6600
K_{bias}	42.05

From the K_{gain} value, the gain of the initial gain stage, K_1, is chosen to be around 68, which is approximately the square root of the entire gain, so as to split up both gains evenly. The initial gain cannot be too high so as to ensure that the K_{bias} has sufficient influence on the output voltage. We solved the op-amp configuration of figure 2.1 to obtain Equations 2.3 to 2.8. The initial gain is determined by the following equation:

$$K_1 = 1 + \frac{R2}{R1} \tag{2.3}$$

We chose *R1* arbitrarily to be 47Ω, and from K_1, *R2* can then be found to be 3160Ω. The second gain stage, without the bias, is found to be:

$$K_2 = \frac{R4}{R3} + \frac{R4}{R5} + 1 \tag{2.4}$$

while K_{bias} is given by the following equation:

$$K_{bias} = \frac{R4}{R5} \tag{2.5}$$

From here, it becomes clear that another resistor value needs to be chosen arbitrarily. We decided to use *R5* as the starting point, and it was chosen to be 5000Ω. This number is chosen in the region that allows V_{bias} to be a reasonable value close to the value found in [1] which is 3V. $V_{bias,max}$ is calculated by using the Thevenin equivalent of the current DAC and *R5* and is thus found to be 1.53V. From Equation 2.2, K_{bias} is found to be 42.05 and from equation 2.5, *R4* can then be calculated and is found to be 210kΩ. *R3* can finally be found by the following equation:

$$R3 = \frac{R4}{K_2 - K_{bias} - 1} = 3918\Omega \tag{2.6}$$

From the op-amp configurations, the expected output voltage can be found by:

$$V_{out} = \left(\frac{R4}{R3} + \frac{R4}{R5} + 1\right)(1 + \frac{R2}{R1})((I_{\text{sense}} R_{sense}) - R4 I_{bias} \tag{2.7}$$

Hence, through algebraic manipulation, the current that is sensed is calculated by:

$$I_{sense} = \frac{V_{out} + I_{bias} R4}{\left(1 + \frac{R2}{R1}\right)\left(\frac{R4}{R3} + \frac{R4}{R5} + 1\right) R_{sense}} \tag{2.8}$$

A summary of the resistor values is shown in Table 2.3.

Table 2.3. Resistor Values

Resistor	Value (Ω)
R1	47
R2	3160
R3	3918
R4	210294
R5	5000

3. Simulations

For proof of concept, the circuit configuration and resistor values were simulated using ideal op-amps. From Equation 2.1, the amplified DC signal should fall in the middle of the input voltage range of the ADC, which is dependent on the value of I_{bias}. The ideal I_{bias} value can be calculated by first assuming that the output voltage will appear at 1.65V, which is the middle of the 0-3.3V ADC range. For example, assuming a current value of 1A, then, I_{bias} can be found by the following method:

$$I_{bias} = \frac{K_{gain} I_{sense} R_{sense} - V_{out}}{R4} = 23.54\mu A \tag{3.1}$$

Hence, for every input current value, the corresponding bias current can be calculated. We used Matlab Simulink for the simulation where the input current was allowed to vary. The current bias was changed accordingly based on the following equation:

$$\Delta I_{bias} = \frac{K_{gain} \Delta I_{sense} R_{sense}}{R4} \tag{3.2}$$

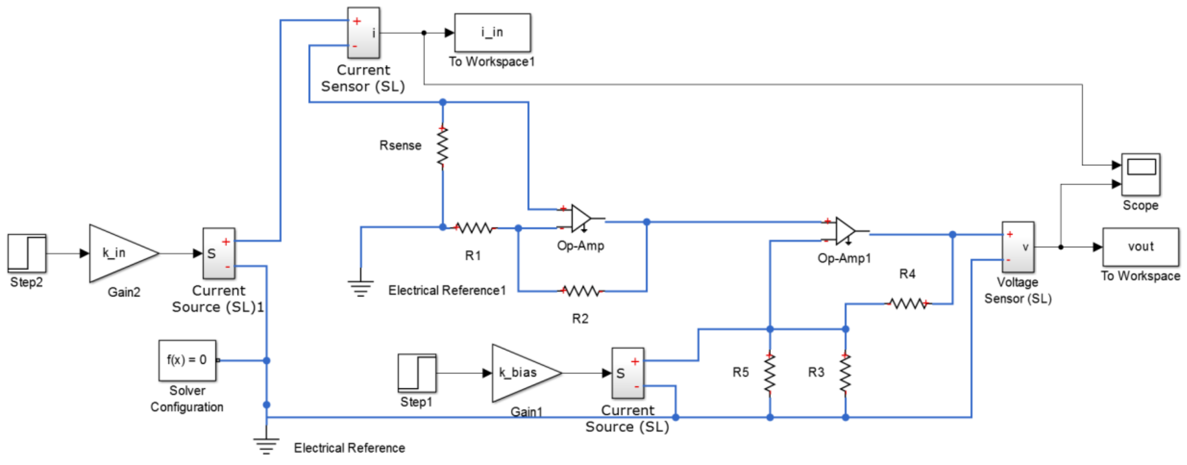


Figure 3.1. Circuit Diagram for DC Current Simulation in Matlab Simulink

We built the system shown in Figure 3.1 to see if the voltage output would be centered at 1.65V throughout the input DC current range of 0 to 10A. It was simulated in Matlab Simulink, and a Matlab script was used to modify the values of *k_in* and *k_bias* at the beginning of each simulation iteration. Figure 3.2 shows the output voltage of the second op-amp is clearly around 1.65V. Due to rounding off error in the *k_bias* calculations, the DC output increases slightly from 1.653V to 1.664V from the input current of 1A to 10A. This proved that the resistor value calculations were indeed correct and that the op-amp configurations could measure current through the entire input current range through this windowing technique.

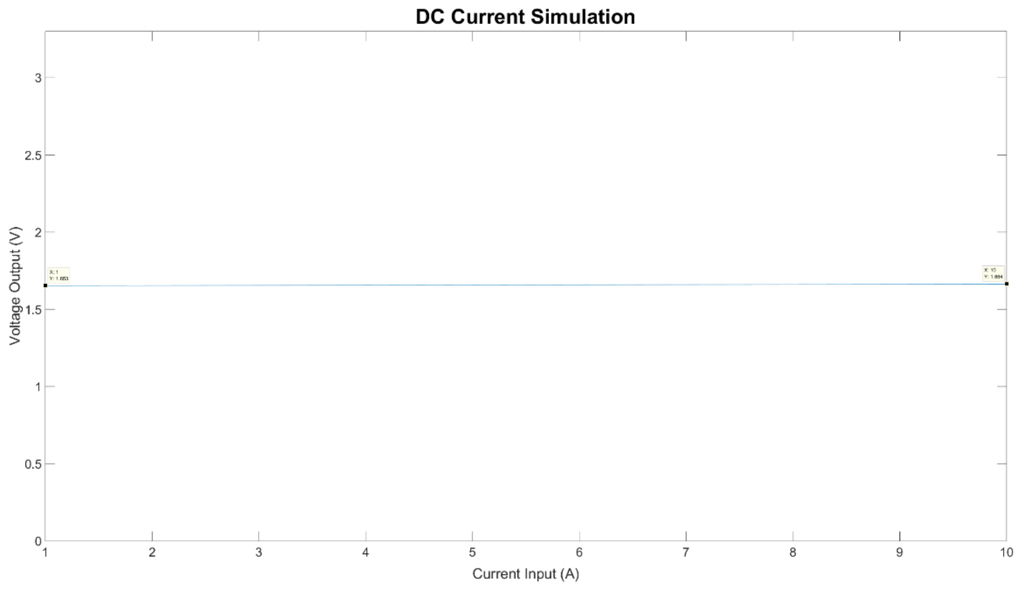


Figure 3.2. Voltage Output Plot vs DC Current Input for Matlab Simulink Simulation

We next sought to determine if current ripple was scaled to fit half of the ADC range, as well as follow the window set by the current bias. The system shown in Figure 3.3 was built in Simulink and then simulated. This time, a sine wave was placed at the input with a DC offset determined by the *k_in* parameter.

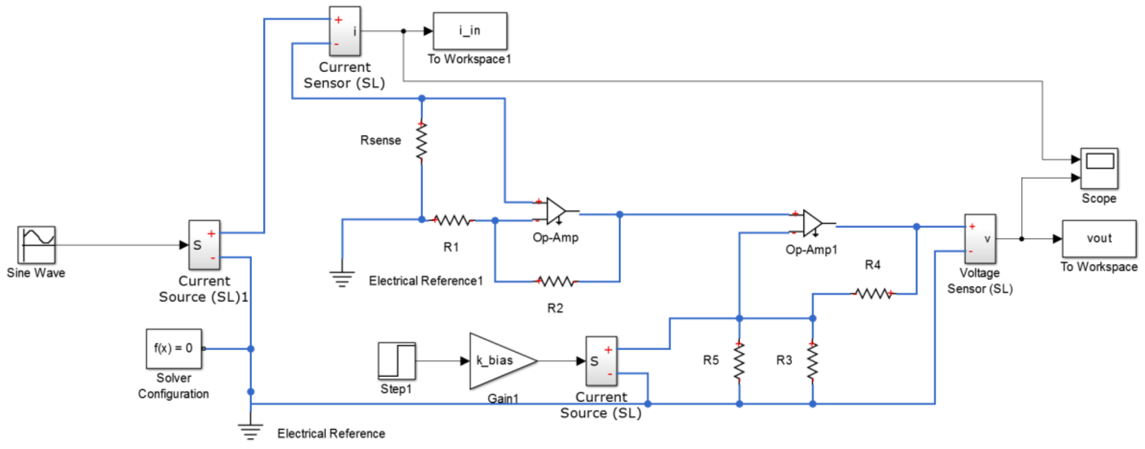


Figure 3.3. Circuit Diagram for Current Ripple Simulation in Matlab Simulink

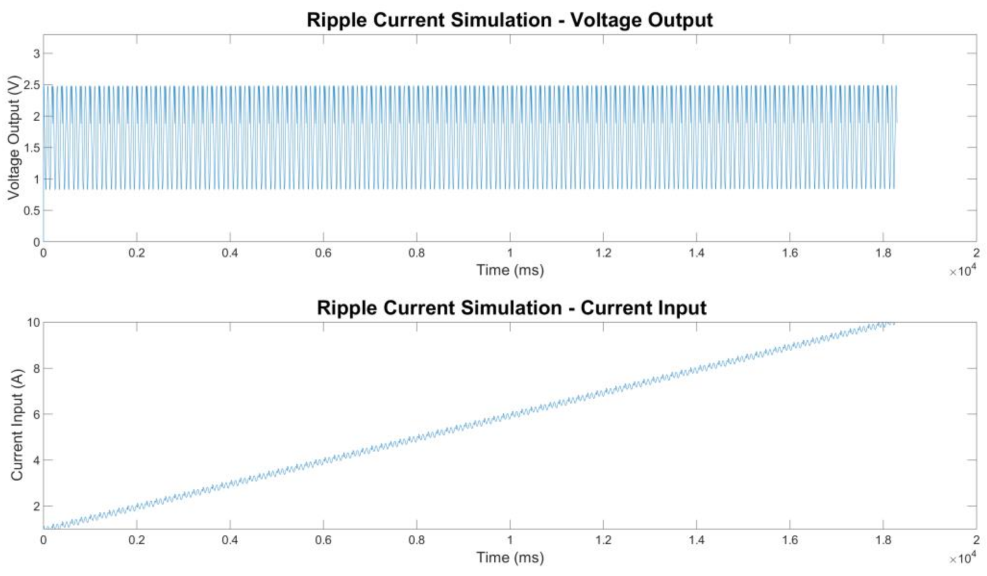


Figure 3.4. Voltage Output Plot (top) and Current Ripple Input (bottom) vs Time for Matlab Simulink Simulation

The sine wave input had an amplitude of 0.125A which gives a ripple of 0.25A as specified in Table 2.1. The output voltage ranged from 0.847V to 2.498V, which amounts to 1.646V, which is about half of the input ADC range, as calculated in the earlier sections. The input current ripple was scaled and varied based on the current bias. We have shown through simulation that despite the increase in current, the voltage output remains in the window for the entire input current range.

4. Programming Cypress

The CYPRESS PSOC Creator gives the programmer the option of dragging and dropping components into a schematic as shown above in Figure 4.1. The parameters of each of the components can be specified using a GUI, hence reducing the use of C code to initialize registers. How each of the components behave is then determined by the main function and C code. The resistors cannot be programmed in the microcontroller, hence we needed to arrange the resistors on a breadboard. We then programmed the output pins of the microcontroller and wired them to the appropriate nodes accordingly.

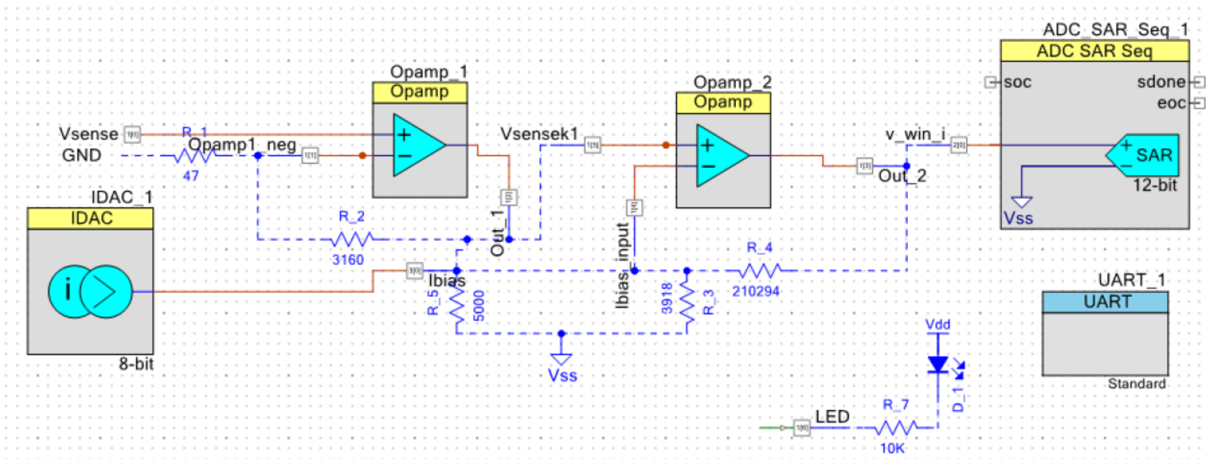


Figure 4.1. PSOC Creator Circuit Diagram

4.1 Op-amp

The PSOC4 boasts two highly configurable programmable op-amps. We needed high output current so as to be able to drive the output pins. The PSOC 4 could be programmed to output current at 1mA or 10mA. Hence we selected the 10mA option. We selected the op-amps to run at high power mode for high slew rates and a high gain bandwidth. A high slew rate is needed due to the ripple measurements the circuit is required to perform. The high gain bandwidth is needed since the gain of both op-amp configurations are large. We chose high stability, despite the slower speed, due to its high compensation which stabilizes the voltage output of both op-amps.

4.2 Analog-to-Digital Converter (ADC)

The PSOC4 also has a highly configurable analog-to-digital converter, with up to 4 channels. However, we only need one channel for our purposes. The ADC would only begin a conversion when needed and hence we programmed it to be hardware triggered. Based on the technical reference manual, for a single ended input ADC with a range that starts from 0V, having the single ended negative input connected to Vss and the voltage reference connected to an external voltage reference, which was set at 3.3V, would provide the ADC with the best signal-to-noise ratio. Initially, we chose a slow sample rate and high number of samples averaged, so as to ensure stability in readings, and the sampling rate will be optimized further at later stages. Hence a 1MHz sampling rate was chosen, with 256 samples averaged. We also lengthened the acquisition time to 500µs for greater stability. Finally, the ADC's resolution was 11 bits which meant each bit change amounted to a 1.612mV change in output voltage, in turn translating to a 0.244mA change in current sensed.

The ADC can generate interrupts based on window limits or if the ADC saturates. We set the window limits to 1.15 to 3.044V, which is about 200mV larger than 1.65V, the expected peak to peak scaled voltage output waveform. More about how the algorithm leverages on this limit interrupts will be explained in subsequent sections.

4.3 Current Digital-to-Analog Converter (IDAC)

To provide the bias, a current DAC was used. The DAC will be in parallel with *R5* so as to create the Thevenin voltage bias so as to have the same bias equations as in [1]. We set the IDAC to a resolution of 8 bits which translates to 1.2µA per bit.

4.4 . Pins

All pins are programmed to be analog, as we are dealing completely with analog signals.

4.5 LEDs

An RGB LED was programmed to indicate when the voltage signal exceeded the limits by lighting up in red.

4.6 UART

UART was programmed to have a baud rate of 9600bps. The buffer size is set to 8 bits (1 byte) with 1 stop bit. With the buffer size as such, the values of the *bias* and the converted ADC reading, *mvolts* must be sent to the first in first out buffer (FIFO) one byte at a time. We wrote some C code to ensure that only 1 byte is being sent to FIFO at a time and this can be viewed in the Appendix.

4.7 Code Algorithm

Figure 9.2 shows a flowchart of the code algorithm incorporated into the circuit.

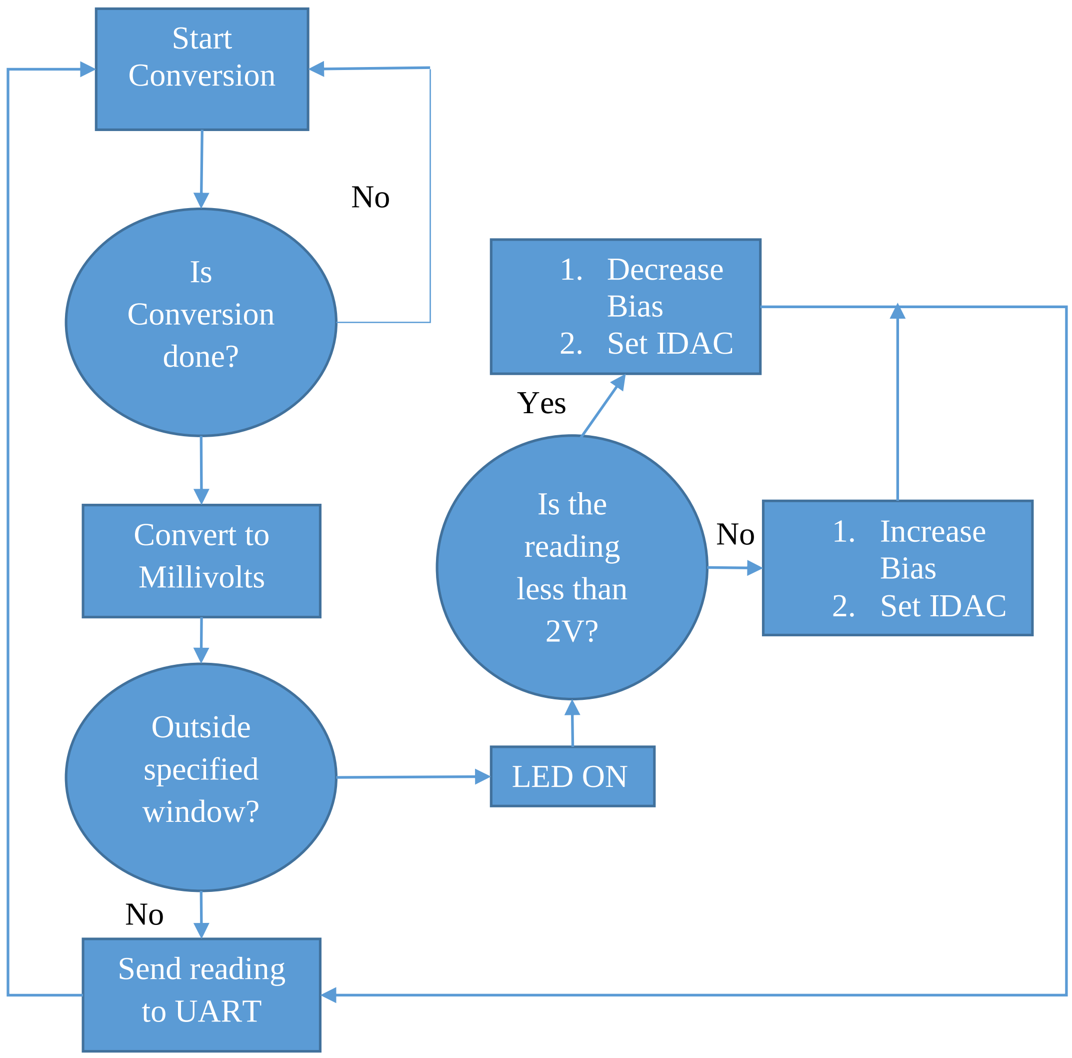


Figure 4.2. Flowchart of Code Algorithm

The ADC is hardware triggered, and hence, each loop begins with a function call to start the ADC conversion process. A buffer state is created to ensure that the ADC conversion process is completed before further code is executed. The code then calls a function that converts the ADC reading into millivolts so that we do not have to deal with hexadecimal interpretation. A flag will be generated if the ADC reading falls outside the window. If there is no flag, then the reading is sent to UART and the loop restarts. If there is however, then a simple "if" condition checks if either the upper or lower limit is reached and adjusts the bias current accordingly so as to shift the window. The bias is adjusted by only one bit, which equates to 1.2µA, and so that the window shift is as small as possible to ensure finer window adjustment.

5. Initial Measurements and Debugging

The circuit in Figure 5.1 was set up on a breadboard for initial measurements.

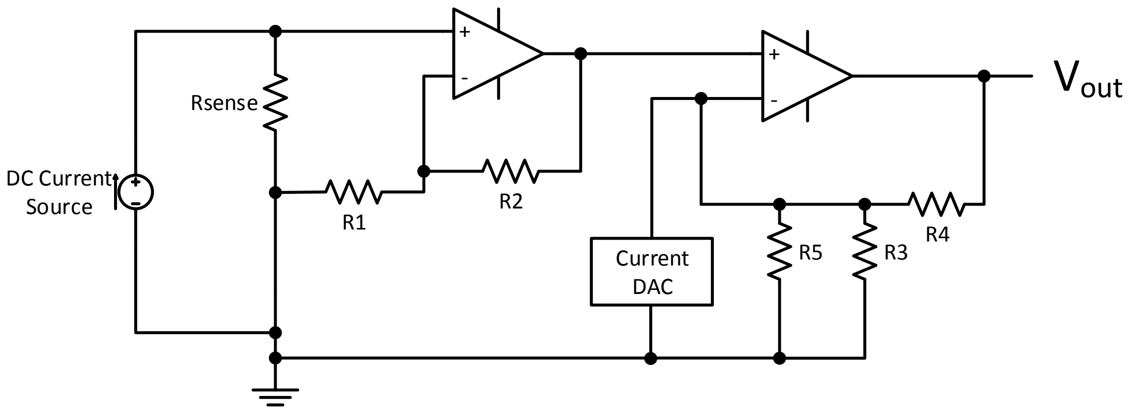


Figure 5.1. Test Circuit with Cypress Op-amps

A DC current source was used to bias the sense resistor. Rsense was initially a wire measured out to be approximately 1mΩ using a 4-wire measurement. However, after the circuit was set up, no voltage was registered at the output.

5.1 Initial Debugging

The Cypress op-amps were immediately suspected, as upon probing, the voltage output from the initial state was only 16mV, as compared to a 68mV expected output at 1A input current. To debug the cypress op-amps, the following circuit shown in Figure 5.2 was used.

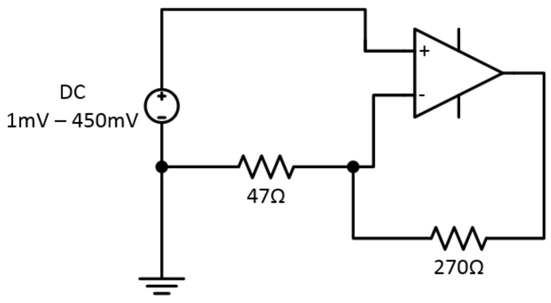


Figure 5.2. Debugging Circuit for Initial Op-amp Stage

To rule out gain bandwidth issues, we decided to decrease the gain of the circuit by a factor of 10, and hence the 3160Ω was switched out for a 270Ω resistor so that the gain would be 6.744. Table 5.1 below shows the results of varying the input from 1mV to 450mV.

Table 5.1. Measurements for Initial Gain Debugging Circuit 1st Op-Amp

DC (V)	Vout (V)	$Gain_{Calculated}$	$Gain_{Ideal}$	Gain Error (%)
0.001	0.003	2.920	6.745	382.47
0.010	0.006	0.561	6.745	618.37
0.020	0.068	3.416	6.745	332.87
0.030	0.147	4.900	6.745	184.47
0.040	0.225	5.612	6.745	113.29
0.050	0.299	5.980	6.745	76.47
0.060	0.374	6.232	6.745	51.30
0.070	0.446	6.374	6.745	37.04
0.080	0.518	6.469	6.745	27.59
0.090	0.588	6.536	6.745	20.91
0.100	0.658	6.579	6.745	16.57
0.110	0.727	6.609	6.745	13.56
0.150	1.000	6.667	6.745	7.80
0.200	1.338	6.691	6.745	5.42
0.310	2.077	6.699	6.745	4.56
0.410	2.747	6.701	6.745	4.37
0.450	3.015	6.700	6.745	4.51

The voltage measurements for the input DC voltage of 1mV explains the issue of having no voltage amplification at the current input of 1A, since the voltage output was a mere 3mV. However, as the DC input voltage increases, we begin to see that the gain converges towards the correct value. However, the DC input in which the gain approaches an acceptable value, is in the region of 200mV, which corresponds to 200A of input current.

The second op-amp was also tested using the same circuit and Table 5.2 shows the voltage measurements acquired. Note the difference in $Gain_{ideal}$ because different resistors of the same value were used as it was done on a separate occasion.

Table 5.2. Measurements for Initial Gain Debugging Circuit 2nd Op-Amp

DC (V)	Vout (V)	$\text{Gain}_{\text{Calculated}}$	$\text{Gain}_{\text{Ideal}}$	Gain Error (%)
0.001	0.042	42.210	7.059	497.94
0.002	0.048	24.200	7.059	242.81
0.003	0.055	18.267	7.059	158.76
0.004	0.061	15.350	7.059	117.45
0.005	0.068	13.526	7.059	91.61
0.010	0.100	10.000	7.059	41.66
0.015	0.133	8.867	7.059	25.60
0.020	0.166	8.300	7.059	17.58
0.025	0.199	7.960	7.059	12.76
0.035	0.267	7.629	7.059	8.07
0.045	0.333	7.400	7.059	4.83
0.055	0.400	7.273	7.059	3.02
0.075	0.537	7.165	7.059	1.50
0.100	0.710	7.100	7.059	0.58
0.200	1.412	7.060	7.059	0.01
0.300	2.117	7.057	7.059	-0.04
0.400	2.823	7.058	7.059	-0.02

The results shown for the second op-amp showed similar gain calculations, where they converged to the correct value as the input increased.

5.2 Using an External Op-amp

To ensure that the circuit design worked, and that the op-amps were the points of failure, external op-amps were used. As a starting point, we chose the op-amps used in [1] to replace the Cypress Op-amps which were the OPA4364. Initial simulations were also performed to ensure that the Op-amps could work at such a low sense resistor and high gain. The simulation of a model of OPA4364 was performed over LTSPICE and the circuit diagram of which is shown in Figure 5.3.

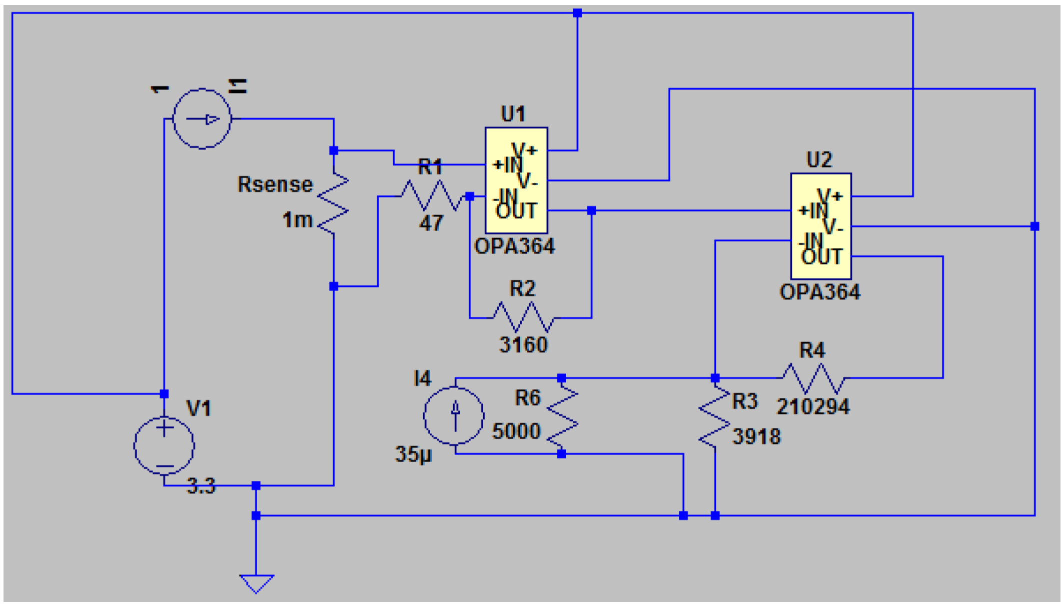


Figure 5.3. LTSPICE Simulation of Circuit with OPA364

Since the op-amp model was not ideal, it was difficult to determine the correct current bias at a given current input. Hence a DC sweep of the current bias source was performed. We initialized I4 to be 23.5µA and varied by 1.2µA up to 50µA. After which, data cursor was used to determine the current bias value in which the output of the circuit was around 1.65V. With a slight error value, we determined that the current bias should be initialized to slightly below 34.3µA.

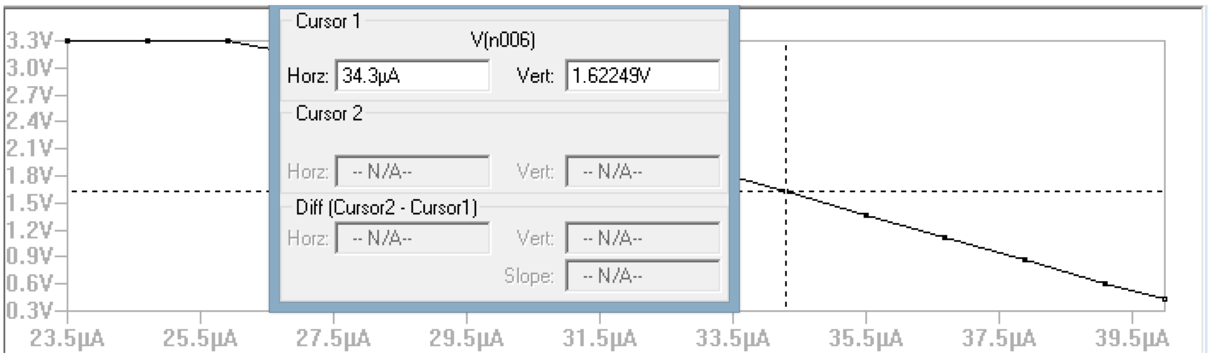

Cursor 1
V(n006)
Horz: 34.3µA Vert: 1.62249V
Cursor 2
Horz: -- N/A-- Vert: -- N/A--
Diff (Cursor2 - Cursor1)
Horz: -- N/A-- Vert: -- N/A--
Slope: -- N/A--

Figure 5.4. LTSPICE Simulation Plot of Circuit with OPA364 with DC Sweep of I4 Current Source from 23.5µA to 50µA at 1.2µA Intervals

Using the information obtained above, we initialized the current bias to 34.15µA to obtain a voltage output signal at 1.65V as shown in figure 5.5 below.

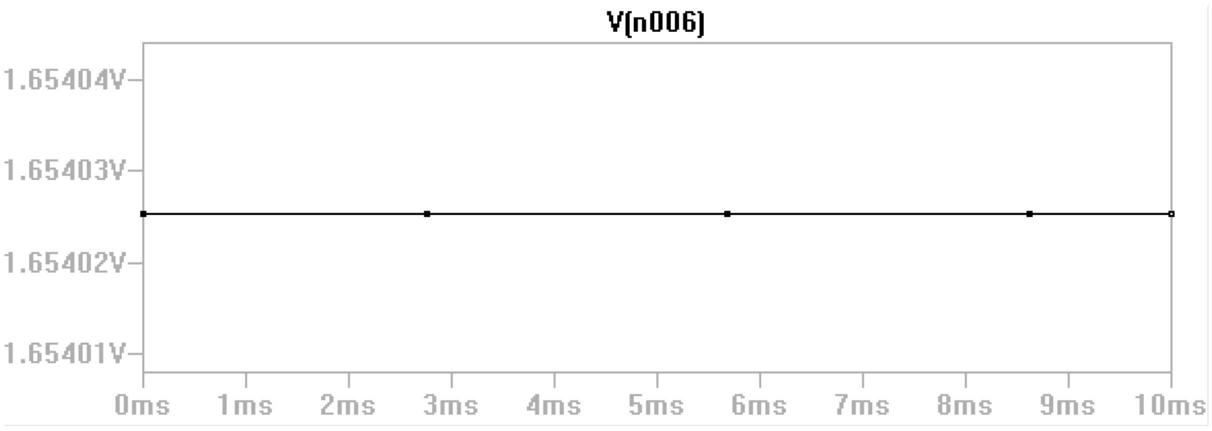


Figure 5.5. LTSPICE Simulation Plot of Circuit at Current Bias of 34.15µA

We have proven that OPA364 could be used for our application and hence we proceeded to breadboard the op-amps with the same setup as in previous sections. However, the same issue occurred, where at 1A, no voltage output was detected at the initial gain stage. Hence, other factors were looked at. After further analysis, we then suspected that the minimum offset voltage of the op-amps could explain why at such low sense voltages, no output voltage was attained. It was discovered that the voltage offset of OPA364 was 0.5mV. Voltage offset can be viewed as the voltage across the inverting and non-inverting input to make the output voltage 0V. Having a voltage offset of 0.5mV could prove too close to 1mV and hence the op-amps could 'think' that it was sensing 0V and hence not output anything. We then looked at the input offset voltage of the Cypress op-amps and discovered that it was 1mV which also explained the behavior in the previous instances.

5.3 Alternative External Op-amps

We were then aware that the input voltage offset proved to be a critical electrical characteristic of the op-amp amongst other factors, and so, eventually we settled on OPA330 as it had comparable electrical characteristics as OPA364, but a lower input offset voltage at 50µV.

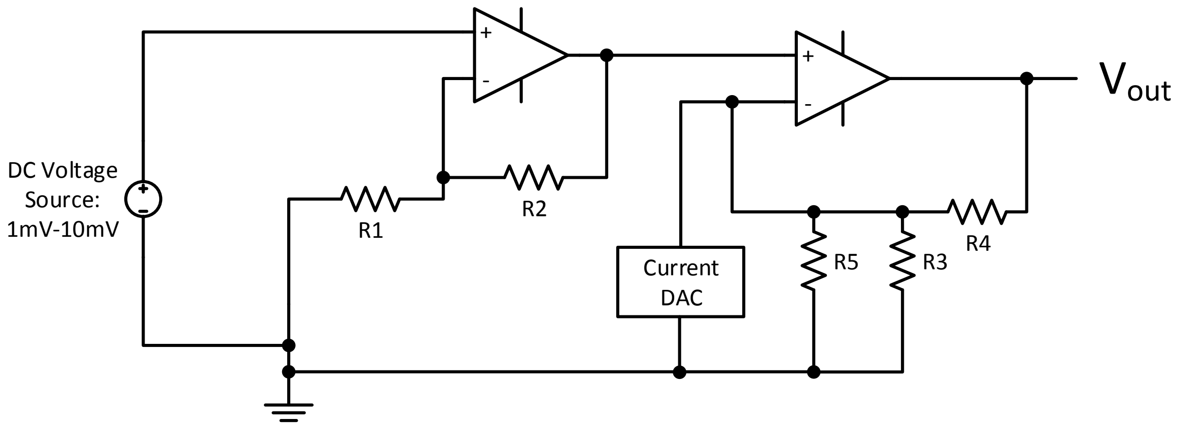
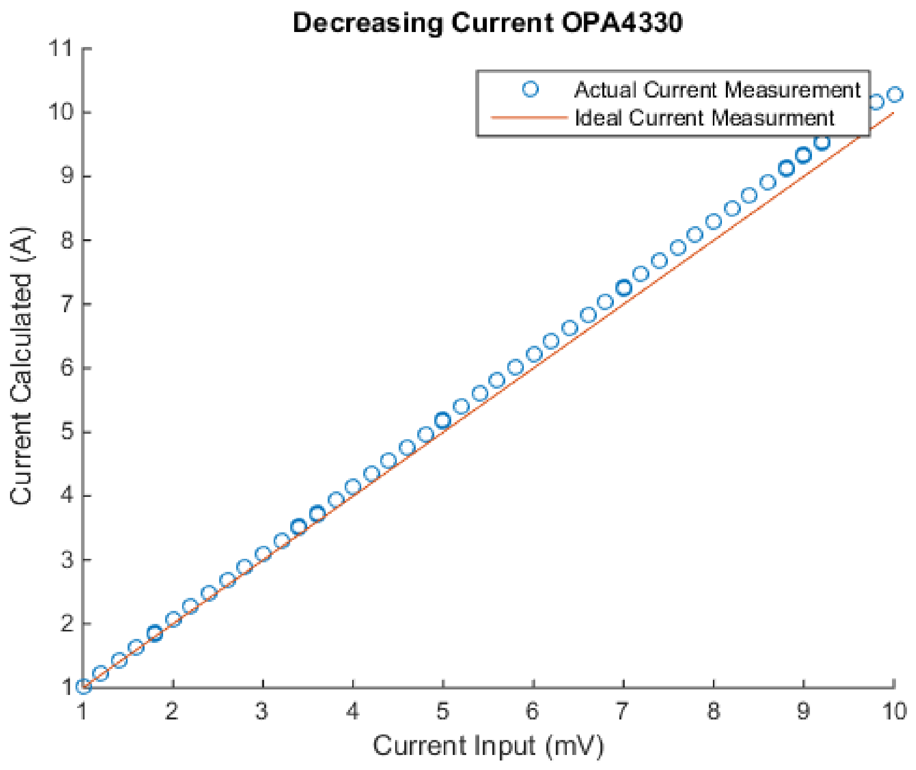


Figure 5.6. Testing Circuit with External Op-amps

As seen in Figure 5.6, the DC voltage input was allowed to vary from 1 to 10mV and the ADC readings were manually taken down at an input interval of 0.2mV. Measurements were only made when the current bias remains constant for three consecutive ADC readings. In this way, the ADC readings are the settled voltage output and not the intermediate sensed current measurements when the bias is adjusting while the signal is outside the window. Measurements were made for both increasing and decreasing current, and plots were made via Matlab to observe the trend in the ADC readings. Figure 5.7 shows the results of those measurements. The current calculated was based on Equation 2.8 and represents the y-axis.

The solid line indicates the expected current plot in ideality and the scatter plot indicates actual data points measured and plotted at different input DC voltages. The key takeaway here is that both plots are monotonic. Digital calibration would be able to process the slight discrepancies between the real and the ideal current measurements.

These plots proved that the code works, and that the resistor values chosen allowed current measured across the entire range.



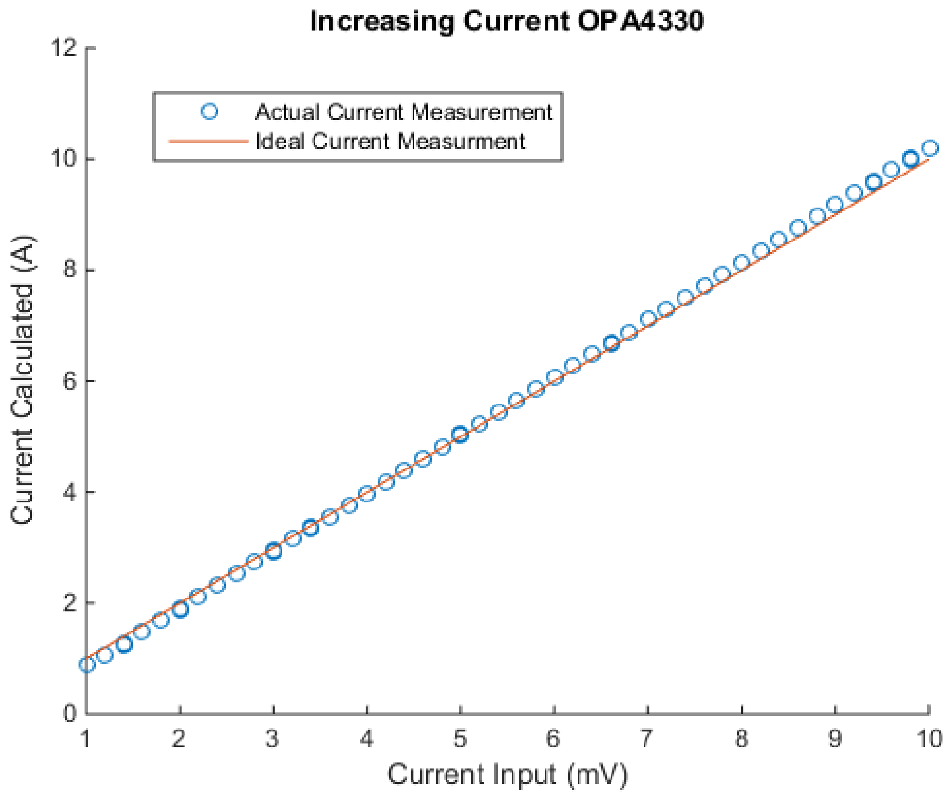


Figure 5.7. Plots for OPA4330 for Current Measurements

6. Cypress Op-amp Workaround

The ultimate goal in this research was to use the Cypress op-amps. Hence, after proving that the code and resistor values worked, we needed to ensure the full-functioning of the circuit with the Cypress op-amps.

6.1 Creating a Fixed Offset through High Side Current Sensing

From previous sections, we showed that the input voltage offset was a problem, hence it was decided to attempt to force an offset at the input terminals of the op-amps. In order for this to be possible, we needed to sense current at the high side instead and hence, set up the circuit as shown in Figure 6.1.

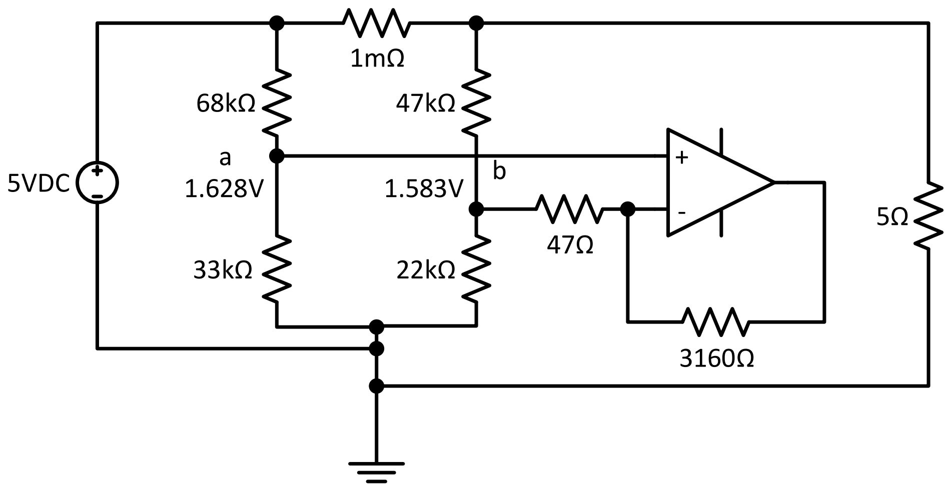


Figure 6.1. Input Offset Voltage Forcing Circuit

As proof of concept, we placed a 5V DC source to represent a panel, and placed a 5Ω resistive load so as to force 1A through the sense resistor. As seen in Figure 6.1, the expected voltages at nodes *a* and *b* were 1.628V and 1.583V respectively causing a difference of 45mV. However, both in simulation and on the breadboard, the inverting input forced the voltage difference to be 0.5mV. The simulation result is shown in Figure 6.2.

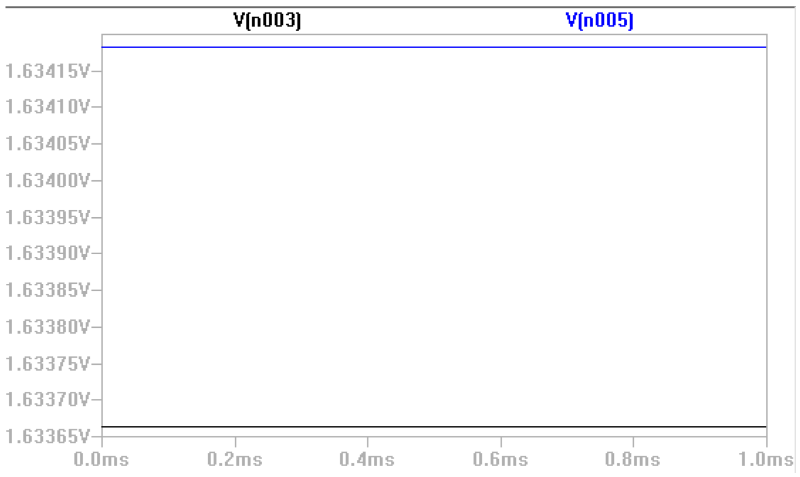


Figure 6.2. Simulation Result for Voltage Offset Divider

It was suspected that the resistors values were too high and hence too little current could have flowed to the inputs of the op-amp. However, the resistor values were chosen in the tens of kiloOhms range so as to reduce power loss. Minimal effort was expended in debugging this circuit since this voltage divider added power more power loss (an added 0.6mW) to a current sense solution whose sole purpose was to sense virtually losslessly. Further, high side sensing would make the voltage dividers panel dependant which would hence alter the resistor values.

6.2 Re-characterization of Cypress Op-amp 2

The Cypress was re-plotted at the required gain as the data shown table 5.2 indicated that one of the op-amps could detect an input of 1mV. The measurements were done and are shown in Table 6.1. We noticed that if we subtracted the voltage at 0mV DC input, then our gain error falls to the region of 7%, which would not pose too much of a problem after digital calibration.

Table 6.1. Measurements for Re-characterization of 2nd Op-Amp

DC (V)	Vout (V)	Vout-Voff	$Gain_{Calculated}$	$Gain_{actual}$	Gain_error(%)
0	0.327	0	-	-	-
0.001	0.394	0.067	67.300	72.083	6.64
0.002	0.462	0.135	67.700	72.083	6.08
0.003	0.529	0.202	67.467	72.083	6.40
0.004	0.595	0.268	67.000	72.083	7.05
0.005	0.662	0.335	66.960	72.083	7.11
0.010	1.000	0.673	67.320	72.083	6.61
0.015	1.339	1.012	67.480	72.083	6.39
0.020	1.678	1.352	67.580	72.083	6.25
0.025	2.013	1.686	67.448	72.083	6.43
0.030	2.357	2.030	67.667	72.083	6.13
0.040	3.045	2.718	67.953	72.083	5.73

6.3 Creating the Workaround

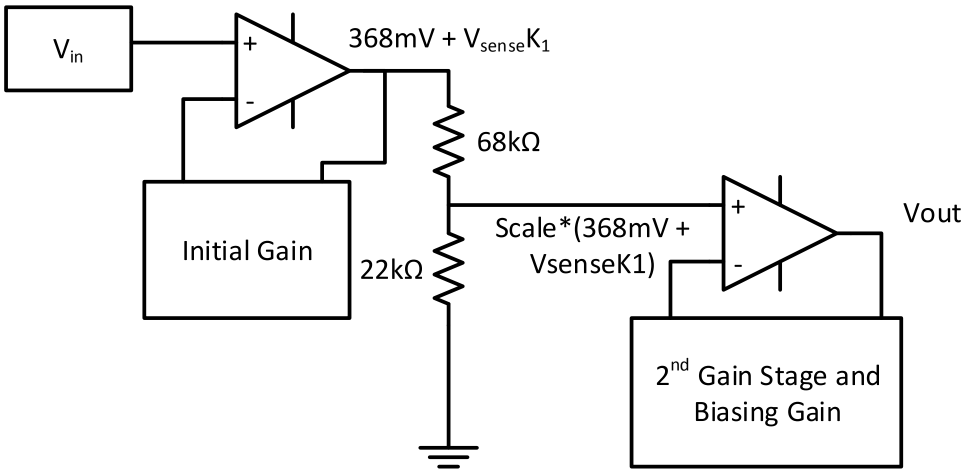


Figure 6.3. Cypress Workaround

From, the previous section and in the earlier chapters, we showed how the two op-amps behaved very differently despite the same setup. For the set of PSOC4 that we had at our disposal, we decided to leverage on the two very separate idiosyncrasies of the op-amps to make the circuit work. One op-amp only started amplifying at voltage inputs above 40mV, and the other op-amp amplified linearly, but had a 360mV offset. Hence, the initial gain stage was used for the op-amp with the voltage offset, and then scaled downwards using a voltage divider to a reasonable value as an input to the second stage. As the same characteristics were observed in other Cypress op-amps, prima facie, it seemed like a viable option.

With the new scale, the circuit will behave quite differently, and the relationship between the input and output of the initial gain stage is shown in Figure 6.4.

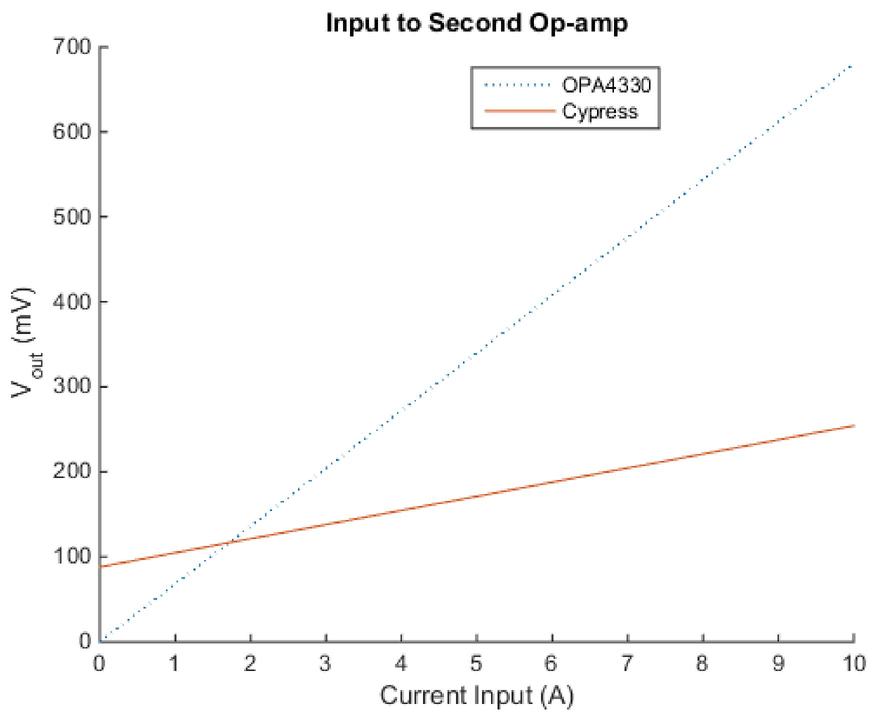


Figure 6.4. Input to Second Stage

From the graph above, it should be expected that the maximum biasing current should be less for the Cypress setup than for the OPA4330. Also, with the new scale, Equation 2.8 will be different and can be found by Equation 6.1 below.

$$I_{sense} = \frac{V_{out}+I_{bias}R4}{(\frac{R7}{R7+R8})\left(1+\frac{R2}{R1}\right)\left(\frac{R4}{R3}+\frac{R4}{R5}+1\right)R_{sense}} - \frac{V_{off}}{(1+\frac{R2}{R1})(R_{sense})} \tag{6.1}$$

The same measurements process as in the previous chapter was performed with this new set up. It should also be noted that a low-pass filter is placed at the input of the ADC as well as the input of the inverting terminal of the op-amp in the second stage. This was to filter out any noise inherent in the Cypress op-amps.

Figure 6.5 shows the plots of increasing and decreasing current using the Cypress op-amps. The y-axis shows the current calculated after the sweep using Equation 6.1. We recorded the values of the *bias* variable as well as the converted ADC reading, *mvolts* and we used these values to calculate the current. The bias current is calculated based on Equation 6.2 below.

$$I_{bias} = \frac{bias}{255} * 0.000306 \tag{6.2}$$

255 is the maximum value bias can take since the DAC has an 8 bit resolution, and 0.000306 is the maximum current output from the DAC.

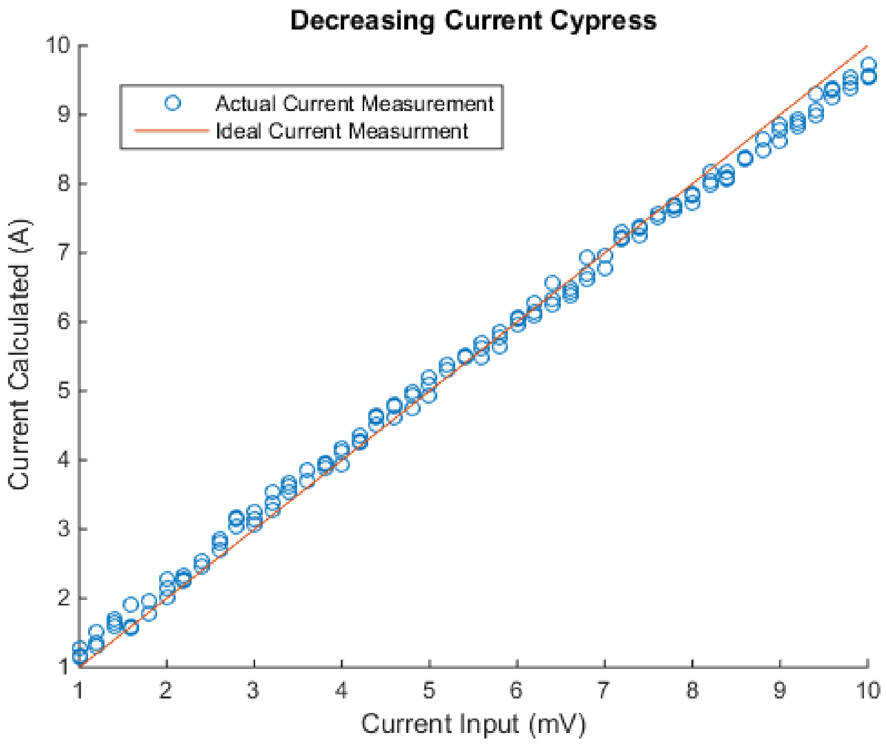


Figure 6.5. Decreasing Current Measurement Plots for Cypress Op-amps

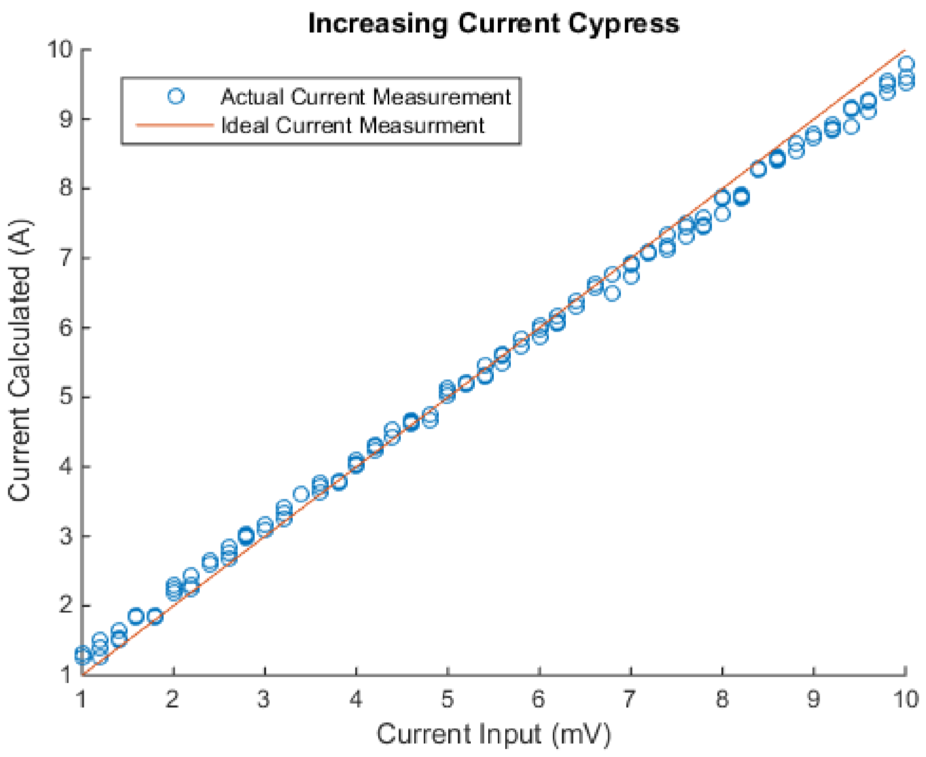


Figure 6.6. Increasing Current Measurement Plots for Cypress Op-amps

There was a lot more flutter in the ADC readings as compared to the OPA4330, indicating that the Cypress op-amps are not as stable as the OPA4330. However, the general trend follows very closely to the ideal case. Digital calibration can be performed to account for the flutter in ADC reading. Further, these measurements were performed on a circuit built on a breadboard which tend to introduce noise into signals.

6.4 New Circuit Parameters

With the added scale, the gain of the circuit changed and can be found by Equation 6.3 below:

$$K_{gain} = K_1 K_2 K_{scale} \tag{6.3}$$

Where,

$$K_{scale} = \frac{R7}{R7 + R8}$$

The rest of the resistor values remained the same and are shown in Table 6.2 and the gain values are summarized in Table 6.3.

Table 6.2. Resistor Values with K_{scale}

Resistor	Value (Ω)
R1	46.5
R2	3160
R3	3920
R4	221000
R5	4990
R7	22000
R8	68000

Table 6.3. Gain Values with K_{scale}

Gain	Value (V/V)
K_1	68.23
K_2	96.73
K_{scale}	0.244
K_{gain}	1610

It is thus apparent that the scale decreases the gain of the circuit, which effectively increases the range of current that can be measured, but reduces the resolution of the measured signal.

7. Printed Circuit Board (PCB) Implementation and Input Sweeps

A 2-layer PCB was designed using Eagle in attempt to eliminate noise in signals and hence reduce ADC reading flutter. Both the circuit for the Cypress op-amps and the OPA2330 were built on the same board. Note that OPA2330 is a dual version of the quad package of OPA4330. The figure below shows the PCB connected to the Cypress microcontroller. The actual schematic and Eagle layout can be found in the appendix. The PCB was also characterised and we observed that the voltage offset from the first op-amp was 300mV at 0mV input.

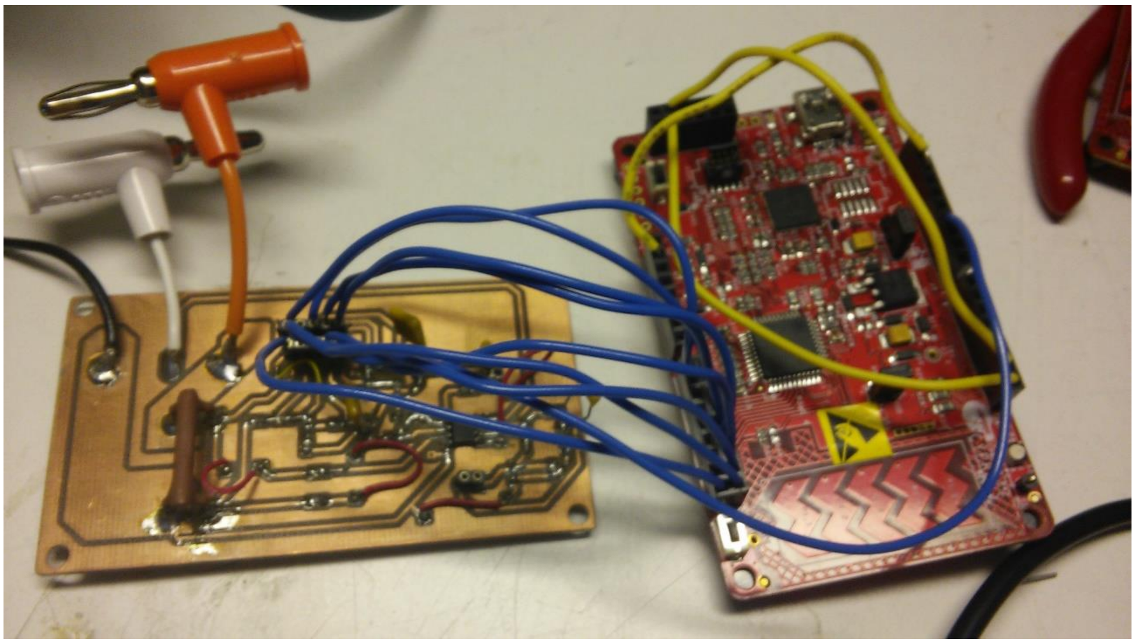


Figure 7.1. Picture of PCB with Cypress Pioneer Kit Microcontroller

7.1 Improvements in ADC parameters

Initial measurements showed that there was very little flutter in the ADC readings and hence we decided to quicken the acquisition and sampling rate for better processing since this current sense amplifier would eventually need to measure current ripple. The clock frequency was increased from 1Mhz to 15Mhz, with just two samples averaged per ADC conversion, further the acquisition time was reduced to 4 clock cycles, that is 266.67ns per acquisition and thus the conversion time is improved to 1.2µs. In this case, we are certain that a high frequency current ripple can indeed be measured with high enough resolution with these acquisition and conversion times.

7.2 Python Serial/UART Communication

In order to generate voltage sweeps, we used Python code to communicate with a Kiethley 2400 power sourcemeter through GPIB. Python was also used to set up serial communication using the UART protocol so that the *bias* and *mvolts* variables can be saved in a log file. We would then process these measurements to produce the current measurements digitally after the whole sweep was performed. An algorithm had to be set up such that the serial output from the microcontroller is sent only when the Python code is ready to receive the data. The python code of these sweeps can be found in Appendix C.

7.3 Input DC Voltage Sweep

The same set up shown in Figure 5.5 was used for the input voltage sweep, except that the sweep begun from 0mV and ended at10mV, with 0.1mV increments and then the input voltage was decreased from 10mV to 0mV. At each voltage input operating point, 10 ADC readings were taken to observe the flutter. Since an array of *bias* and *mvolts* were obtained, we could use those values to calculate the current measured using Equations 6.1 and 6.2 with *mvolts* converted to volts and used as the input to V_{out} in Equation 6.1. The results of the experiment is shown in Figures 7.2 and 7.3.

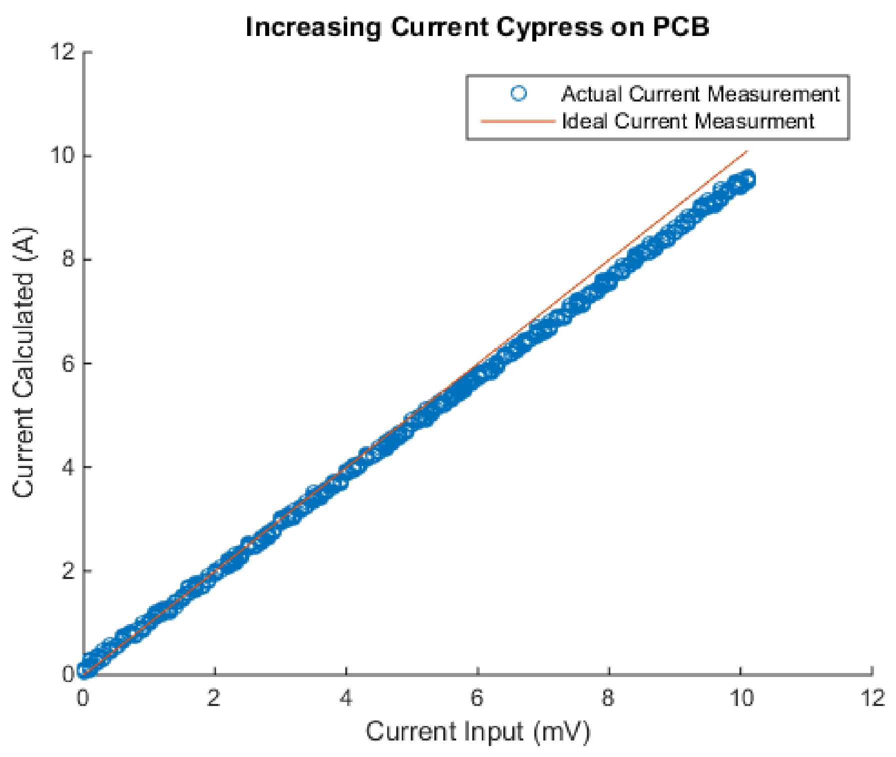


Figure 7.2. Current Measurements for an Increasing Input Voltage Sweep

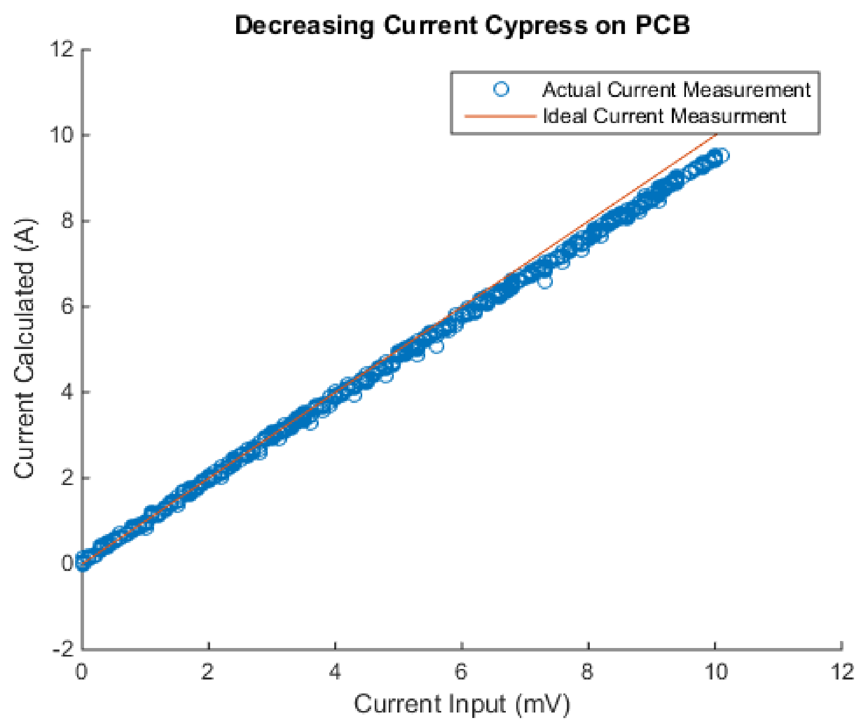


Figure 7.3. Current Measurements for a Decreasing Input Voltage Sweep

The results in Figures 7.2 and 7.3 show a vast improvement in ADC flutter as compared to the measurements performed on the breadboard. This can be attributed to the low pass filters as well as the PCB. The slight difference in slope between the measured plot and the ideal plot could be eliminated through digital calibration which will be performed once actual current is swept through a sense resistor instead of an input voltage sweep.

7.4 Input Voltage Ripple Sweep

Since current ripple is intended to be measured with this current sense amplifier, a current ripple sweep was performed using the Kiethley 2400 power sourcemeter. A voltage ripple of 0.250mV was generated by simply incrementing and decrementing the input voltage by 0.005mV so that a triangle wave is generated and the sweep was induced by increasing the DC offset of this ripple. It should be noted that the minimum DC offset was set to 1mV. Only one ADC reading is taken per input voltage level.

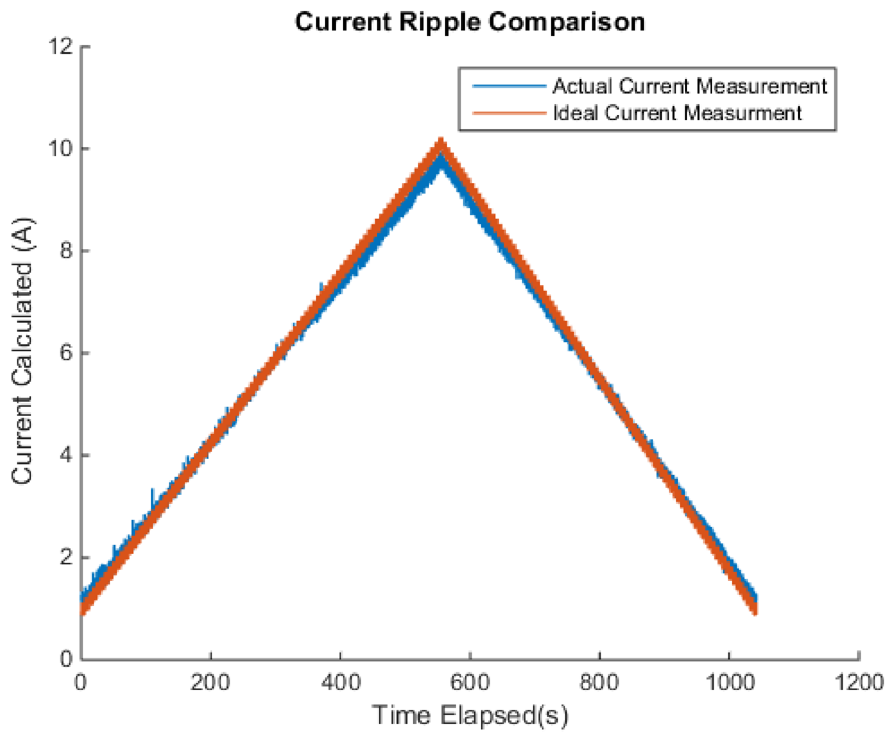


Figure 7.4. Current Measurements for an Input Ripple Voltage Sweep

Figure 7.4 show that the calculated current follows the trend of the ideal current measurement very closely, with a slightly less steep slope for the entire input range.

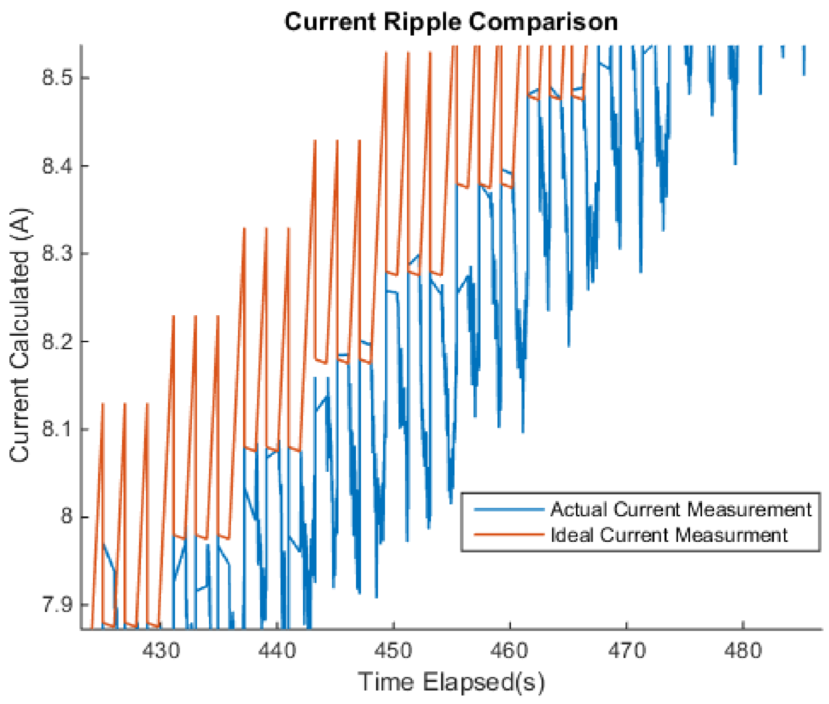


Figure 7.5. Zoomed in Current Measurements for Input Ripple Voltage Sweep

The zoomed in figure of the current ripple show that a 0.2A difference between the ideal and actual current at the 8A current level. Also, the calculated ripple shows a slightly different ripple pattern than the actual current, however, this was not too concerning as for DPP and MPPT, the change in current is more important than the absolute current level. Further, there was a slight timing issue in the Python Code at the time this measurement was taken, however, for the actual current ripple measurement shown in the succeeding sections, the bug was fixed and the pattern matched.

7.5 Input Current DC Sweep

The following set up shown in Figure 7.6 was used to sweep the current from 0.5 to 9.5A. A 1Ω resistor load was used to draw the current based on a 0.5 to 9.5V DC voltage sweep using the Agilent 6674a power supply with 0.1V increments and decrements. It should be noted that current was only limited to 9.5A as the fuse rating of the ammeter was 10A and hence we sought to limit the current to 9.5A.

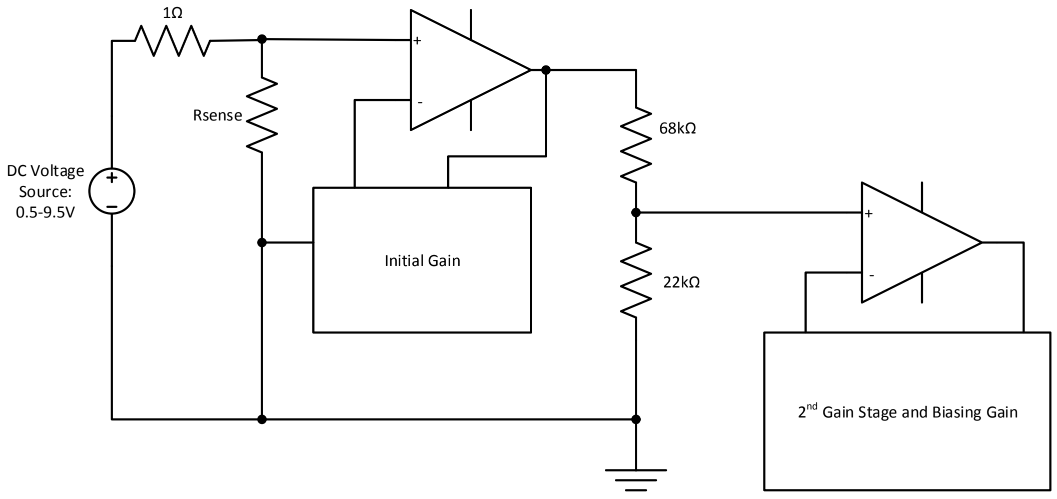


Figure 7.6. Circuit Diagram for Current Sweep Measurements

Figure 7.7 below shows the results of the current sweep. The plot in red shows the current measured from the ammeter placed just before Rsense and the plot in blue shows the current calculated based on Equation 6.1.

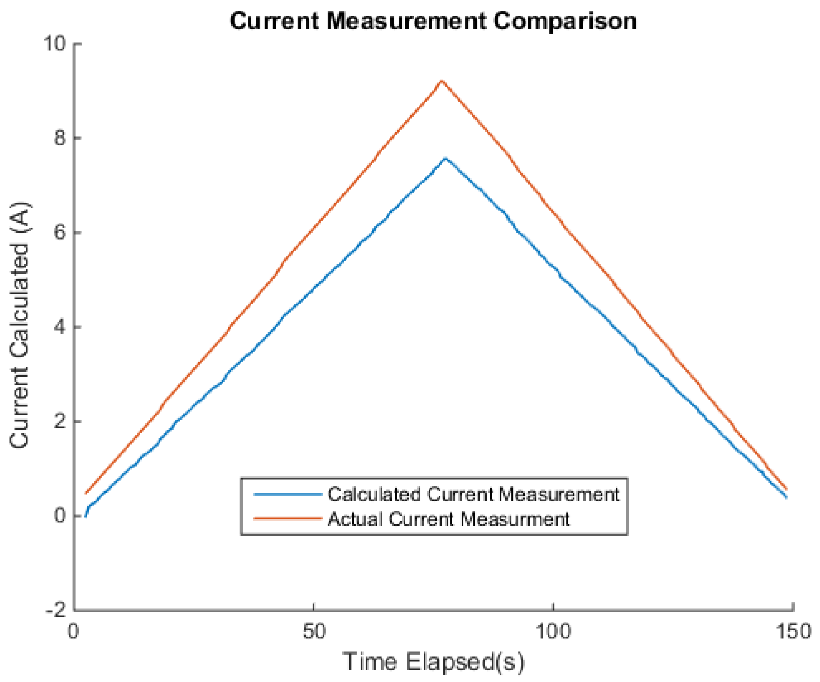


Figure 7.7. Current Measurement Comparison between Calculated Current and Actual Current

As current increases, the difference between the calculated current and the actual current increased linearly. However, it is clear that the current measurement response is monotonic which thus makes it easier to digitally calibrate and reliable especially for MPPT applications where the need to detect the change in current is more important than the absolute current levels. The Figure 7.8 below shows the effects of digital calibration.

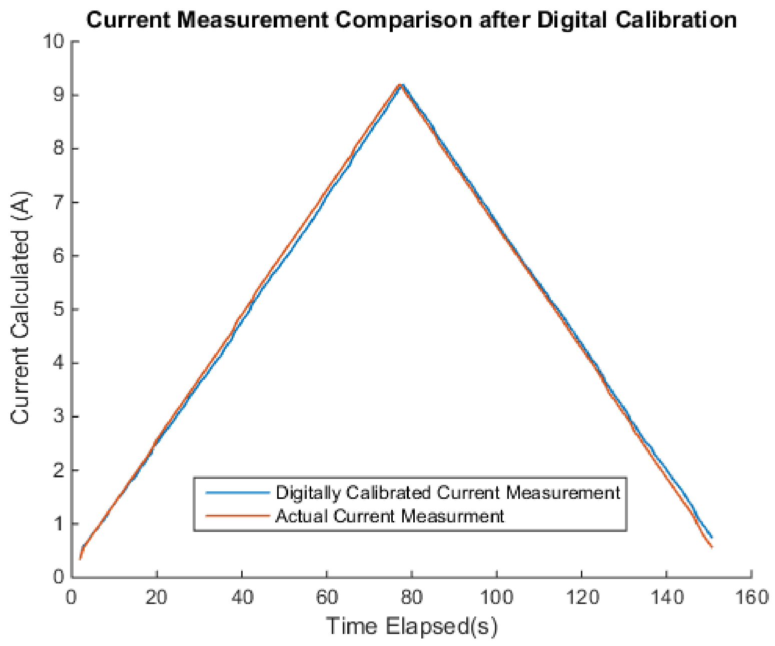
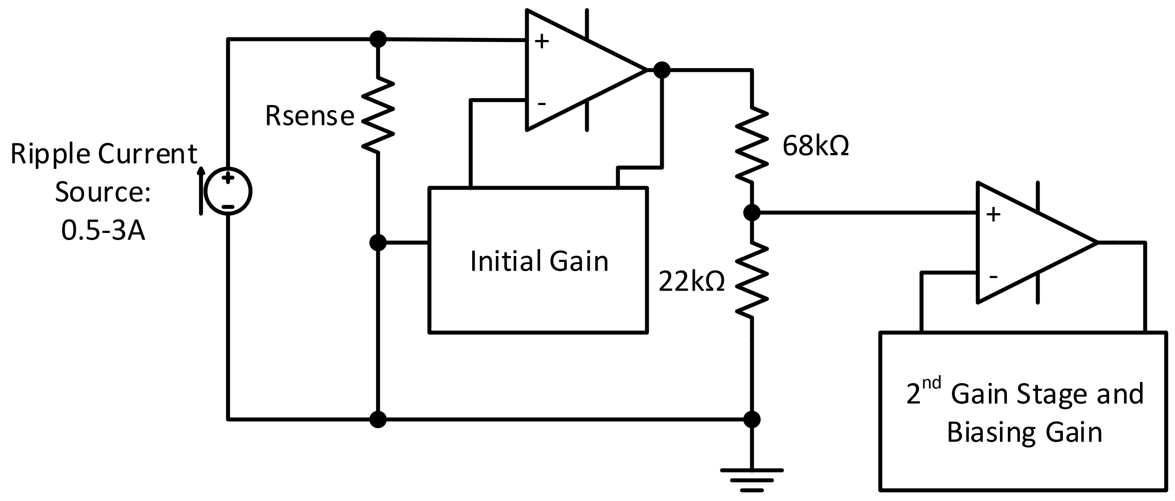


Figure 7.8. Current Measurement Comparison between Calculated Current After Digital Calibration and Actual Current

The digital calibration was performed in Matlab, though the algorithm can be easily applied in C via the main microcontroller or even the PSOC4 chip to process the current values. The characteristic relationship between the calculated current and the actual current was exploited where there was a linear increase in the difference between the two currents and hence as current increased, the calculated current was increased by an amount that increases linearly with increase in current. Refer to Appendix D1 for the Matlab code.

7.6 Input Current Ripple Sweep

Since the current that this circuit was designed to be measuring was actually a current ripple, a Kiethley 2400 sourcemeter was programmed to generate a ripple current in a triangular waveform and was allowed to vary from 1 to 3A with a ripple amplitude of 0.125A and a DC offset increment (and decrement) of 0.1A. The setup is shown in Figure 7.9, with a current source used instead of a voltage source, hence there was no need for a 1Ω load resistor.



Ripple Current Source: 0.5-3A
Rsense
+
-
68kΩ
22kΩ
Initial Gain
2nd Gain Stage and Biasing Gain

Figure 7.9. Circuit for Current Ripple Plots

The results of the sweep is shown in Figure 7.10. There is the same increase in the difference between the calculated current and the measured current, but in this case, the measured current is the reading from current output from the Kiethley 2400. There were some spikes in the calculated current that was not in the measured current, these spikes could be attributed to the transition where the errors occur as the bias is shifting and current is still being measured.

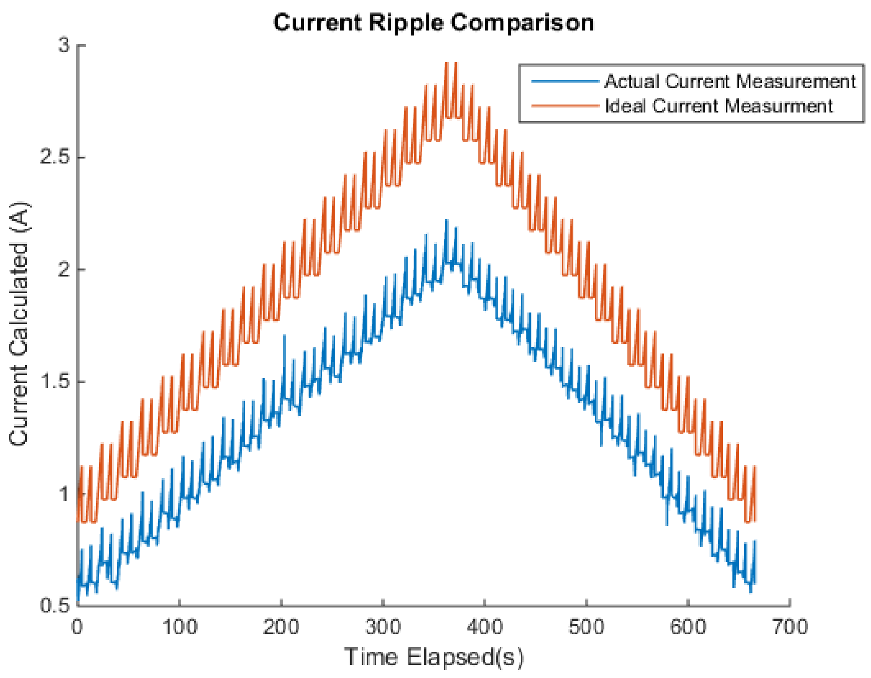


Figure 7.10. Current Ripple Plots with Kiethley 2400

Clearly, the calculated current required digital calibration similar to the previous section and Figure 7.11 shows the effects of digital calibration. A slightly different algorithm was used this time, where the current was shifted by a predetermined number based on the current level.

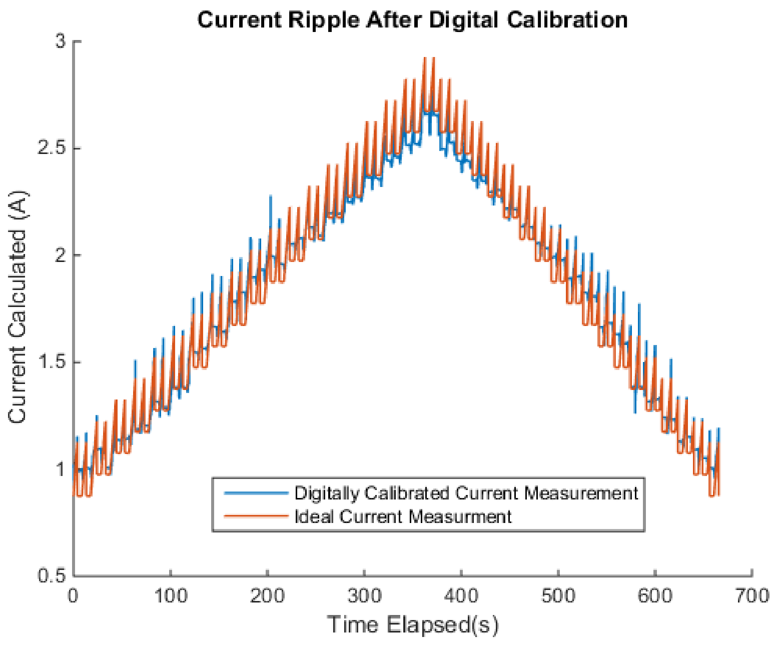


Figure 7.11. Current Ripple Plots after Digital Calibration

Figure 7.11 proves that once again, digital calibration is highly effective in matching the calculated current to the ideal current measurements. Figure 7.12 emphasises on this point where the current ripple after digital calibration follows the pattern of the current ripple produced by the current source very well. With the spikes attributed to the timespan during the changes in the ADC window. Digital calibration was performed on Matlab once again, though the algorithm can be easily implemented in C code and programmed into the microcontroller. Refer to Appendix D2 for the Matlab code.

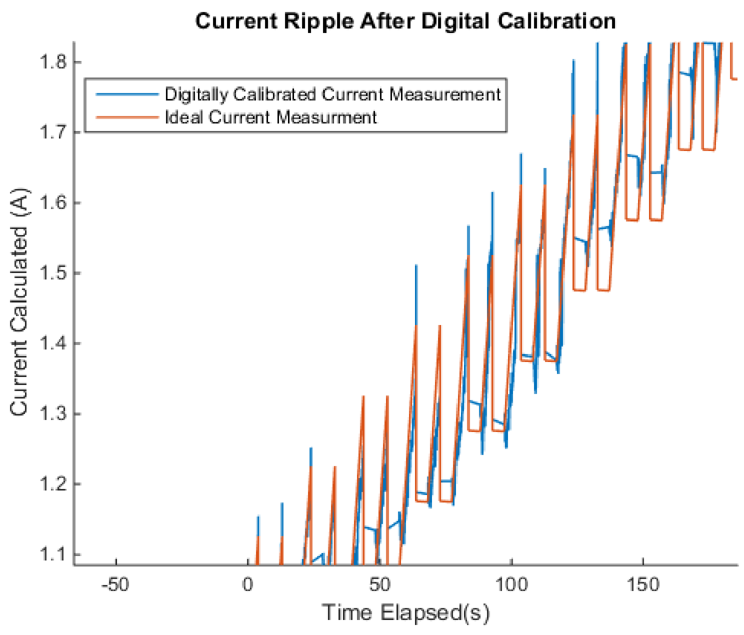


Figure 7.12. Zoomed in Current Ripple Plots after Digital Calibration

7.7 ADC Window in Relation to the Calculated Current Ripple

The current ripple was plotted in relation with the ADC window and is shown in Figure 7.13. The ADC window limits (i.e. 1.15V and 3.044V) were calculated via Equations 6.1 and 6.2 to see the 'current level' the ADC limit windows actually represent.

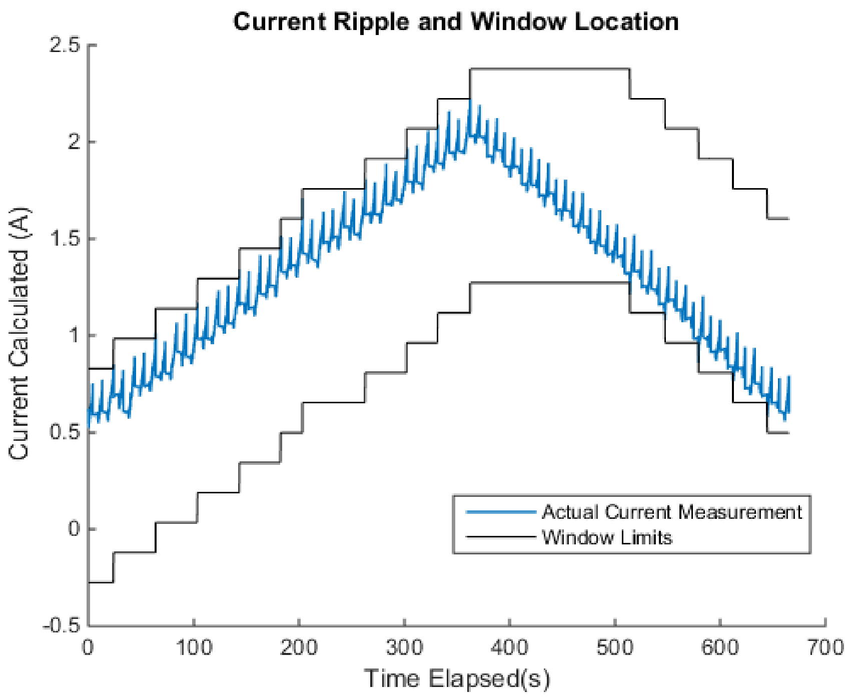


Figure 7.13. Current Ripple in Relation to the ADC Window

Figure 7.13 shows that the peaks in the current ripple are responsible for the shifting up of the ADC window and the troughs of the current ripple are responsible for the shifting down of the ADC reading. This was deduced by observing that each shift in the ADC window is aligned with a peak or a trough. It is also clear that the ripple stays at the edge of the ADC window throughout the measurement which is a product of the bias code algorithm where the bias is only changed incrementally by 1.2µA each time the limit of the window is hit.

7.8 ADC Ripple Analysis

The scale down due to the DC voltage offset from the Cypress op-amps also scales the ripple amplitude of the ADC readings down to the region of 450mV instead of the calculated 1.65V. This is easy to see from Equation 7.1:

$$I_{ripple} * R_{sense} * scale * K_{gain} = 0.250 * 0.001 * 0.244 * 6600 = 0.4026V \qquad (7.1)$$

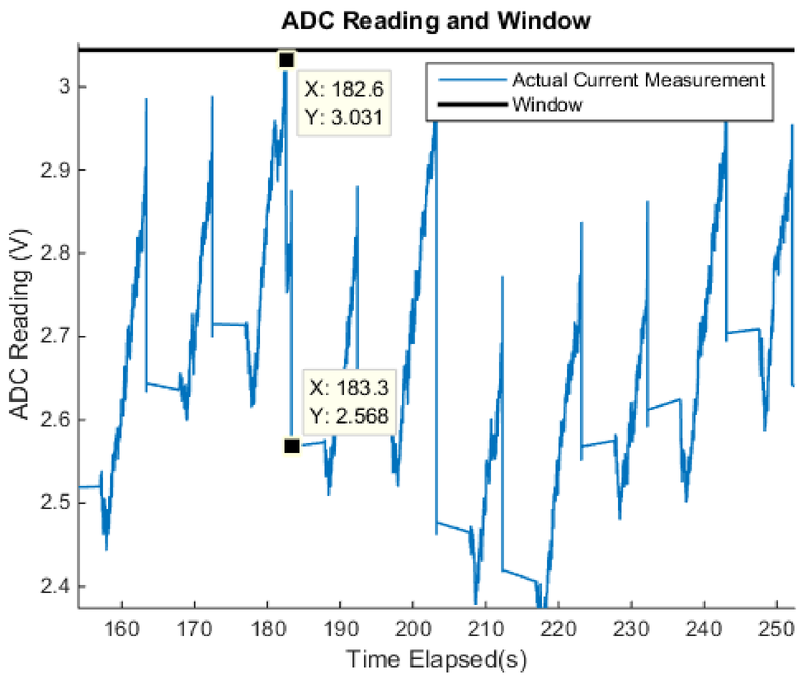


Figure 7.14. Current Ripple in Relation to the ADC Window

Figure 7.14 shows an enlarged version of the voltage ripple in relation to the ADC window where the ADC reading's ripple amplitude is 0.463V which is expected from Equation 7.1.

This does not fully utilize the capability of this current sense topology and hence we would need to make adjustments to the resistor values in order to counter this. This ripple can be scaled back up however and the next chapter will delineate how this process will be achieved.

8. Optimization

As seen from the previous chapter, the ripple did not fully utilise the capabilities of the circuit configuration. Hence we did a rethink on how to scale manipulate the gains such that the output ripple utilizes as many bits as possible.

The first step was to increase *s* which is the portion of the ADC range to scale the ripple to from 0.5 to 1. That is, for the case of OPA4330, the ripple will fill the entire range of the ADC. In this way, all 11 bits of the ADC range will be used fully, which is, the peak of the ripple will ideally be at 2047_{10} and the trough would be at 0_{10}. However, because of the scale down at the initial stage, it will be expected that the resultant ripple will be slightly smaller than the entire ADC range.

Since the scale down occurred at the initial stage, we would only be able to change K_2 for the scale up process. However, in K_2 we could only change *R3* since *R4* and *R5* affect K_{bias}. We first calculate the resistor values as per normal, but with *s*=1. We then set an equation to find the minimum value of *R3* that we can have. This is achieved by corresponding the maximum current level that the circuit has to measure with the maximum bias the DAC can output and equating that to the maximum ADC reading. This is shown in Equation 8.1:

$$\left(I_{\max+\frac{I_{ripple}}{2}}\right)R_{sense}K_1K_2K_{scale} + V_{off}K_{scale}K_2 = r_{ADC} + R4I_{bias,max} \qquad (8.1)$$

Rearranging 8.1 to find the minimum R3 for the optimized circuit we have Equation 8.2:

$$R3_{min} = R4\left(\frac{r_{ADC}+R4I_{bias,max}}{\left(I_{\max+\frac{I_{ripple}}{2}}\right)R_{sense}K_1K_{scale}+V_{off}K_{scale}} - 1 - \frac{R4}{R5}\right)^{-1} = 914.9\Omega \qquad (8.2)$$

The minimum resistance value for *R3* was found to be 914.9Ω at a 10A input current. Table 8.1 below shows the new resistor values of the circuit based on the resistances measured that are implemented in the circuit.

Table 8.1. Resistor Values for Optimized Circuit

Resistor	Value (Ω)
R1	47
R2	3160
R3	932
R4	421646
R5	4990
R7	22000
R8	68000

The new gain values are shown in Table 8.2 below.

Table 8.2. Gain Values for Optimized Circuit

Gain	Value (V/V)
K_1	68.23
K_2	537.91
K_{scale}	0.244
K_{gain}	8955

8.1 Simulation of Optimized Circuit

To test the equations, simulations were performed on LTSPICE. The voltage input to the second op-amp was determined by Equation 8.3. This is to simulate the expected output from the initial gain stage after the scale down.

$$Input\ to\ Second\ Opamp = (I_{sense}R_{sense}K_1 + V_{off})K_{scale} \qquad (8.3)$$

In simulation, a sinuosoidal input with DC offset of 10A and an amplitude of 0.125A was put into Equation 8.3 to figure out the parameters of the voltage source, *V1*, in the simulation. These parameters are 0.2391V DC offset with a 0.0021V amplitude. The circuit diagram of the simulation and its result is shown in Figures 8.1 and 8.2.

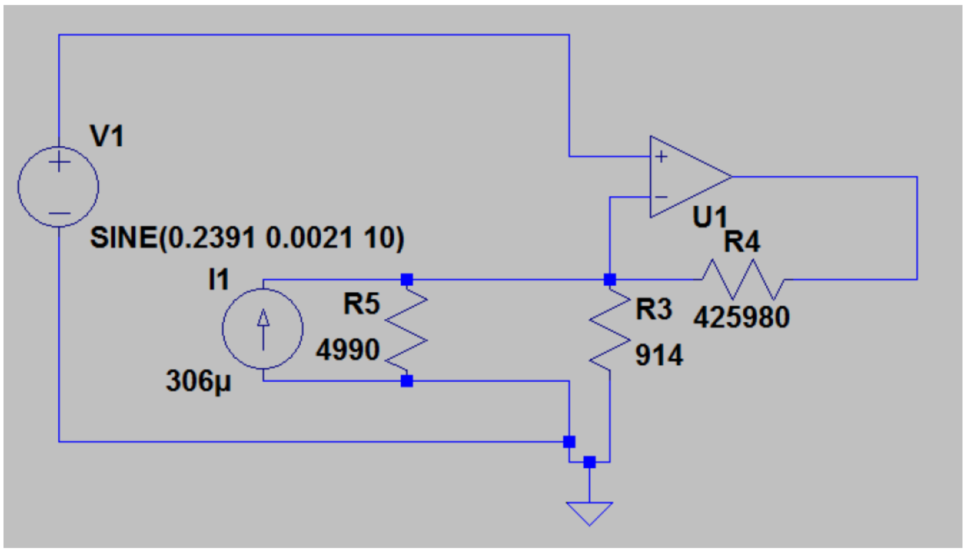
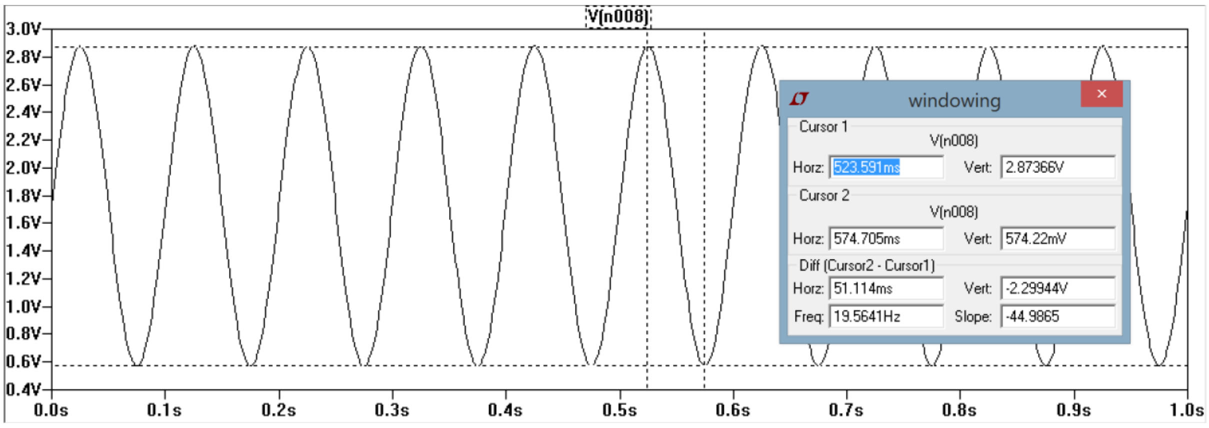


Figure 8.1.Circuit Diagram for Optimized Circuit Simulation



```
Cursor 1
                    V(n008)
Horz: 523.591ms      Vert: 2.87366V
Cursor 2
                    V(n008)
Horz: 574.705ms      Vert: 574.22mV
Diff (Cursor2 - Cursor1)
Horz: 51.114ms       Vert: -2.29944V
Freq: 19.5641Hz      Slope: -44.9865
```

Figure 8.2. Simulation Results for Optimized Circuit at Max Current

The simulation waveform shown in Figure 8.2 shows that at maximum current, the voltage ripple falls within the range of the ADC at the maximum bias output current. Also, the resultant voltage ripple's peak to peak voltage is 2.874V which proves promising that we would have high enough resolution in the ripple.

8.2 Ripple Sweep

The ripple sweep similar to the one done in Section 7.4 was performed to see if the simulation results carried over to the hardware implementation. Figure 8.3 below compares the ripple output from the Kiethley 2400 and the calculated current with a sweep from 0-3A. The ADC window limits were set to between 0.4 and 3V to create a very tight bound on the ripple.

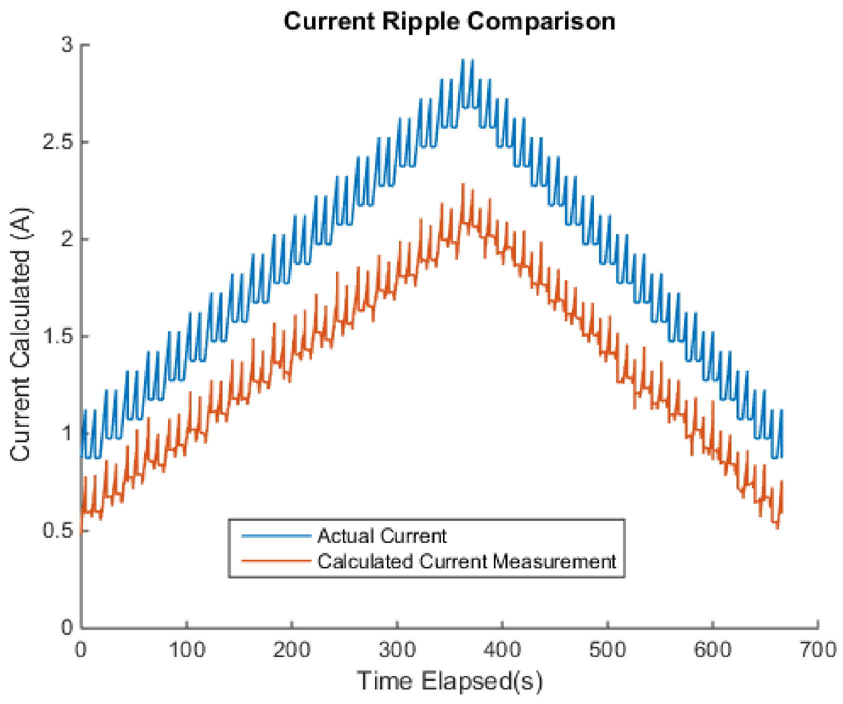


Figure 8.3. Current Ripple Sweep

The general pattern of the ripples shown resembles that of Figure 7.10 with a similar deviation away as current increases. The current ripple was digitally calibrated, using the same Matlab code as before to prove consistency in readings and shown in Figure 8.4 below.

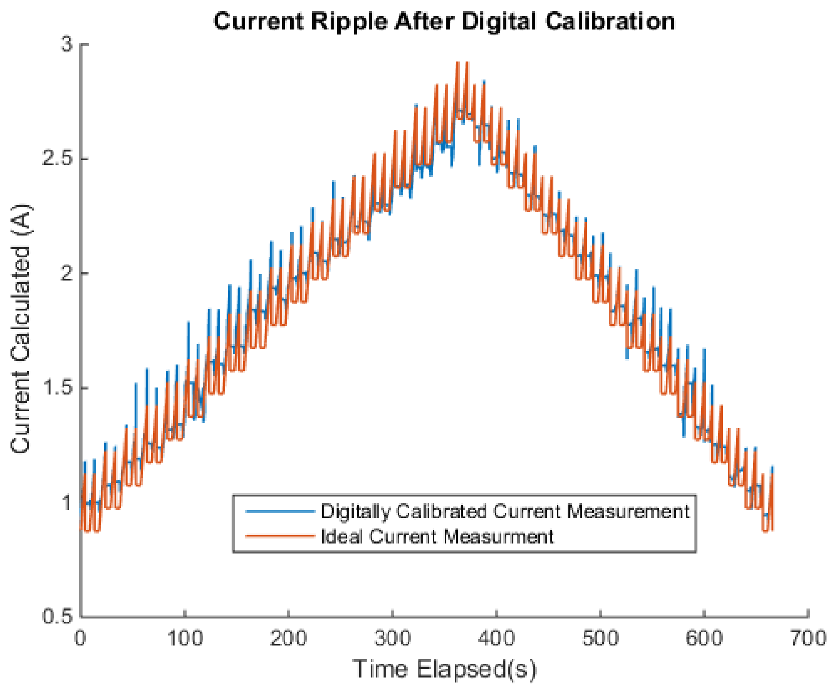


Figure 8.4. Current Ripple Sweep After Digital Calibration

8.3 ADC Window in Relation to the Calculated Current Ripple

The same plot as in Chapter 7.6 is performed to see how the window is shifted according to the changing current levels. From Figure 8.4 below, it becomes more apparent that the window shifts more as the current ripple peaks just meets the window limits of the ADC.

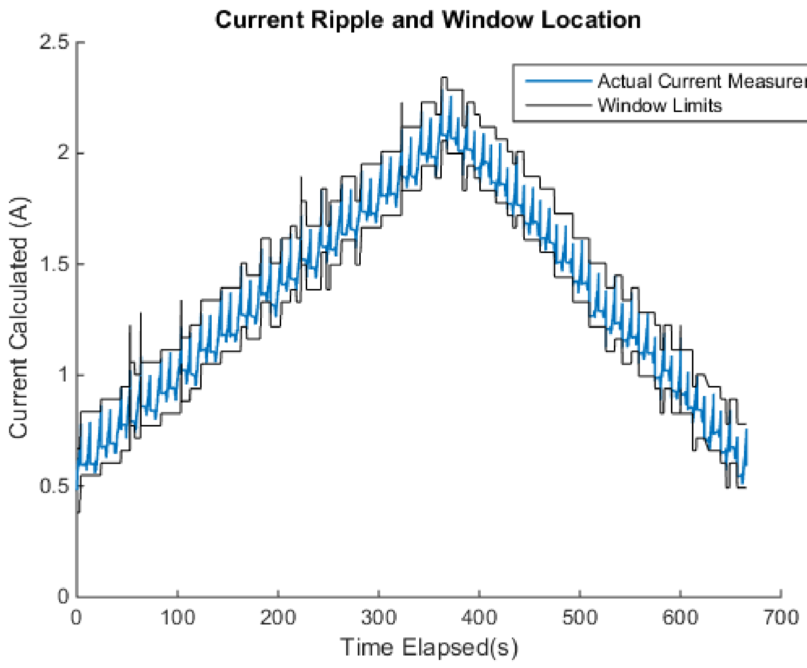


Figure 8.5. Current Ripple in Relation to Window Limits

The spikes in the ADC window limits corresponds heavily to the spikes in the current ripple which can be attributed to the transitions when the bias levels are changing. It can also be observed from the figure that the ripple fits very snugly within the window limits.

8.4 ADC Optimized Analysis

The ripple was enlarged in Matlab to view the actual peak to peak values measured by the ADC and is shown in Figure 8.6.

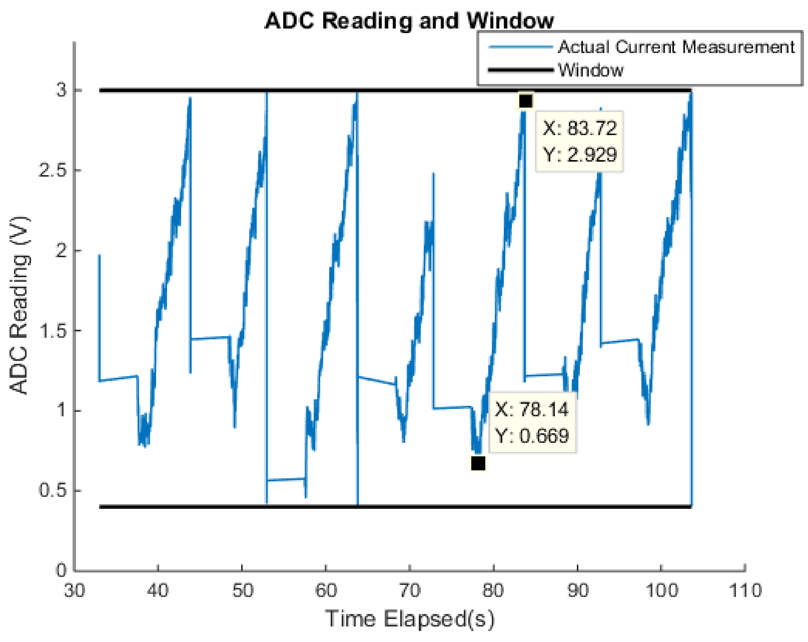


Figure 8.6. Enlarged Current Ripple

From figure 8.6, we can see that the ripple peak to peak is around 2.26V which is far better than the 0.4V measured originally.

8.5 Full Range Current Sweep

By increasing the gain of the second stage, there is a small risk that the maximum current may fall outside the range of the current sense circuit. To ensure that the circuit can measure the entire range of current, a 0.5-10A DC sweep was performed using the HP 6674A DC Power Supply. Figure 8.7 shows a comparison between the current measured by a Fluke 45 Digital Multi-Meter (red line) and the current measured from the current sense circuit (blue line). As seen before, the current measured from the current sense circuit deviates away from the measured current as current increases, with that difference changing linearly. Figure 8.8 shows the current measurements after digital calibration which indicates that the digitally calibrated measurements matches the current measurements very closely for the entire 0.5-10A range.

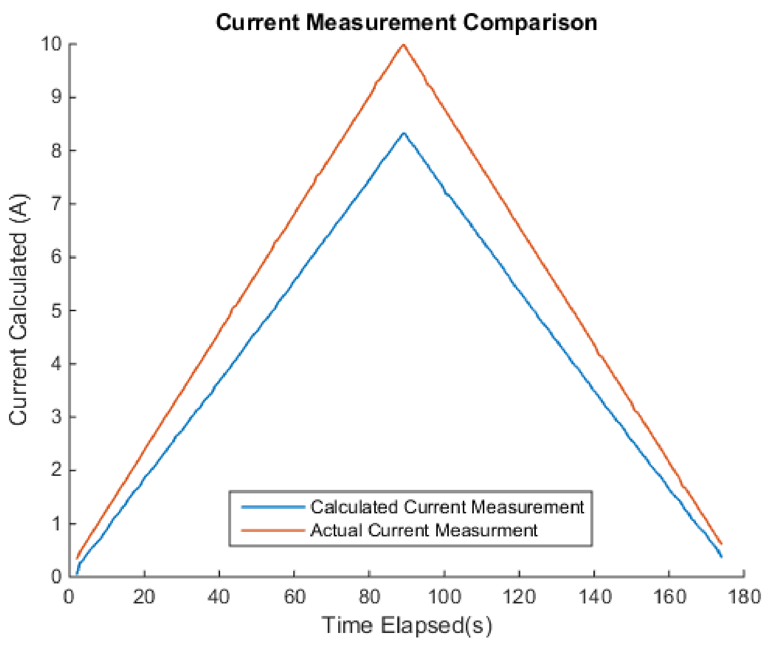


Figure 8.7. 0-10A Current Sweep Current Plot Comparison

Using the same algorithm as in Section 7.4 to prove consistency in readings, the results of the current measurement can be digitally calibrated and is shown in Figure 8.8.

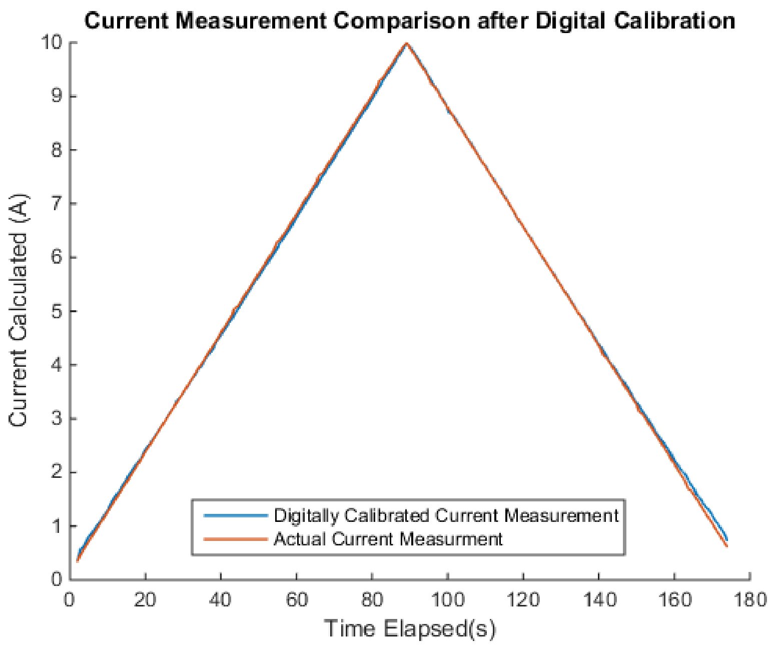


Figure 8.8. 0-10A Current Sweep Current Plot Comparison after Digital Calibration

9. Analysis

9.1 Resolution Analysis

We have seen that four different instances where the resolution has changed: a current sense without windowing, windowing circuit with OPA4330 with 1.65V window, windowing circuit with Cypress op-amps and the optimized windowing circuit with Cypress op-amps. Since an 11bit ADC range was used since an external Vref was used the total number of bits is 2047. The resolution is found by Equation 9.1:

$$Current\ Resolution = \frac{I_{ripple}}{\frac{V_{adc,ripple}}{r_{ADC}} * 2047} \left[\frac{mA}{bit\ change}\right] \tag{9.1}$$

$V_{adc,ripple}$ is the voltage ripple measured by the ADC. It should be noted that K_{gain} of the current sense amplifier without windowing is 325.9 since the maximum current has to coincide with the maximum range of the ADC, this makes $V_{adc,ripple}$ 0.0814V. Table 9.1 shows the different current resolution for the different current sense amplifiers.

Table 9.1. Current Resolution Comparison

Current Sense Amplifier	Current Resolution (mA/bit change)
Without Windowing	4.96
Windowing with OPA4330	0.244
Windowing with Cypress Op-amps	1.001
Windowing with Cypress Op-amps after optimization	0.178

As expected, the current resolution for the circuit without windowing is only just under 5mA per bit. It is also apparent that the circuit with OPA4330 has the most potential as a 0.244mA per bit change resolution is achieved even though only 10 bits were used effectively. The optimized circuit offers better resolution than the circuit with OPA4330 as a larger window is used. The optimization process improves the resolution of the current sense amplifier by 5.6 times and the final circuit offers an improvement of 28 times compared to current sense amplifier without windowing thereby proving the benefits of the windowing technique.

9.2 Error Analysis

To analyse the effectiveness of digital calibration, the error was plotted as a before and after comparison. Figure 9.1 shows the error plot for current measurements from 0.5A to 10A.

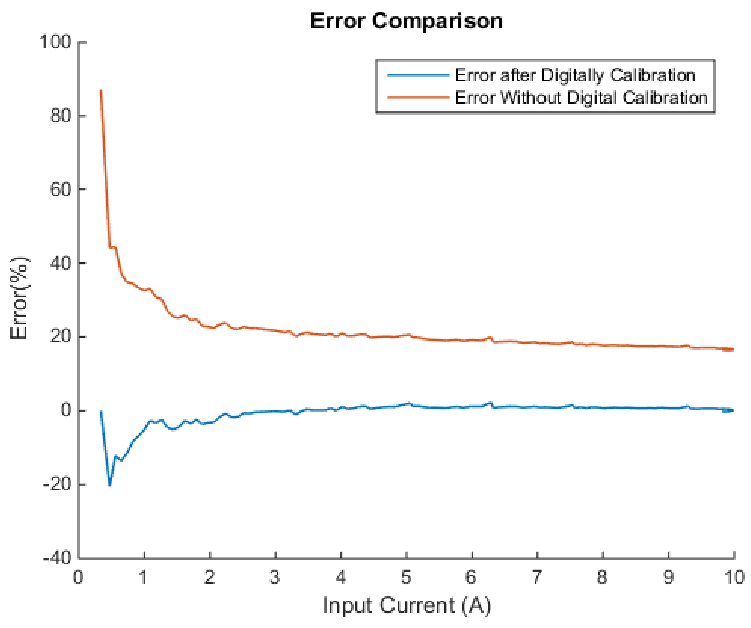


Figure 9.1. Error Plot Comparison

From Figure 9.1 above, we can see that without digital calibration, the error converges to slightly under 20% as current exceeds 2A. With digital calibration, the current stays close to 1% as current increases past 1A. We can thus conclude that the effective range of the current sense amplifier is from 1 to 10A. Table 9.2 shows the errors at every 10 data points in the measurement sweep.

Table 9.2. Error Measurements

Current Measured (A)	Current Calculated (A)	Error (%)	Current Calculated after Digital Calibration (A)	Error (%)
1.1755	0.8123	30.9	1.2145	-3.31
2.0592	1.5972	22.44	2.1235	-3.12
2.9565	2.3107	21.84	2.9612	-0.16
3.8388	3.0377	20.87	3.8124	0.69
4.7355	3.7846	20.08	4.6834	1.1
5.6173	4.5523	18.96	5.5753	0.75
6.4976	5.2795	18.75	6.4266	1.09
7.3924	6.0465	18.21	7.3178	1.01
8.27	6.8097	17.66	8.2052	0.78
9.1596	7.5778	17.27	9.0975	0.68
9.9938	8.3375	16.57	9.9814	0.12

9.3 Cost Analysis

Part of the reason why the Cypress op-amps were used was to reduce the cost of the current sense amplifier since high precision op-amps are expensive. Table 9.3 shows that this current sense solution proffers a nearly $4 reduction in price as compared to a high-precision op-amp with a TI low-power MCU.

Table 9.3. Cost Comparison

	Component	Cost ($)
Current Sense with OPA2330 and MSP430 Chip	MSP430FR5870	2.15
	OPA2330	3.05
	Total	5.20
Current Sense with Cypress PSOC4	CY8C4125AXI-473	1.34
	Total	1.34

10. Conclusion

This thesis has shown how high resolution lossless current sensing can be achieved through the windowing technique. We have proved that by optimising the circuit, we can achieve a high enough resolution on the current ripple for the entire current range specified. The windowing technique provides nearly 28 times the resolution than without it. The op-amps are not meant for such high measurements, hence we had to create workarounds and rely heavily on digital calibration to achieve accurate current sensing. Nevertheless we have shown how digital calibration helps achieve very accurate current measurements for the entire current range with errors very close to 1%. With that error, and at a much lower cost, the Cypress PSOC4 with its highly programmable in-built op-amps proves to be a cost-effective solution for the windowed current sense amplifier circuit.

11. References

[1] C. B. Barth and R. C. N. Pilawa-Podgurski, "Dithering Digital Ripple Correlation Control with Digtally-Assisted Windowed Sensing for Sloar Photovoltaic MPPT," in *Applied Power Electronics Conference (APEC),* Fort Worth, 2014.

[2] C. B. Barth and R. C. N. Pilawa-Podgurski, "Dithering Digital Ripple Correlation Control for Photovoltaic Maximum Power Point Tracking," *IEEE Transactions on Power Electronics,* vol. 30, no. 8, pp. 4548-4559, 2015.

[3] C. B. Barth and R. C. N. Pilawa-Podgurski, "Implementation of Dithering Digital Ripple Correlation Control for PV Maximum Power Point Tracking," in *IEEE Workshop on Control and Modeling for Power Electronics (COMPEL),* Salt Lake City, 2013.

[4] C. Schaef and J. T. Stauth, "Multilevel Power Point Tracking for Partial Power Processing Photovoltaic Converters," *IEEE Journal of Emerging and Selected Topics in Power Electronics,* vol. 2, no. 4, pp. 859-869, 2014.

[5] A. M. Latham, R. C. N. Pilawa-Podgurski, K. M. Odame and C. R. Sullivan, "Analysis and Optimization of Maximum Power Point Tracking Algorithms in the Presence of Noise," *IEEE Transactions on Power Electronics,* vol. 28, no. 7, pp. 3479-3493, 2013.

[6] M. Schuck and R. C. N. Pilawa-Podgurski, "Ripple Minimization Through Harmonic Elimination in Asymmetric Interleaved Multiphase dc-dc Converters," *IEEE Transactions,* vol. PP, no. 99, 2015.

[7] S. S. Pradeep, K. A. Kim, B. B. Johnson and P. T. Krein, "Differential Power Processing for Increased Energy Production and Reliability of Photovoltaic Systems," *IEEE Transactions,* vol. 28, no. 6, pp. 2968-2979, 2013.

[8] R. Bell and R. C. N. Pilawa-Podgurski, "Asynchronous and Distributed Maximum Power Point Tracking of Series-Connected Photovoltaic Sub-Modules Using Differential Power Processing," in *Control and Modeling for Power Electronics (COMPEL),* Santander, 2014.

[9] S. Qin, A. J. Morrison and R. C. N. Pilawa-Podgurski, "Enhancing Micro-inverter Energy Capture with Sub-module Differential Power Processing," in *Applied Power Electronics Conference and Exposition (APEC),* Fort Worth, 2014.

[10] L. Ng, S. Prawira, L. S. Ng and Y. Y. H. Lam, "Analysis of Lossless Current Sensing Techniques with High Accuracy and Linearity," in *International Conference on Communications, Circuits and Systems (ICCCAS),* Kokura, 2007.

[11] E. Dallago and M. Passoni, "Lossless Current Sensing in Low-Voltage High-Current DC/DC Modular Supplies," *IEEE Transactions on Industrial Electronics,* vol. 47, no. 6, pp. 1249-1252, 2000.

[12] R. C. N. Pilawa-Podgurski, W. Li, I. Celanovic and D. J. Perreault, "Integrated CMOS DC-DC Converter with Digital Maximum Power Point Tracking for a Portable

Thermophotovoltaic Power Generator," in *Energy Conversion Congress and Exposition (ECCE)*, Phoenix, 2011.

[13] S. Qin, S. T. Cady, A. D. Dominguez-Garcia and R. C. N. Pilawa-Podgurski, "A Distributed Approach to Maximum Power Point Tracking for Photovoltaic Sub-Module Differential Power Processing," *IEEE Transactions on Power Electronics,* vol. 30, no. 4, pp. 2024-2040, 2015.

[14] G. Barbose, N. Darghouth, S. Weaver and R. Wiser, "Tracking the Sun VI - An Historical Summary of the Installed Price of Photovoltaics in the United States from 1998 to 2012," July 2013. [Online]. Available: http://emp.lbl.gov/sites/all/files/lbnl-6350e.pdf. [Accessed 09 May 2015].

[15] R. C. N. Pilawa-Podgurski and D. J. Perreault, "Sub-Module Integrated Distributed Maximum Power Point Tracking for Solar Photovoltaic Applications," *IEEE Transactions on Power Electronics* , vol. 28, no. 6, 2013.

Appendix A: PSOC4 C Code

A1. Main File

```
/* ========================================
 *
 * Copyright Benedict Foo, University of Illinois Urbana-Champaign
 * All Rights Reserved
 * UNPUBLISHED, LICENSED SOFTWARE.
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION
 * WHICH IS THE PROPERTY OF University of Illinois Urbana-Champaign ECE
Department
 *  Date: 05/09/2015
 * ========================================
*/
#include <project.h>

/* Macro definitions */
#define LOW                 (0u)
#define HIGH                (1u)
#define CHANNEL_1           (0u)
#define CLEAR_SCREEN        (0x0C)
#define CONVERT_TO_ASCII    (0x30u)

/* Global variables */
volatile uint32 windowFlag    = 0u;
volatile uint8  dataReady     = 0u;
volatile uint16 adcVal = 0u;
volatile char shit;

/* Interrupt prototypes */
CY_ISR_PROTO(ADC_ISR_Handler);


/* Send the channel number and voltage to UART */
static void SendChannelVoltage(int16 mVolts, int32 bias);

int main()
{
    /* Place your initialization/startup code here (e.g. MyInst_Start()) */
   // clock_t start;
    //time_t t1;
    //Variable initializaions

    int16 mVolts;
    int32 bias = 91; //initial bias assuming 1A DC offset
    int16 previousValue = 0;
    char ready;
    char start;
    int readyCompare;
    int startCompare = 0;

    /* Start the Components */
    Opamp_1_Start();
    Opamp_2_Start();
    UART_1_Start();
    ADC_SAR_Seq_1_Start();
    IDAC_1_Start();
    IDAC_1_SetValue(bias);

   /* Start ISRs */
    CyGlobalIntEnable; /* Interrupt Enable*/
   /* ADC_SAR_Seq_1_IRQ_StartEx(ADC_ISR_Handler);*/
    ADC_SAR_Seq_1_IRQ_Enable();
   UART_1_SpiUartClearTxBuffer(); //ensures no junk gets in there
   while  (startCompare != 1)       //infinite loop till python command
says Go!
    {
       start = UART_1_UartGetChar();
       startCompare = start - '0';
        if (startCompare == 1)
            break;
    }
    //Uncomment to measure time
    //long i=0;
    //double diff;
    //start = clock();
    for(;;)
    {

        ADC_SAR_Seq_1_StartConvert();
        while (dataReady == 0) /* Wait for ADC conversion, continues
looping if there is no data */
        {
            ;
        }

        mVolts = ADC_SAR_Seq_1_CountsTo_mVolts(0, adcVal); //converts adc
reading to mV

        ready = UART_1_UartGetChar();  //Gets ascii character from Python
        readyCompare = ready - '0';    //Converts ascii to int

        if (readyCompare == 1) //Ensures that UART happens only when Python
code is ready to receive
        {
            SendChannelVoltage(mVolts,bias);

        }
        /* Check for ADC window limit interrupt */
        if(windowFlag != 0u)
        {
            /* Turn ON the LED when input is outside the voltage window
(1.15V - 3.044V) */
            LED_Write(LOW);
            /* Increases/decreases bias depending on which limit it hits*/
            if (mVolts<2000)
            {
                bias = bias-0x1;
                IDAC_1_SetValue(bias);
            }

            else
            {
                bias = bias+0x1;
                IDAC_1_SetValue(bias);
            }
            /* Note: If LED is active HIGH, then replace "LOW" with "HIGH"
*/
        }
        else
        {
            /* Turn OFF the LED when input is within the voltage window
(4.5mV - 2.043V) */
            LED_Write(HIGH);

            /* Note:If LED is active HIGH, then replace "HIGH" with "LOW"
*/
        }
                /* If ADC result or channel has been changed, send the data
to UART */

        dataReady = 0u;

    }
//     diff = ((double)clock()-start)/CLOCKS_PER_SEC;

}
static void SendChannelVoltage(int16 mVolts, int32 bias)
{



    /* Find the sign of the result */
    /*if(mVolts < 0)      //uncomment if measuring current the other way ->
ADC needs to be reprogrammed to differential
    {
        UART_1_UartPutString("-");
        mVolts = -mVolts;
    }*/

    /* Send voltage and bias to UART */
    UART_1_UartPutChar((mVolts/1000u) + CONVERT_TO_ASCII);
    mVolts %= 1000u;
    UART_1_UartPutChar((mVolts/100u) + CONVERT_TO_ASCII);
    mVolts %= 100u;
    UART_1_UartPutChar((mVolts/10u) + CONVERT_TO_ASCII);
    mVolts %= 10u;
    UART_1_UartPutChar(mVolts + CONVERT_TO_ASCII);
    UART_1_UartPutChar(32); //space for Excel delimiter
    UART_1_UartPutChar((bias/100u) + CONVERT_TO_ASCII);
    bias %= 100u;
    UART_1_UartPutChar((bias/10u) + CONVERT_TO_ASCII);
    bias %= 10u;
    UART_1_UartPutChar(bias + CONVERT_TO_ASCII);
    UART_1_UartPutCRLF(32); //sends carriage return and new line

    }

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

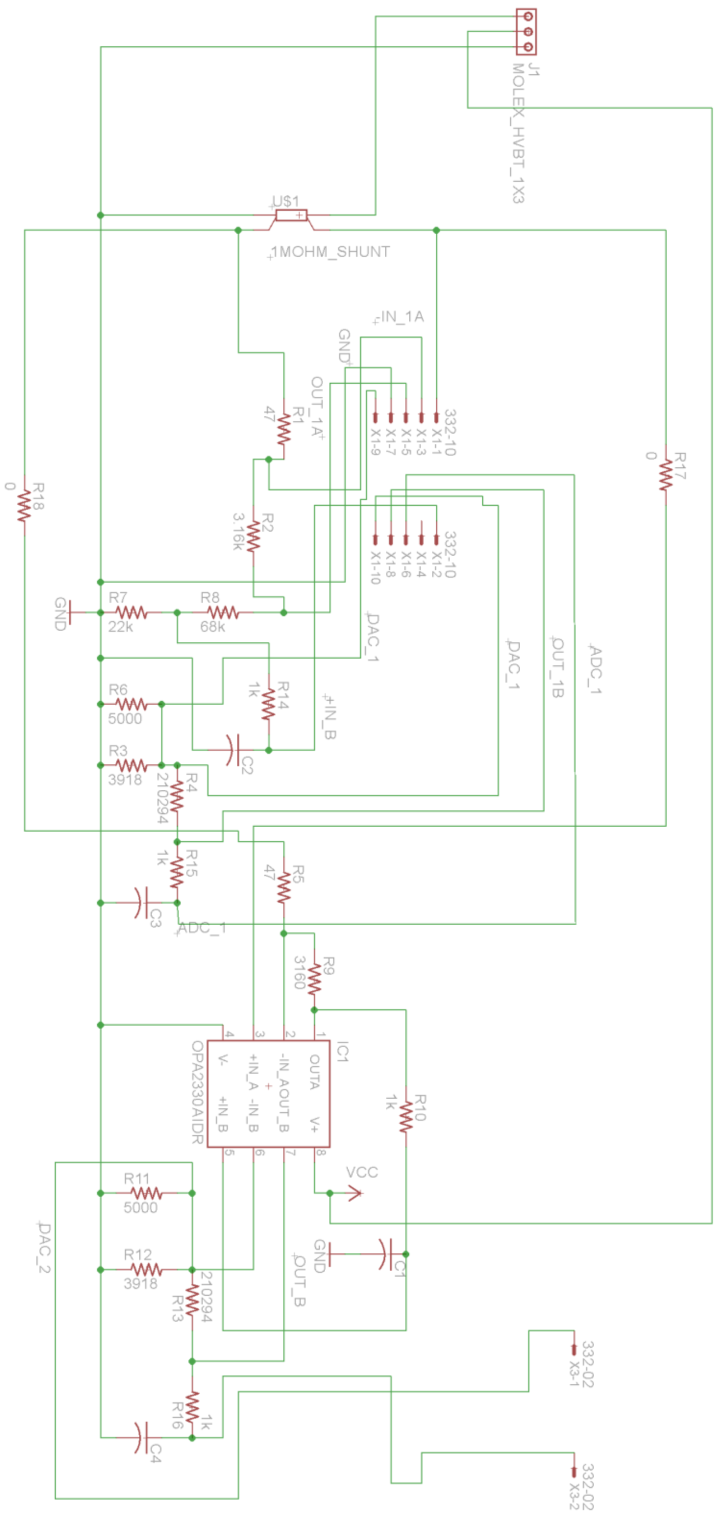
A.2 ADC Interrupt Handler

```
CY_ISR( ADC_SAR_Seq_1_ISR )
    {
        uint32 intr_status;

        /* Read interrupt status register */
        intr_status = ADC_SAR_Seq_1_SAR_INTR_REG;


/************************************************************************
        *  Custom Code
        *  - add user ISR code between the following #START and #END tags

*************************************************************************/
        /* `#START MAIN_ADC_ISR`  */


    /* Check for End of Scan interrupt */
    if((intr_status & ADC_SAR_Seq_1_EOS_MASK) != 0u)
    {
        adcVal=ADC_SAR_Seq_1_GetResult16(0);
        /* Read range interrupt status and raise the flag */
        windowFlag = ADC_SAR_Seq_1_SAR_RANGE_INTR_MASKED_REG; //will not be
a 0 if it is outside the range
        //saturationFlag = ADC_SAR_Seq_1_SAR_SATURATE_INTR_MASKED_REG;

        /* Clear range detect status */
        ADC_SAR_Seq_1_SAR_RANGE_INTR_REG = windowFlag;
        dataReady = 1u;

        /* `#END`  */

        /* Clear handled interrupt */
        ADC_SAR_Seq_1_SAR_INTR_REG = intr_status;
    }

#endif   /* End ADC_SAR_Seq_1_IRQ_REMOVE */

    }
```

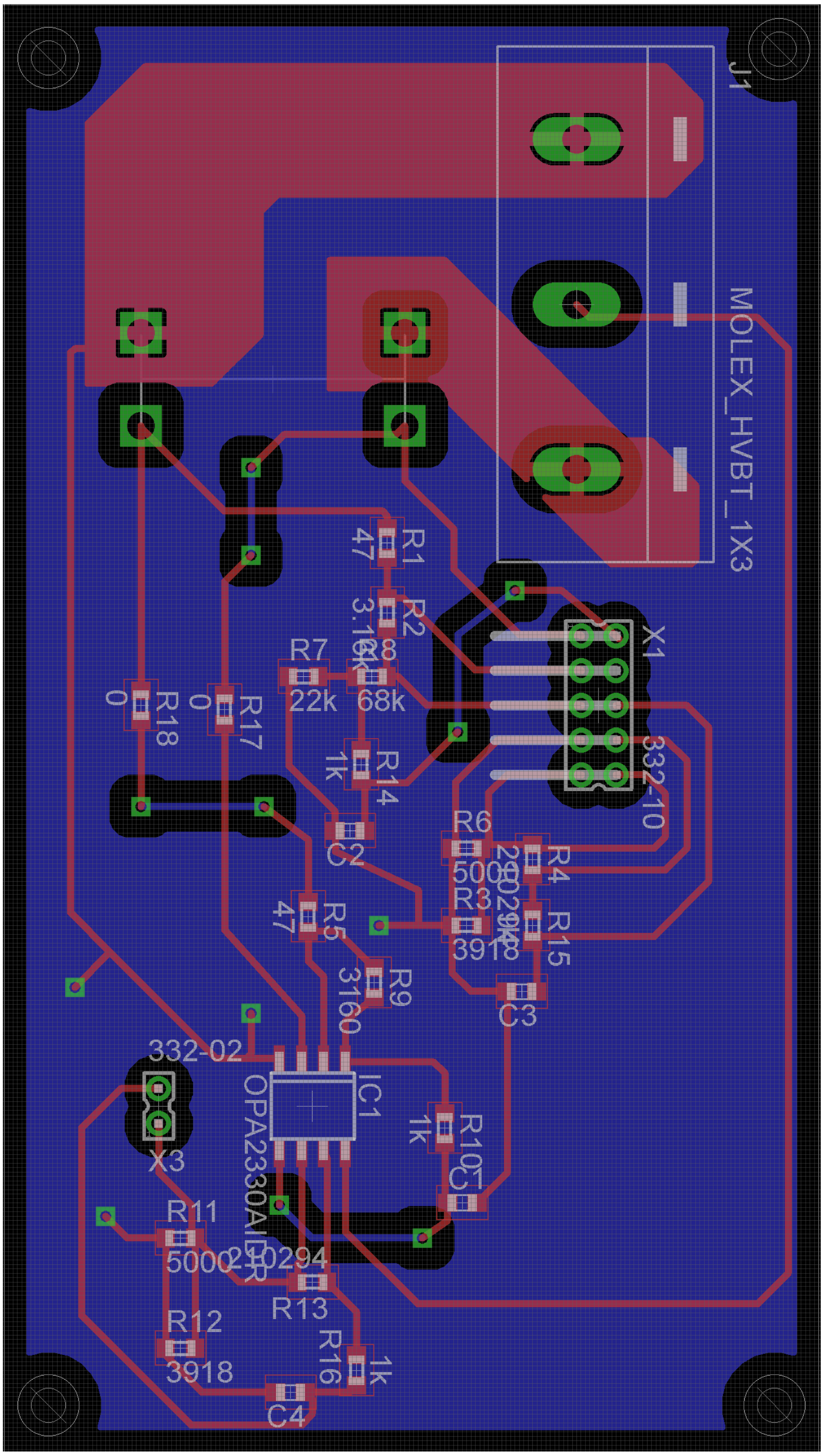
Appendix B: PCB Schematic and Layout

B1. Eagle Schematic

This schematic includes a circuit for OPA2330 for comparison.



B2. Board Layout



Appendix C: Python Codes

C1. DC Voltage Sweep

This code communicates with the Cypress PSOC4 for a 1 to 10mV sweep with ten points taken at each voltage input.

```
__author__ = 'Benedict Foo'

#This code implements PC-based control of the Cypress and
Kiethley 2400
#Also an example of file creation and write to file in Python.

#===============================================================
=============
#  IMPORTS
#===============================================================
=============


from microcontroller_serial import *
import serial
import glob
import time
import os
import os.path
import inspect
from datetime import datetime
import sys
from pilawa_instruments import *


if __name__=="__main__":

    #print sys.path
    # Writing into a file
    #skipLineFlag = True

    debug = False
    # t is a timestamp made by python. it records the year,
month, and day in numerical form, in that order.
    t = time.strftime('%Y%m%d')

    #names a folder in the given path. the name of the folder
is t, the timestamp.
    foldername =
"C:/Users/Benedict/PycharmProjects/Cypress_UART/InputVoltageSw
eep/%s" %t
```

```
    print(foldername)


    #checks if the folder 'foldername' exists
    if not os.path.exists(foldername):
        print "I Should Be Here"
        #this command actually creates the folder named above
given that it does not already exist
        os.makedirs(foldername)
        print "Folder Created"
        # i is a file counter. if directory has been created
then counter = 0
        i = 0

    else:
        # glob.glob returns a list with the file name of each
file in the specified directory with the specified file name
similarity. the similarity must contain a * character
substituting the part that changes, in this case substituting
the counter.
        txtList =
glob.glob("C:/Users/Benedict/PycharmProjects/Cypress_UART/Inpu
tVoltageSweep/%s/test*.txt" %t)

        #if directory exists, then counter = number of files
named test*.txt
        i = len(txtList)
        i = i+1



        #specifies the path to the file and changes the
filename on each iteration (after completing the number of
batches per file). %s (string) is replaced by t as the
timestamp specifying the selected folder. %d (integer) is
replaced by i as the number of the file.
        filename =
"C:/Users/Benedict/PycharmProjects/Cypress_UART/InputVoltageSw
eep/%s/test%d.txt" %(t,i)

        #opens the file or creates a new one if not found. 'w'
specifies the file is to be written on, erasing anything
originally in.
        readf = open(filename,"w")
        stamp = datetime.now().strftime('%H:%M:%S')
        #writes at the top of the file what each column data
is
        readf.write("ADC Bias")
        #starts a new line for the data measurements
        readf.write("\n Time:")
        readf.write(stamp)
        readf.write("\n")
        if debug: print "File created and open"

    #create an instance of the microcontroller class.
    if debug: print "Initializing Micro Instance"
    cypress = microcontroller_serial( port = 6 , baud = 9600,
debug = False, timeout =6 )

    if(debug): print "Initialize GPIB"
    gpib  = prologix_serial( port = 4, baud=9600, debug=False,
timeout=5)

    if(debug): print "Initialize Sourcemeter"
    sourcemeter    = prologix_2400(prologix=gpib, addr=06,
debug=False)

    sourcemeter.setSource('VOLT', 0.001)  #Set output to 1mV
    time.sleep(0.1)
    sourcemeter.activate()
    print "Enter '1' to begin everything"
    userinput = raw_input('--> ')
    cypress.write(userinput)
    cnt = 0.000 #Cnt is the 'current' going through Rsense,
which in this case is Vsense
    inner_loop = 0
    maxim = 0.010
    while cnt <= 0.01: #sweep up

        sourcemeter.setSource ('VOLT',cnt )
        time.sleep(0.1)     #sets a delay to allow bias to
converge to correct value
        while inner_loop < 10: #inner_loop indicates how many
readings taken at each current level

            cypress.write("1")                # Ensures that
data is only read when Python Code is ready
                                              # Also clears the
buffer so new data can be sent
            #time.sleep (0.5)
            read = cypress.readline()         # Read data from
micro until newline is sent
            read = read.rstrip(' \r\n')
            #print adc                        # Print the value
received from the micro
            #readf.write(adc)                 # Write onto the
open file
            #bias = cypress.readline()
            #readf.write(" ")
            readf.write("%s %s\n" %(read, cnt))
            #print bias
            print read
            inner_loop = inner_loop + 1
        #time.sleep(0.3)
        cnt = cnt + 0.0001    #increments 'current' by 0.1A
        inner_loop = 0


    while cnt >= 0.000: #sweep down

        sourcemeter.setSource ('VOLT',cnt )
        time.sleep(0.1)      #sets a delay to allow bias to
converge to correct value (digital calibration)
        while inner_loop < 10: #inner_loop indicates how many
readings taken at each current level

            cypress.write("1")                # Ensures that
data is only read when Python Code is ready
                                              # Also clears the
buffer so new data can be sent
            #time.sleep (0.5)
            read = cypress.readline()         # Read data from
micro until newline is sent
            read = read.rstrip(' \r\n')
            #print adc                        # Print the value
received from the micro
            #readf.write(adc)                 # Write onto the
open file
            #bias = cypress.readline()
            #readf.write(" ")
            readf.write("%s %s\n" %(read, cnt))
            #print bias
            print read
            inner_loop = inner_loop + 1
        #time.sleep(0.3)
        cnt = cnt - 0.0001    #decrements 'current' by 0.1A
        inner_loop = 0

    #Close connection to microprocessor when user exits.
    #sourcemeter.deactivate()
    cypress.terminate()
    readf.close()   #closes file so that data can be written
from io buffer
```

C2. Current Ripple Sweep

This is the Python Code that generates a ripple current on the Kiethley 2400.

```
_author__ = 'Benedict Foo'

#This code implements PC-based control of the Cypress and
Kiethley 2400
#Also an example of file creation and write to file in Python.

#==============================================================
=============
# IMPORTS
#==============================================================
=============


from microcontroller_serial import *
import serial
import glob
import time
import os
import os.path
import inspect
from datetime import datetime
import sys
from pilawa_instruments import *


if __name__=="__main__":

    #print sys.path
    # Writing into a file
    #skipLineFlag = True

    debug = False
    # t is a timestamp made by python. it records the year,
month, and day in numerical form, in that order.
    t = time.strftime('%Y%m%d')

    #names a folder in the given path. the name of the folder
is t, the timestamp.
    foldername =
"C:/Users/Benedict/PycharmProjects/Cypress_UART/InputCurrentRi
ppleSweep/%s" %t

    print(foldername)
    #Suck a dick

    #checks if the folder 'foldername' exists
    if not os.path.exists(foldername):
        #this command actually creates the folder named above
given that it does not already exist
        os.makedirs(foldername)
        print "Folder Created"
        # i is a file counter. if directory has been created
then counter = 0
        i = 0

    else:
        # glob.glob returns a list with the file name of each
file in the specified directory with the specified file name
similarity. the similarity must contain a * character
substituting the part that changes, in this case substituting
the counter.
        txtList =
glob.glob("C:/Users/Benedict/PycharmProjects/Cypress_UART/Inpu
tCurrentRippleSweep/%s/test*.txt" %t)

        #if directory exists, then counter = number of files
named test*.txt
        i = len(txtList)
        i = i+1


        #specifies the path to the file and changes the
filename on each iteration (after completing the number of
batches per file). %s (string) is replaced by t as the
timestamp specifying the selected folder. %d (integer) is
replaced by i as the number of the file.
        filename =
"C:/Users/Benedict/PycharmProjects/Cypress_UART/InputCurrentRi
ppleSweep/%s/test%d.txt" %(t,i)

        #opens the file or creates a new one if not found. 'w'
specifies the file is to be written on, erasing anything
originally in.
        readf = open(filename,"w")
        stamp = datetime.now().strftime('%H:%M:%S')
        #writes at the top of the file what each column data
is
        readf.write("ADC Bias")
        #starts a new line for the data measurements
        readf.write("\n Time:")
        readf.write(stamp)
        readf.write("\n")
        if debug: print "File created and open"

    #create an instance of the microcontroller class.
    if debug: print "Initializing Micro Instance"
    cypress = microcontroller_serial( port = 6 , baud = 9600,
debug = False, timeout =6 )

    if(debug): print "Initialize GPIB"
    gpib  = prologix_serial( port = 4, baud=9600, debug=False,
timeout=5)

    if(debug): print "Initialize Sourcemeter"
    sourcemeter    = prologix_2400(prologix=gpib, addr=06,
debug=False)

    sourcemeter.setSource('CURR', 1)  #Set output to 1mV
    time.sleep(0.1)
    sourcemeter.activate()
    #print "Enter '1' to begin everything"
    #userinput = raw_input('--> ')
    #cypress.write("1")
    minim = 0.000 #Cnt is the 'current' going through Rsense,
which in this case is Vsense

    maxim = 0.03
    offset = 1 #offset starts at 1A
    peak = offset + 0.125 #0.250mA ripple
    trough = offset - 0.125
    cycles = 0.000
    level=trough

    print "Enter '1' to begin everything" #starts the ADC
readings
    userinput = raw_input('--> ')
    cypress.write(userinput)

    #measure loop time
    t0 = time.clock()

    while offset <= 2.8: #sweep up
        while cycles < 2: #2 cycles per offset
            while level<=peak:
                sourcemeter.setSource('CURR',level)
                time.sleep(0.00001)                 # waits 10us
                cypress.write("1")                  # Ensures
that data is only read when Python Code is ready
                                                    # Also clears
the buffer so new data can be sent
                read = cypress.readline()           # Read data
from micro until newline is sent
                read = read.rstrip(' \r\n')
                #print adc                          # Print the
value received from the micro
                #readf.write(adc)                   # Write onto
the open file
                #bias = cypress.readline()
                #readf.write(" ")
                tdiff =  time.clock()-t0
                readf.write("%s %s %s\n" %(read, level,tdiff))
                #print bias
                print read
                #print level
                level = level + 0.001
            while level>=trough:
                sourcemeter.setSource('CURR',level)
                time.sleep(0.00001)                 # waits 10us
                cypress.write("1")                  # Ensures
that data is only read when Python Code is ready
                read = cypress.readline()           # Read data
from micro until newline is sent
                read = read.rstrip(' \r\n')
                #print adc                          # Print the
value received from the micro
                #readf.write(adc)                   # Write onto
the open file
                #bias = cypress.readline()
                #readf.write(" ")
                readf.write("%s %s %s\n" %(read, level,tdiff))
                #print bias
                print read
                #print level
                level = level - 0.001
            cycles = cycles + 1
        offset = offset + 0.1
        peak = offset + 0.125 #0.250mA ripple
        trough = offset - 0.125
        cycles = 0


    while offset >= 1: #sweep down
        while cycles < 2: #2 cycles per offset
            while level<=peak:
                sourcemeter.setSource('CURR',level)
                time.sleep(0.00001)                 # waits 10us
                cypress.write("1")                  # Ensures
that data is only read when Python Code is ready
                                                    # Also clears
the buffer so new data can be sent
                read = cypress.readline()           # Read data
from micro until newline is sent
                read = read.rstrip(' \r\n')
                #print adc                          # Print the
value received from the micro
                #readf.write(adc)                   # Write onto
the open file
                #bias = cypress.readline()
                #readf.write(" ")
                tdiff =  time.clock()-t0
                readf.write("%s %s %s\n" %(read, level,tdiff))
                #print bias
                print read
                #print level
                level = level + 0.001
            while level>=trough:
                sourcemeter.setSource('CURR',level)
                time.sleep(0.00001)                 # waits 10us
                cypress.write("1")                  # Ensures
that data is only read when Python Code is ready
                read = cypress.readline()           # Read data
from micro until newline is sent
                read = read.rstrip(' \r\n')
                #print adc                          # Print the
value received from the micro
                #readf.write(adc)                   # Write onto
the open file
                #bias = cypress.readline()
                #readf.write(" ")
                readf.write("%s %s %s\n" %(read, level,tdiff))
                #print bias
                print read
                #print level
                level = level - 0.001
            cycles = cycles + 1
        offset = offset - 0.1
        peak = offset + 0.125 #0.250mA ripple
        trough = offset - 0.125
        cycles = 0


    #Close connection to microprocessor when user exits.
    #sourcemeter.deactivate()

    cypress.terminate()
    readf.close()  #closes file so that data can be written
from io buffer
```

C3. 10A Python Sweep

This is the code for a 0.5 to 10A DC current sweep with the HP6674A DC Power Supply

```
__author__ = 'Benedict Foo'

#This code implements PC-based control of the Cypress and 
HP6674A and Fluke 45 DMM
#Also an example of file creation and write to file in Python.

#==============================================================
=============
#  IMPORTS
#==============================================================
=============


from microcontroller_serial import *
import serial
import glob
import time
import os
import os.path
import inspect
from datetime import datetime
import sys
from pilawa_instruments import *


if __name__=="__main__":

    #print sys.path
    # Writing into a file
    #skipLineFlag = True

    debug = False
    # t is a timestamp made by python. it records the year, 
month, and day in numerical form, in that order.
    t = time.strftime('%Y%m%d')

    #names a folder in the given path. the name of the folder 
is t, the timestamp.
    foldername = 
"C:/Users/Benedict/PycharmProjects/Cypress_UART/InputCurrentSw
eep/%s" %t

    print(foldername)
    #Suck a dick

    #checks if the folder 'foldername' exists
    if not os.path.exists(foldername):
        #this command actually creates the folder named above
given that it does not already exist
        os.makedirs(foldername)
        print "Folder Created"
        # i is a file counter. if directory has been created
then counter = 0
        i = 0

    else:
        # glob.glob returns a list with the file name of each
file in the specified directory with the specified file name
similarity. the similarity must contain a * character
substituting the part that changes, in this case substituting
the counter.
        txtList =
glob.glob("C:/Users/Benedict/PycharmProjects/Cypress_UART/Inpu
tCurrentSweep/%s/test*.txt" %t)

        #if directory exists, then counter = number of files
named test*.txt
        i = len(txtList)
        i = i+1



        #specifies the path to the file and changes the
filename on each iteration (after completing the number of
batches per file). %s (string) is replaced by t as the
timestamp specifying the selected folder. %d (integer) is
replaced by i as the number of the file.
        filename =
"C:/Users/Benedict/PycharmProjects/Cypress_UART/InputCurrentSw
eep/%s/test%d.txt" %(t,i)

        #opens the file or creates a new one if not found. 'w'
specifies the file is to be written on, erasing anything
originally in.
        readf = open(filename,"w")
        stamp = datetime.now().strftime('%H:%M:%S')
        #writes at the top of the file what each column data
is
        readf.write("ADC Bias")
        #starts a new line for the data measurements
        readf.write("\n Time:")
        readf.write(stamp)
        readf.write("\n")
        print "File created and open"
    #print sys.path
    # Writing into a file
    #skipLineFlag = True
    debug = False

    if(debug): print "Initialize GPIB"
    gpib  = prologix_serial( port = 7, baud=9600, debug=False,
timeout=5)

    if(debug): print "Initialize Sourcemeter"
    sourcemeter    = prologix_6674a(prologix=gpib, addr=9,
debug=False)

    #if(debug): print "Initialize GPIB"
    #gpib1  = prologix_serial( port = 4, baud=9600,
debug=False, timeout=5)

    if(debug): print "Initialize Fluke"
    fluke  = prologix_FLUKE45(prologix=gpib, addr=2,
debug=False)

    #create an instance of the microcontroller class.
    if debug: print "Initializing Micro Instance"
    cypress = microcontroller_serial( port = 6 , baud = 9600,
debug = False, timeout =6 )

    meter_addrs = [2]
     #set fluke to DC ammeter mode

    #time.sleep (0.5)


    sourcemeter.activate()
    sourcemeter.setCurrent(12)  #Set max current to 12A

    sourcemeter.setVoltage(0.2)   #set voltage to 1V

    print "Enter '1' to begin everything" #starts the ADC
readings
    userinput = raw_input('--> ')
    cypress.write(userinput)

    #measure loop time
    t0 = time.clock()
    fluke.setMode('ADC')
    #time.sleep(5)              #wait 5s to read the data
    #gpib1.trigger_devices(meter_addrs)
    #read = fluke.readData()
    #read = sourcemeter.readCurrent()
    volt = 0.5 #Cnt is the 'current' going through Rsense,
which in this case is Vsense
    inner_loop = 0
    time.sleep (0.5)
    #fluke.waitForTrigger()
    #gpib1.trigger_devices(meter_addrs)
    readv = fluke.getMeasurement()
    print readv
    readv = 0

    while volt <= 11.6: #sweep up

        sourcemeter.setVoltage (volt)
        #time.sleep(0.00001)     #sets a delay to allow bias
to converge to correct value
        while inner_loop < 1: #inner_loop indicates how many
readings taken at each current level

            cypress.write("1")                # Ensures that
data is only read when Python Code is ready
                                              # Also clears the
buffer so new data can be sent
            #time.sleep (0.5)
            read = cypress.readline()         # Read data from
micro until newline is sent
            read = read.rstrip(' \r\n')
            time.sleep(0.3)
            readv = fluke.getMeasurement()
            tdiff =  time.clock()-t0
            readf.write("%s %s %s %s\n" %(read, volt, tdiff,
readv))
            #print bias
            print readv
            inner_loop = inner_loop + 1
        #time.sleep(0.3)
        volt = volt + 0.1   #increments 'current' by 0.1A
        inner_loop = 0


    while volt>= 0.500: #sweep down

        sourcemeter.setVoltage (volt)
        #time.sleep(0.000010)     #sets a delay to allow bias
to converge to correct value (digital calibration)
        while inner_loop < 1: #inner_loop indicates how many
readings taken at each current level

            cypress.write("1")                # Ensures that
data is only read when Python Code is ready
                                              # Also clears the
buffer so new data can be sent
            #time.sleep (0.5)
            read = cypress.readline()         # Read data from
micro until newline is sent
            read = read.rstrip(' \r\n')
            time.sleep(0.3)
            readv = fluke.getMeasurement()
            tdiff =  time.clock()-t0
            readf.write("%s %s %s %s\n" %(read, volt, tdiff, 
readv))
            #print bias
            print readv
            inner_loop = inner_loop + 1
        #time.sleep(0.3)
        volt = volt - 0.1   #decrements 'current' by 0.1A
        inner_loop = 0

    #Close connection to microprocessor when user exits.
    sourcemeter.deactivate()
    cypress.terminate()
    readf.close()  #closes file so that data can be written 
from io buffer
```

Appendix D: Matlab Code for Digital Calibration

D1. 10A Current Sweep Digital Calibration

This is a portion of the Matlab code used to digitally calibrate the current measurements

```
% Author: Benedict Foo
old = ten(1:192,5); %Extract Data for processing
diff_end = max(ten(:,4))-max(ten(:,5)); %find the maximum
difference

diff_start = min(ten(1:192,4))-min(ten(1:192,5)); %find the
min difference
change = linspace(diff_start,diff_end,98); %create a vector
for difference
change_t = transpose(change);

for n = 1:98 %%add based on current level since difference
varies linearly
    new(n) = old(n)+change_t(n);
end

for n = 99:192
    new(n) = old(n)+change_t(195-n);
end
```

D2. Current Ripple Digital Calibration

The portion of code that helps digitally calibrate the current ripple measurements, this is the less robust (and accurate) but more efficient digital calibration method.

```
%% Current current ripple post processing Author: Benedict Foo

current_new = current(:,4);
top = length(current_new);
for i=1:top
    if current_new(i)<1.0
        current_new(i) = current_new(i)+0.4;

    elseif current_new(i)<1.2
        current_new(i) = current_new(i)+0.5;

    elseif current_new(i)<2.0
        current_new(i) = current_new(i)+0.57;
    elseif current_new(i)<2.3
        current_new(i) = current_new(i)+0.63;

    elseif current_new(i)<2.9
        current_new(i) = current_new(i)+0.72;

    end
end
```