# Current Control of a DC to DC Buck Converter

Willem R. Slump
s2452081

June 25, 2018

rijksuniversiteit
groningen

**Abstract**

This paper presents the design and testing of a DC to DC buck converter and a PI controller for controlling the output current of the converter. Simulations were provided by MATLAB Simulink simulation environment to test the State Space model and LTspice to simulate the circuit. The converter and its controller were tested in a laboratory setup using an Arduino microcontroller, to implement the PI controller, and a Texas Instrunments INA240A1 Current Sense Amplifier, for measuring the current.

The LC filter of the buck converter was designed to meet a maximum output current ripple of 100 mA and a maximum load voltage ripple of 50 mV. The PI controller was designed to meet bandwidth, phase margin, and gain margin requirements. The resulting controller has gain values: $k_p = 0.75$ and $k_i = 700$. The controller was shown to have 132° of closed loop and 90° of open loop phase margin. The closed loop gain margin is infinite since the closed loop phase never crosses ±180°. The settling time of the controllers tested in Simulink and LTspice are 0.25 and 0.2 ms, respectively. The rise time of the controller built in the lab is 0.2 ms, while its settling time is 0.5 ms. The digital controller realized in the lab provides a 31,250 Hz PWM signal, and has a $52\mu s$ output update lag (19.23 kHz update rate), due to the analog to digital converter.

# Contents

# 1 Introduction

DC to DC converters are of interest due to the growing demand for renewable energy sources, such as Solar Panel Systems. The particular type of DC to DC converter that will be used in this project is called a buck converter, which is a type of DC to DC converter capable of supplying any DC voltage from zero Volts up to the voltage of the power supply. This converter plays an integral role in DC microgrid systems because they regulate the current output to the load. The term "micro" is used because low-voltage, high-current networks are limited in their scale to a size much smaller than that of the traditional AC power networks. In an AC system, the voltage level is changed by using a transformer. This is done to reduce the current that the large transmission lines carry, thereby reducing the power loss. At the time of the implementation of the first power networks, there was no circuit or component which could provide a similar function in a DC network. The solution to this problem would only be made possible with the advent of the widespread use of semi-conductor technology, on which DC to DC converters heavily rely. DC to DC converters use semi-conductors such as MOSFETs (Metal Oxide Silicon Field Effect Transistors) which act as switches. Switching the power from the supply on and off many times a second is the working principle behind a DC to DC buck converter.

Solar panels naturally produce DC electricity, which is ideal for use in any electronic device that is powered by a battery. DC power transmission has the ability to provide an efficiency benefit over AC transmission [1]. This benefit is greatest when the source of the electricity produces DC voltage. Since many electronic devices use DC electricity, a DC power network using a DC source would not have to convert and re-convert the electricity, as would be necessary with an AC distribution scheme. Another key benefit to using DC power systems is that they can be used either alongside, or independently from, existing AC power infrastructure [2]. For these reasons, interest in the field of DC Microgrids is increasing. The research carried out during this BIP will contribute to the knowledge already produced on the subject of DC Microgrids, specifically contributing to the knowledge on how to implement DC Microgrids. Therefore, this research will benefit those who seek to implement the DC microgrid systems.

## 1.1 Design Topic

The topic for this Bachelor Integration Project (BIP) is the design of a controller for a DC to DC buck converter. These converters facilitate the operation of DC Microgrids, the name given to the modern approach to implementing DC power transmission networks. The DC to DC Converter will be connected to a resistor that will model the load placed on the converter by an electrical device. This is a necessary consideration in the modeling of the system because the model should reflect reality as much as possible. A proportional-integral (PI) controller will be used to achieve the stabilization and control of the system. The bulk of the work in this BIP will be spent focusing on the implementation of the system, including the converter, the controller, and load circuit. The implementation of the system will be carried out in order to determine whether the assumptions made during the design phase of the controller were accurate.

The field of scientific inquiry relevant to this project is Control Systems Engineering. Understanding concepts from Control Systems Engineering is required because the goal of the design is to stabilize and regulate the behavior of a DC to DC buck converter. For this project the control scheme will be a PI controller because their behavior is well understood in the field of Control Systems Engineering. In addition, the methods for designing a PI controller are well documented and understood. Other concepts from Control Systems Engineering that are of importance are phase and gain margin. These concepts allow the stability of the system to be rigorously quantified.

# 2 Problem Analysis

## 2.1 Problem Context

The central problem which must be solved in this BIP is the regulation of the output current of a DC to DC buck converter. Successfully controlling one DC to DC buck converter in a laboratory setting is a necessary milestone in implementing a fully functional distributed control scheme in a DC microgrid for use in an actual power distribution network. As stated in the introduction, the goal of implementing DC microgrids is to utilize their high efficiency and to use them with renewable energy sources. This project will specifically consider a DC microgrid which uses one source of power, one DC to DC buck converter, and a controller to supply energy to the circuits in one house.

## 2.2 Problem Owner Analysis

The problem owner in this case is the party that owns the DC microgrid and all corresponding components of the system. The owner of these components is the University of Groningen, more specifically the DTPA lab. The interests of the DTPA lab are represented by Prof. dr. ir. Scherpen and Michele Cucuzzella. They will ensure that the goals of the research carried out by the DTPA lab on the subject of DC microgrids are furthered as a result of this project.

## 2.3 Stakeholder Analysis

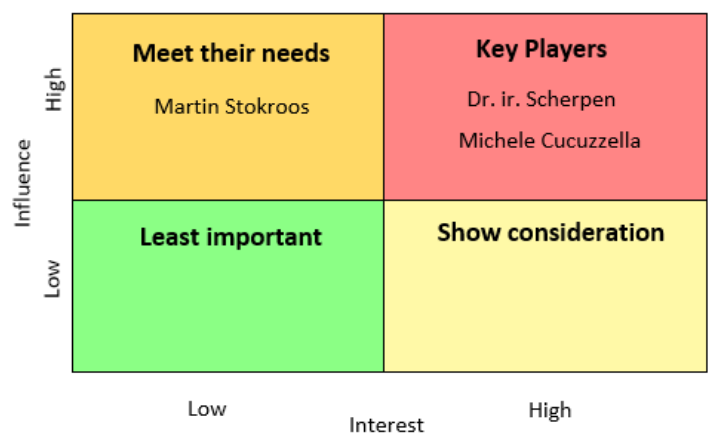



Figure 1: The stakeholders in this project organized into a matrix showing their role in the project

The stakeholders in this project are as follows:

- **Prof. dr. ir. Scherpen -** As the chair of the DTPA lab, dr. Scherpen is interested in managing the research group and ensuring the success of the research done at the lab. Dr. Scherpen is involved with research into the field of Control Systems, although not specifically into DC microgrids.
- **Michele Cucuzzella -** Michele is actively involved with research into the field of DC microgrids. He will benefit from the research carried out in this BIP because it will serve as another proof-of-concept for a simplified example of his research

- **Martin Stokroos -** As a key manager of the laboratory equipment, Martin Stokroos' role is important to the successful completion of this project. Given his position at the DTPA lab, he has a vested interest in ensuring that the equipment is utilized properly.

The decision was made to place Prof. dr. ir. Scherpen and Michele Cucuzzella in the position of 'Key Players' because they have both a high degree of interest in the proper completion of this project, as well as a great degree of influence over the project. With regard to the 'Meet their needs' category, Martin Stokroos was placed here because he has a high degree of influence over how the project will be implemented on a test circuit board. However, Martin does not have a high degree of interest in the specifics of the project in his capacity at the University of Groningen.

### 2.4 System Description

The system that will be researched in this project is a simplified DC microgrid, with a power regulator based on a buck converter. A block diagram representation of the system is provided in Figure 2, where the names of input and output signals are shown to add clarity in the diagram. The design of the buck converter circuit topology is such that the input voltage is higher than the output voltage. Other essential elements of a DC microgrid system include the power source and load network. For the purpose of this project, the power source will be a bench power supply, which is capable of providing voltages up to 150V and currents up to 4A. The system will be modelled considering that the load network may be accurately modelled as a resistive load. The value of the load is assumed to be constant in order to simplify the model of the system.

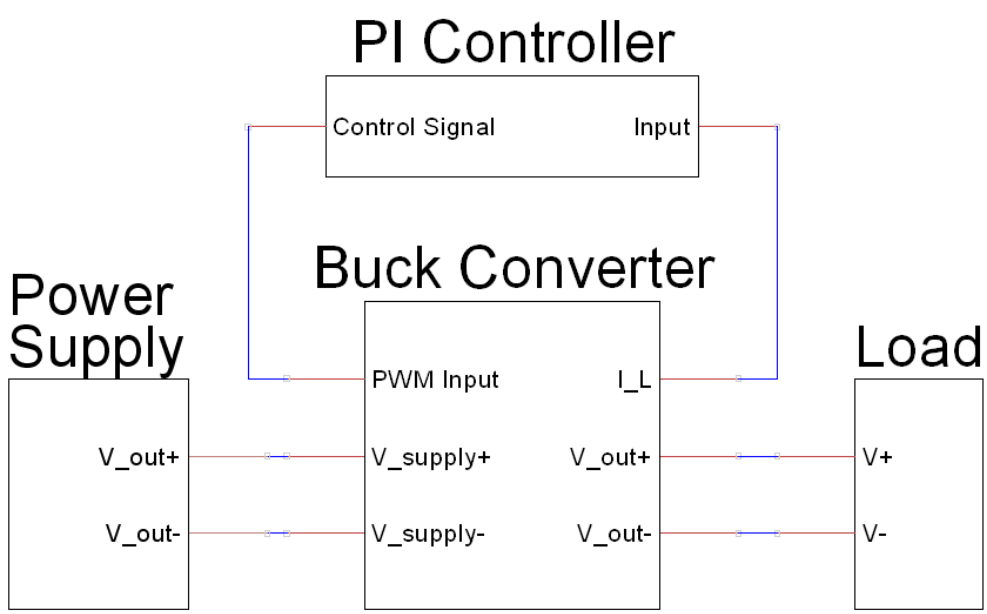



Figure 2: Block diagram showing the system, names of inputs and outputs are also included

#### 2.4.1 Buck Converter Circuit

As shown in Figure 2, the buck converter has three input connections, a connection for both the positive and negative terminals of the power source, and another input for the PWM signal

from the controller. The PWM technique will be elaborated upon in the Literature Analysis section (Section 2.5). Based on these inputs, the buck converter produces three outputs, a measurement for the inductor current of the buck converter, $I_L$, positive output voltage terminal, to be connected to the positive terminal of the load, and a negative output terminal, to be connected to the negative terminal of the load. In order to understand how the input signals translate into output signals by the buck converter, a diagram showing the buck converter in greater detail is given in Figure 3.

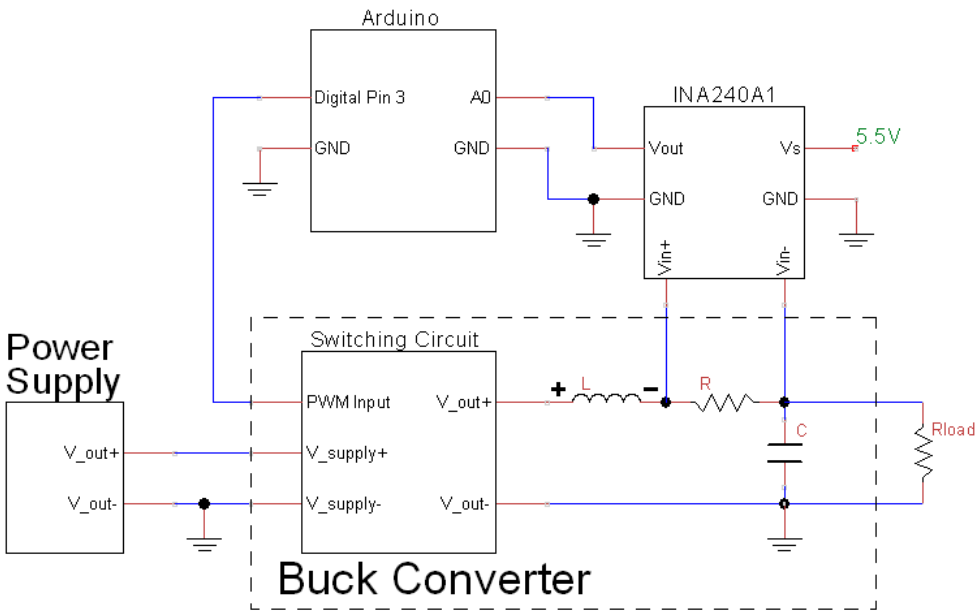



Figure 3: Diagram of the buck converter system with its controller and power source

The buck converter itself is shown as the components within the dashed line rectangle. The major components in the buck converter circuit at this level are the switching cicuit, the inductor ($L$), the capacitor ($C$), and the current sensing resistor ($R$). Despite its name, the current sensing resistor is nothing more than a regular resistor, the voltage across which will be used to determine the current through the inductor. Note that the positive terminal of the inductor is shown to the left of the component while the negative terminal of the inductor is shown to the right. The component directly above the resistor is the Texas Instruments INA240A1 chip, which will be used to provide a measurement of the voltage across the resistor. The output of the INA240A1 chip will be sent to an Arduino microcontroller as input. The controller for the buck converter circuit will be implemented on the Arduino.

### 2.4.2 Switching Circuit

The working principle behind a buck converter power regulator circuit is to switch the connection of the load to the power source on and off rapidly. The signal on the output of the switching circuit will be either the voltage of the power source, $V_s$, or the voltage of the ground, 0V. In principle, if the switching occurs faster than the load has time to naturally respond, the voltage on the output will be the average of the switching output signal over the switching period. Two switches are needed to implement the switching circuit, shown in Figure 4 as $TR_1$ and $D_1$.

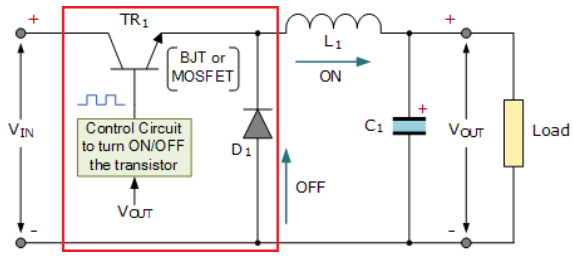



Figure 4: Switches of the circuit implemented as discrete semiconductor elements

In a typical buck converter the two switches are complementary, meaning that the two switches are never both in the same state. As shown in Figure 4, a diode can be used with a transistor of either BJT or MOSFET type. Assuming a MOSFET transistor is used for $TR_1$, if the MOSFET is switched on, a connection is made between the voltage source and the inductor, allowing current to flow. At this time the voltage potential of the positive terminal of the inductor is positive with respect to ground. Therefore, the diode in this configuration is reverse biased, and no current will flow.

Conversely, if the MOSFET is switched off, the voltage on the positive terminal of the inductor instantaneously drops to a negative voltage due to the current build up in the inductor. Since the positive terminal of the diode is connected to ground, which has a voltage of 0V by definition, and the negative terminal of the diode is connected to a relative negative voltage, the diode becomes forward biased and switches on. Therefore, it is clear that the state of the diode switch is dependent on the transistor switch. As such, the transistor switch will be called the principal switch in the system and the diode switch will be called the secondary switch.

## 2.5 Literature Analysis and Background

### 2.5.1 Kirchoff's Voltage and Current Laws

Kirchoff's Voltage Law (KVL) and Kirchoff's Current Law (KCL) are important to the work in this BIP because they provide a mathematical framework for translating an electric circuit diagram into a system of equations which describe relationship between the voltages and currents in an electrical network. For this reason, establishing a set of KVL and KCL equations is the first step in finding the dynamic equations for a circuit. The goal of the mathematical analysis of the buck converter circuit is to establish a valid State Space model for the buck converter. Therefore, the KCL and KVL equations are the first step towards formulating the State Space model for the buck converter.

Kirchoff's Voltage Law states that the sum of the voltages measured in any closed loop of an electrical circuit must be zero. Kirchoff's Current Law states that the sum of all currents entering any node of the circuit must equal zero. This implies that certain currents will have positive sign and others will have negative sign. Since electrical current describes the flow of electrical charge through a circuit, the direction of the current flow determines its sign. It is important to note that Kirchoff's Current Law holds independently of which currents are considered positive and which are considered to be negative flows, as long as the reference is consistent for all nodes in the circuit.

Generally, the reference for the positive and negative current flows are defined in terms of the incoming and outgoing flows. For the purpose of the circuit analysis, currents flowing into a node are considered to be positive with respect to that node and currents flowing out of a node are considered to be negative with respect to that node. The precise mathematical formulation of the

Kirchoff's Voltage and Current Law equations can be found in the next section.

### 2.5.2 Mathematical Formulation

Consider:

L, the set of all closed loops in the circuit

N, the set of all nodes in the circuit

$$\underbrace{\sum_i V_i = 0, \quad \forall i \in L}_{Voltage\ Law} \qquad \underbrace{\sum_j I_j = 0, \quad \forall j \in N}_{Current\ Law} \tag{1}$$

Furthermore, it is important to mention that the entire set of equations generated by the KVL and KCL analysis of the system is larger than the minimal set of equations required to solve the system. That is to say, one does not need to consider the KVL and KCL equations for all loops and all nodes of the circuit. Instead, the number of equations, resulting from KVL and KCL, that are considered in the model should equal the number of system states. As a heuristic to the problem of choosing the right KVL and KCL equations to consider the following rules should be applied:

1. Apply KCL to nodes connecting three or more components
2. KCL is not required for the ground node
3. Apply KVL to circuit loops until the number of equations equals the number of variables

### 2.5.3 Fundamental Circuit Elements and Equations

While the KVL and KCL equations are useful to determine how the voltages and currents in an electrical network relate to one another, they do not provide insight into how these voltages and currents will develop over time. In order to fully understand the dynamic behavior of a given electrical network, the equations which specifically describe the behavior of the network's constituent components must be properly substituted into the set of KVL and KCL equations. The three fundamental passive elements in any circuit are resistors, capacitors, and inductors. These circuit elements are considered passive elements because they dissipate, rather than generate power. The dynamics of these elements are given by the following equations:

$$\underbrace{V = I \times R}_{Resistor}, \quad \underbrace{I_C = C \times \frac{dV_C}{dt}}_{Capacitor}, \quad \underbrace{V_L = L \times \frac{dI_L}{dt}}_{Inductor} \tag{2}$$

### 2.5.4 Article: Decentralized sliding mode voltage control in DC microgrids

This article proposes a model for a DC microgrid based on Distributed Generation units (DGus)[3]. Instead of considering a network of DGus, this project will focus on researching a circuit which is equivalent of a single DGu. Thus, the mathematical model relevant to this BIP, while more simple than that proposed in this paper, will necessarily follow similar logical steps in the formulation of the model. The article also points out that control systems for DC-based power systems are less complex than control systems for AC power distribution systems. This is because issues with synchronizing the voltages produced by multiple sources and regulating the frequency

of the grid are challenging problems that need solving in AC power systems. DC power systems, on the other hand, do not work by transmitting the power as a wave down the transmission cables and, therefore, avoid these challenging aspects.

#### 2.5.5 Pulse Width Modulation

The output current of the converter will be regulated through a technique known as pulse width modulation (PWM). The controller in the system will provided the buck converter with a series of pulses at a set frequency. These pulses are defined as being either on, and thus at maximum voltage, or off, and thus at zero voltage. The variable in a pulse width modulated signal is the proportion of the time the voltage is on. This quantity is represented by the 'duty cycle' of the wave. The buck converter will be controlled by varying this duty cycle each period of the PWM wave.

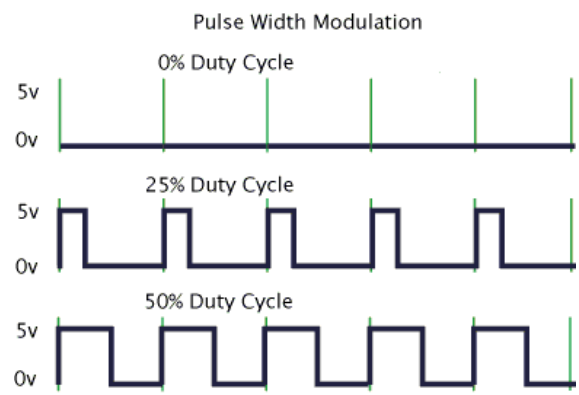



Figure 5: Example Pulse Width Modulation signals with varying duty cyles

# 3 Design Goal and Scope

## 3.1 PI Controller

The goal of this BIP is to control the output current of a DC to DC buck converter. The regulation of the output current around a desired reference current will be accomplished using a PI controller to be designed during the BIP. In order to ensure that the controller works well, a number of design constraints will be imposed:

1. Settling time under 0.5 seconds: $t_s \leq 0.5s$
2. Less than 5% overshoot: $M_o \leq 1.05$
3. Zero steady state error

In addition to these constraints, focus must be placed on ensuring the stability of the system. Stability can be quantified using analysis in the frequency domain by using Bode plots and Nyquist plots to quantify the gain and phase margins of the system. The exact phase and gain margin requirements are difficult to quantify, but it can be said that these margins should be as large as possible, while maintaining the design constraints set forth above.

## 3.2 Buck Converter

The buck converter design presented in this report is meant to be used with the PI current controller. The converter will be built and tested in the laboratory provided by the DTPA lab. The goal of the design of this converter is to provide empirical evidence that the PI controller will work as designed. In order to quantify the design goal of the buck converter the following specifications must be met:

1. Input: 12 Volts

2. Output: 1 Ampere of current at 5 to 12 Volts
3. Switching frequency $f_{sw}$ = 31,250 Hz
4. Maximum output current ripple 100mA

### 3.3 Scope

This BIP will focus on the design of a controller for a simple DC microgrid, where the goal is to regulate the output current of the DC to DC buck converter. Additionally, importance will be placed on validating the design of the controller. Since the mathematical model used to describe the system will necessarily be a simplification of the real world, the validation of the controller is key. The validation process will consist of comparing the predictions made with the mathematical model and simulation with the experimental results. It will be necessary to compare the results of a number of reference inputs to the system as well as comparing the stability predictions.

## 4 Research Questions

*How should a proportional-integral (PI) controller be designed for regulating current in a DC microgrid, and is the behavior of this controller validated by experimental results?*

### 4.1 Sub-questions

In order to answer the main research question, the following sub-questions were designed:

1. Which method of designing a PI controller should be used?
2. Does the PI controller maintain system stability when the input reference is disturbed?
3. Does the PI controller maintain stability when the system parameters are altered?
4. Do the simulation results from MATLAB Simulink match the results from LTspice?
5. Do the experimental results reflect the simulation results?

## 5 Cycle Choice


Figure 6: The three-cycle model as described by Hevner

In Figure 6 is an illustration of the three-cycle model as described by Hevner [4]. Hevner outlines the activities of design science research into the three categories of the Relevance Cycle, the Design Cycle, and the Rigor Cycle. The idea behind the model is to understand the field of design science research as being broken down into three cycles, which feed from and into one another. For instance, the Rigor Cycle is where the most abstract research occurs. The Rigor cycle is closely related to the experimental cycle because it starts with a gap in knowledge and produces experimental/test results. The link between the Rigor Cycle and the Design Cycle is that information gathered in the Rigor Cycle is applied to achieve the design of a desired artifact or construct. Understanding what the desired artifact is, and therefore what the goal of the Design Cycle should be, is the focus of the Relevance Cycle. The Relevance Cycle starts with the recognition of a need or demand, and through analysis of the situation, chooses the artifact or construct designed in the Design Cycle that best meets the needs or demands identified at the start of the Relevance Cycle.

Given this description of each of the three cycles, the Design Cycle can be identified as the applicable cycle of design science research. The Design Cycle begins with a goal and, after outlining and validating the design, results in a construct that can be used in the Relevance Cycle. This is the same process that is required for the proper completion of this BIP project. Given the fact that the steps needed to properly design the DC to DC controller are congruent to the steps of the Design Cycle, it can be concluded that this project is a Design Cycle project.

# 6 Methods

This BIP will make use of literature research and mathematical modeling as research methods. Literature research will be necessary to fill any knowledge gaps recognized before commencing work on the mathematical model and during the simulation/experimental phase of the project. The mathematical model will be used to design the PI controller which will regulate the system. Simulations and experiments will be used to provide a validation of the design. The goal is to verify that the predictions made with the mathematical model and simulations are accurate for the real system. If the results of the theoretical predictions made with the mathematical model, the simulation predictions, and the experiments are the same, then it is a strong indication that the controller is properly designed.

## 6.1 Mathematical

The central mathematical model in this BIP is a State Space model, which expresses a system in terms of a system of first order linear differential equations. State Space models are used to provide a general description which can be used for systems in many different domains, including the mechanical, electrical, and hydraulic domains, for example. The State Space model relevant to the DC microgrid being modeled in this BIP will be used to find its transfer function. The transfer function of the DC microgrid is important in designing the controller and for verifying the its stability.

## 6.2 Simulation

Additionally, simulations will be carried out to provide additional evidence that the controller is properly designed. Simulations are helpful because they allow an engineer to test the design in a more realistic setting without needing to design an experimental setup. In this way, the simulations carried out on the system in this BIP will provide additional certainty that the controller will work when implemented in an experimental setting. The simulation resources which will be used for this

project are the MATLAB® Simulink package and LTspice. Simulink will provide an environment to simulate the State Space model found during the research associated with this project and LTspice will be utilized to obtain circuit simulations, which will be used to better understand the filter and load circuit component of the DC microgrid.

### 6.3 Experimental

The implementation of a DC microgrid controller will be done in the laboratory provided by the DTPA lab. An Arduino micro-controller will be used to provide the switching signal for the DC to DC converter and, thus, will be used for implementing the controller as well. A test circuit will be built in the DTPA lab and experimentation will commence. An oscilloscope will be used to take measurements needed to understand the behavior of the system.

## 7 State Space Model

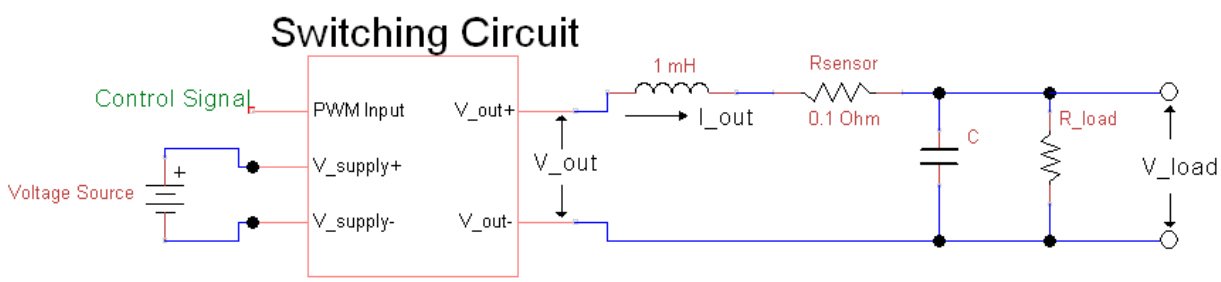



Figure 7: The description of the buck converter on which the State Space model is based

In Figure 7 a depiction of the buck converter system is given. In this case, the buck converter is described as a switching circuit and an accompanying LC filter connected to a DC load. The output of the 'Switching Circuit' is considered to be a pulse-width-modulated square wave with the same frequency and duty-cycle as the control signal. When the switch is on, the voltage of the output of the switching circuit is the same as the voltage of the source, appropriately labeled 'Voltage Source'. The inductor and the capacitor shown in Figure 7 serve as a low-pass filter, which is necessary to transform the high-frequency PWM signal, provided by the output of the switching circuit, into the average voltage over the period of the PWM signal.

In this case, the switching circuit considered will switch between two states. The 'ON' state of the switch represents the state of the circuit when a connection is made between the positive terminal of the inductor, as shown in Figure 7, and the voltage source. The 'OFF' state represents the state in which the positive terminal of the inductor is connected to the ground node.

At this level of analysis it is possible to assume that the switching circuit is approximated by an ideal switch which reacts instantaneously upon input from the control signal. Non-ideal aspects of the behavior of the capacitor, such as equivalent series resistance (ESR), are not considered in this section. In the previous section, where the values of each component in the circuit will be found, a more in-depth analysis of the capacitor will be provided. With respect to the non-ideal elements of the inductor's behavior, the ESR of the inductor can be modelled with the sensor resistor, $R_{sensor}$. The equivalent parallel capacitance of the inductor is not considered. Since the circuit is switching between two states, the average value of the system states over the switching period will be considered.

## 7.1 Notation

Let:

$V_s$ denote the voltage of the power source
$V_{out}$ denote the output voltage of the switching circuit
$I_{out}$ denote the output current of the switching circuit
$V_R$ denote the voltage of the load
$I_L$ denote the current through the inductor

### 7.1.1 ON State

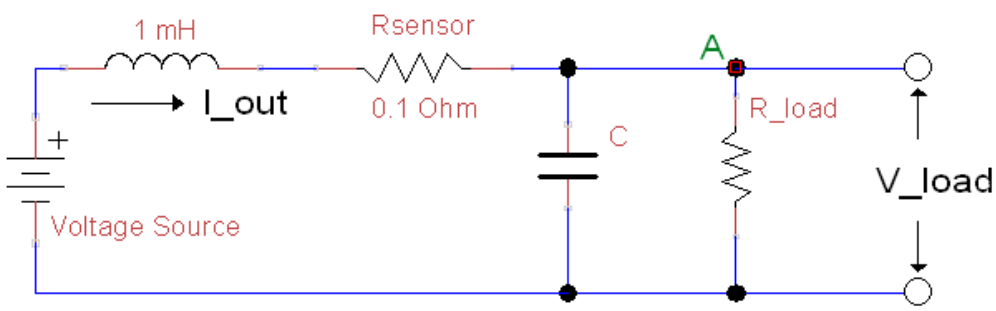



Figure 8: Equivalent circuit of the buck converter when the switch is on

When the switch of the buck converter circuit is on, the switching circuit makes a connection between the voltage source and the rest of the circuit. In the ideal case, there is no voltage drop between the output of the switching circuit and the voltage source. The KCL equation for this circuit was based on considering the currents going through node A, as depicted in Figure 8. Additionally, the KVL equation was formulated using the outside loop of the circuit.

$$\underbrace{I_{out} - I_C - I_{R_{load}} = 0,}_{KCL} \qquad \underbrace{V_s - V_L - V_{R_{sensor}} - V_{R_{load}}}_{KVL} = 0 \tag{3}$$

The element equations now need to be substituted into Equation (3) in order to have a complete system of equations which can be formulated into a State Space model. The resulting model should include the supply voltage $V_s$, the inductor current $I_L$, and the voltage of the load $V_R$. Substituting the element equations from (2) into the equations from (3), keeping in mind that $I_{out} = I_L$ and $V_C = V_R$:

$$I_{out} - C\dot{V}_R - \frac{V_R}{R} = 0 \quad \longrightarrow \quad C\dot{V}_R = I_{out} - \frac{V_R}{R} \tag{4a}$$

$$V_s - L\dot{I}_L - I_L Rsensor - V_R = 0 \quad \longrightarrow \quad L\dot{I}_L = V_s - I_L Rsensor - V_R \tag{4b}$$

It is now possible to formulate the State Space equations from Equation (4a) and (4b) by considering the state variable, $x = [I_L \quad V_{R_{load}}]^T$, system input, $u = V_s$, and system output $y = I_L$:

$$\begin{aligned} \dot{x}_{on} &= \begin{bmatrix} -\frac{R_{sensor}}{L} & -\frac{1}{L} \\ \frac{1}{C} & -\frac{1}{RC} \end{bmatrix} x_{on} + \begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix} u \\ y_{on} &= \begin{bmatrix} 1 & 0 \end{bmatrix} x_{on} \end{aligned} \tag{5}$$

### 7.1.2 OFF State

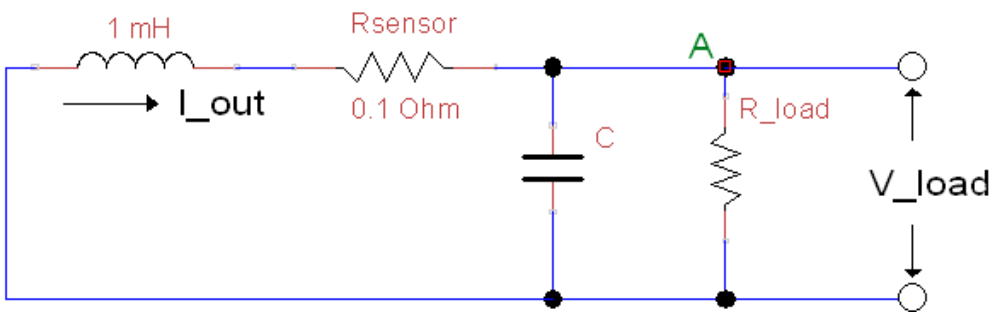



Figure 9: Equivalent circuit of the buck converter when the switch is off

As shown in Figure 9, the diagram of the converter in the off state includes a short between the positive terminal of the inductor and ground. In this case, the voltage of the positive terminal is 0 Volts and the voltage of the negative terminal is $V_{load}$. Since $V_{load}$ always has a positive voltage with respect to ground, the voltage drop across the inductor is negative with respect to the convention defined in the ON State section. Therefore, the inductor voltage term has negative sign in the KVL equation for the circuit in this state.

$$I_{out} - I_C - I_{R_{load}} = 0 \tag{6a}$$

$$-V_L - V_{R_{sensor}} - V_{R_{load}} = 0 \tag{6b}$$

Applying the same substitution method as in Section 7.1.1 yields:

$$I_{out} - C\dot{V}_R - \frac{V_R}{R} = 0 \quad \longrightarrow \quad C\dot{V}_R = I_{out} - \frac{V_R}{R} \tag{7a}$$

$$-L\dot{I}_L - I_L Rsensor - V_R = 0 \quad \longrightarrow \quad L\dot{I}_L = -I_L Rsensor - V_R \tag{7b}$$

Now from Equations (4a) and (4b) the State Space model for the converter in the OFF state can be derived, considering $x = [I_L \quad V_{R_{load}}]^T$, $u = V_s$, and $y = I_L$:

$$\begin{aligned} \dot{x}_{off} &= \begin{bmatrix} -\frac{R_{sensor}}{L} & -\frac{1}{L} \\ \frac{1}{C} & -\frac{1}{RC} \end{bmatrix} x_{off} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} u \\ y_{off} &= \begin{bmatrix} 1 & 0 \end{bmatrix} x_{off} \end{aligned} \tag{8}$$

### 7.1.3 Average Model

At this stage of the circuit analysis two State Space models have been derived for the behavior of the system when the switch of the circuit is on and when the switch is off. In order to obtain one model which describes the system dynamics, the average of these two models will be taken over the switching period, for a full derivation see Appendix A.

$$\begin{aligned} \dot{x} &= \begin{bmatrix} -\frac{R_{sensor}}{L} & -\frac{1}{L} \\ \frac{1}{C} & -\frac{1}{RC} \end{bmatrix} x + \begin{bmatrix} \frac{d}{L} \\ 0 \end{bmatrix} u \\ y &= \begin{bmatrix} 1 & 0 \end{bmatrix} x \end{aligned} \tag{9}$$

Where $x$ represents the average state of the system over the switching period and $u$ represents the average input over the switching period. The average value of the input can also be expressed in terms of the input voltage $V_s$ and the proportion of the switching period when the switch is on. Keeping in mind that the duty cycle of the switch is the proportion of the switching period when the switch is on, $u = V_s * d$. Considering the input from the controller to the system will represent the duty cycle of the switch, as discussed in Section 2.4, the input in the model can be changed to $u = d$. This results in the following model:

$$\begin{aligned} \dot{x} &= \begin{bmatrix} -\frac{R_{sensor}}{L} & -\frac{1}{L} \\ \frac{1}{C} & -\frac{1}{RC} \end{bmatrix} x + \begin{bmatrix} \frac{V_s}{L} \\ 0 \end{bmatrix} u \\ y &= \begin{bmatrix} 1 & 0 \end{bmatrix} x \end{aligned} \tag{10}$$

### 7.2 System Transfer Function

Now that the State Space model for the system is derived, it is possible to obtain the transfer function of the system using the following formula:

$$H(s) = C(sI - A)^{-1}B \tag{11}$$

Using this method the transfer function for this system is found to be (full derivation in Appendix B):

$$H(s) = \frac{V_s}{L} \frac{s + \frac{1}{R_{load}C}}{s^2 + s\left(\frac{1}{R_{load}C} + \frac{R_{sensor}}{L}\right) + \frac{1}{LC}} \tag{12}$$

## 8 Buck Converter Design

### 8.1 Circuit Topology

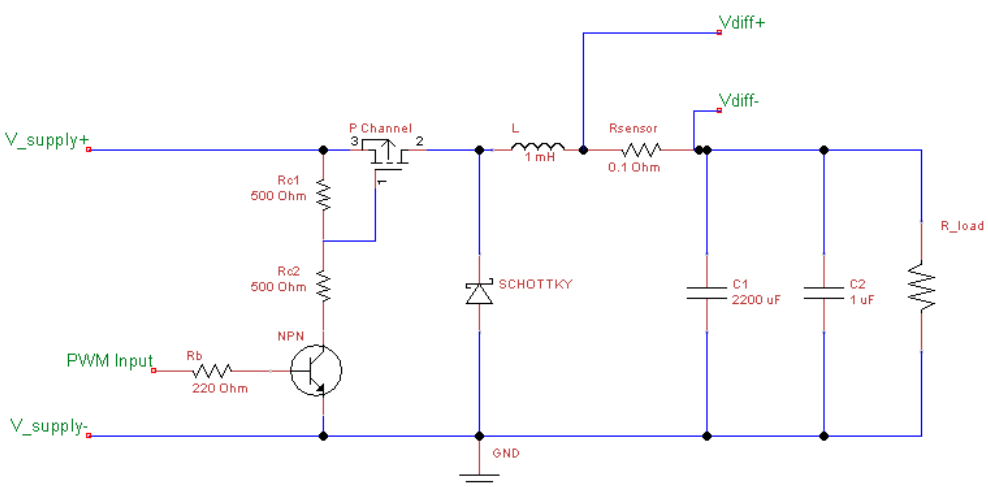



Figure 10: The diagram of the buck converter used for experimentation

Figure 10 shows the topology of the circuit that will be built for experimentation. This is also the circuit topology on which the controller will be based.

### 8.2 High-Side Switch

A MOSFET-diode pair was used to implement the switching behavior of the buck converter. In Figure 10, the MOSFET is seen at the top left of the diagram, labeled 'P Channel'. The diode is found above the ground connection in the diagram. Both the MOSFET and the diode can be seen as switches for the purpose of this analysis, where if the MOSFET is on, the diode will be off, and vice-versa.

The decision was made to implement the circuit with a p-channel MOSFET so that the source of the transistor could be connected to the supply voltage. This is desirable since MOSFETs switch between conducting and not conducting based on the gate to source voltage, $V_{gs}$. A p-channel MOSFET typically requires a gate to source voltage more negative than approximately 2 Volts, although this is a minimum value. This means that if the voltage on the gate of the MOSFET (shown as terminal '1' in the figure) is pulled more than 2 Volts below the supply voltage, the MOSFET will start conducting current. Conversely, the gate must be driven by a source capable of bringing the gate voltage back to the supply voltage.

In the practical circuit, an Arduino microcontroller will be used to provide the switching signal to the gate of the MOSFET. However, it must be taken into account that the maximum voltage range of the Arduino output is 0 to 5 Volts, and the maximum current output is approximately 40 mA. The input voltage of the converter, as defined in Section 3.2, is 12 Volts, meaning the raw output of the Arduino is insufficient to switch the MOSFET into the off state. Therefore, a simple gate driver circuit must be developed.

### 8.3 Driver Circuit

A simple gate driver circuit based on an NPN transistor was used to drive the gate voltage of the MOSFET. The NPN transistor when switched on will allow current to flow through through its collector, and thus through the two resistors connected between the voltage supply and the collector. When the transistor is behaving linearly, the relation between the base current and the collector current is:

$$I_C = I_B \times h_{FE} \tag{13}$$

The expression for the collector current given in Equation (13) will hold until the collector current, $I_C$, starts being limited by the value of the resistors connected in series with the collector. In this case, the maximum collector current, assuming zero collector-emitter voltage, is:

$$I_{C_{max}} = \frac{V_s}{R_{C_1} + R_{C_2}} \tag{14}$$

As long as the current that the transistor is trying to force through the collector, given by Equation (13), exceeds the limit on the collector current due to the resistors, given by (14), the transistor can be considered as an ideal switch. The equation for the base current is:

$$I_B = \frac{V_{arduino} - V_{be}}{R_B} \tag{15}$$

Considering (15), the base resistor, $R_B$ should satisfy:

$$\frac{V_{arduino} - V_{be}}{R_B} \times h_{FE} \geq \frac{V_{supply}}{R_{C_1} + R_{C_2}}$$

$$\frac{R_B}{(V_{arduino} - V_{be}) \times h_{FE}} \leq \frac{R_{C_1} + R_{C_2}}{V_{supply}}$$

$$R_B \leq \frac{(V_{arduino} - V_{be})\,(R_{C_1} + R_{C_2}) \times h_{FE}}{V_{supply}} \tag{16}$$

This means that the gate voltage will transition from the full supply voltage when the NPN is off, and half the supply voltage when the NPN is switched on. Assuming the supply voltage is large enough such that half the supply voltage exceeds 2 Volts the MOSFET will switch on and off in accordance with the NPN.

The values of $R_{C_1}$ and $R_{C_2}$ should be large enough such that they do not draw a significant amount of power from the load, however if they are too large then the switching speed of the MOSFET will suffer. A value of 500Ω is a good middle ground. The Arduino digital output pin is capable of supplying a voltage of 5V, the base-emitter voltage, $V_{be}$, of the transistor is assumed to be 0.7V, and the specified supply voltage is 12V. The datasheet of the BC547, the NPN transistor which will be used in the gate driver circuit indicates that the lowest typical value for the $h_{FE}$ of the transistor is 110 [5]. To check whether $R_B = 220\Omega$ will satisfy the condition in (16), all the aforementioned values are substituted in:

$$220\Omega \leq \frac{4.3 \cdot 1000 \cdot 110}{12}$$
$$220\Omega \leq 39.416k\Omega$$

Since this inequality holds, the value of 220Ω for the base resistor of the NPN is a valid choice.

## 8.4 Operating Load

One of the essential components of the buck converter system is the load connected across the output of the converter because the purpose of a buck converter circuit is to regulate the voltage across or current flowing to the load. In the simplest case, the load on the buck converter can be modelled by an ideal resistor with a constant value. However, if the buck converter and its controller can only be verified to be stable when the load is constant, the results will not be applicable to the general scenario where the load on the converter is subject to change. Ultimately, the goal is to show that the converter, regulated by the controller, can be used to regulate the power to a network of light bulbs in a home. Since the lights of houses are subject to be switched on and off arbitrarily, the converter must be shown to be robust against large, instantaneous, changes in the load.

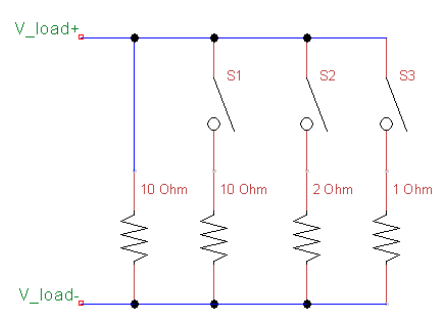



Figure 11: The load network built for the purpose of testing load disturbance rejection of the controller

In order to test the converter in this scenario, a network of resistors was built, shown in Figure 11. The resistance of the load can be changed by flipping the switches (S1,S2, and S3) on and off independently from one another. Given that there are three switches, a total of 8 combinations can be made, each providing a slightly different load.

## 8.5 LC filter

As mentioned briefly in Section 7, the LC filter circuit will serve as a low-pass filter on the output of the switching circuit. Since the precise behavior of the LC filter depends on the value of the resistor to which it is connected, a constant load of 10Ω will be considered. Additionally, since the goal of the controller is to regulate the current of through the inductor of the buck converter, the value of the inductor will be determined independently from the value of the load resistor and the capacitor. Once a specification for the inductor is given, the value of the capacitor will be found such that the PWM switching frequency is much higher than the resonance of the LC circuit.

### 8.5.1 Value of the Inductor

According to the design requirements of the buck converter in Section 3.2, $\Delta I_{pp_{max}}$ should be 100 mA. As can be seen in Equation (17), the value of the supply voltage, $V_s = 12V$, and the switching frequency, $f_{sw} = 31,250$Hz are important parameters in choosing the appropriate value of the inductor. The formula for the value of the peak-to-peak current ripple through the inductor is given by:

$$\Delta I_{pp} = \frac{V_s DT\,(1-D)}{L} \tag{17}$$

In order to find the minimum value of the inductor that ensures the ripple current stays below 100 mA, a duty cycle of $D = 0.5$ will be considered. This is because the maximum ripple current, $\Delta I_{pp_{max}}$ is found when $D = 0.5$. Rearranging Equation (17) to solve for the inductor and substituted $D = 0.5$:

$$\begin{aligned} L &= \frac{V_s T}{4\ \Delta I_{pp_{max}}} \\ L &= \frac{12 \times 32 \times 10^{-6}}{4\ \ \times 100 \times 10^{-3}} \\ L &= 0.96 \times 10^{-3} \quad [mH] \end{aligned} \tag{18}$$

Given the result of Equation (18), the value of the inductor can be rounded to 1 mH.

### 8.5.2 Value of the Capacitor

Since the value of the inductor has now been determined, and the value of the resistor is considered to be constant, the only degree of freedom left in the design of the LC filter is the value of the capacitor. Two important parameters of the capacitor will be considered in this analysis: the capacitance of the capacitor, as well as the equivalent series resistance (ESR) of the capacitor. The capacitance of will be chosen by analyzing the frequency characteristics of the LC filter with a 10Ω load connected to the output. Once this is done, the maximum ESR of the capacitor will be specified in order to mitigate the output ripple current.

**Frequency Characteristics** – An RLC circuit is an example of a second order system. As such, the system will resonate if the switching frequency is located near the resonance frequency of the RLC circuit. To prevent this, the resonance frequency of the RLC circuit should be taken into account when designing the filter for the buck converter. Given the formula for the resonance frequency of an RLC circuit in this configuration, see Appendix C for the full derivation:

$$\omega_{res} = \sqrt{\frac{1}{LC} - \frac{1}{2\left(RC\right)^2}} \tag{19}$$

One can then consider that the switching frequency should be some arbitrary multiple of the resonance frequency of the circuit, such that:

$$\omega_{sw} = r \times \omega_{res} \tag{20}$$

If the value of the ratio 'r' is significantly larger than 1, the resonance frequency will be much lower than the switching frequency and the circuit will not resonate due to the PWM. To simplify Equation (19), one can assume that the resistor will have little effect on the resonance frequency of the RLC circuit. Stated mathematically:

$$\frac{1}{LC} \gg \frac{1}{2\left(RC\right)^2} \tag{21}$$

Therefore, it is valid to assume that the resonance frequency of the circuit is given by $\frac{1}{\sqrt{LC}}$. The validity of this assumption can be checked after C is calculated. Combining Equations (19) and (20):

$$\omega_{sw} = r \times \sqrt{\frac{1}{LC}} \tag{22}$$

$$C = \frac{r^2}{L\left(2 * pi * f_{sw}\right)^2} \tag{23}$$

Given a switching frequency of 31,250 Hz, and a ratio 'r' of 300, and a value of 1mH for the inductance (calculated in Section 8.5.1) the capacitor should have a value of 2.33 mF. This value is closely approximated by using a capacitor with 2.2 mF of capacitance for the real circuit.

With respect to the accuracy of the assumption made, given in (21), the actual resonance frequency should be calculated. Given that the capacitance value has changed in order to take into account the capacitor which will be used to realize the circuit, a new value for the estimated resonance frequency should also be calculated. Assuming that $\omega_{res} = \frac{1}{\sqrt{LC}}$, the estimated resonance frequency is 674.2 rad/s, or 107.3 Hz. Taking into account the values $L = 1$mH, $C = 2.2$mF, and $R = 10\Omega$, the actual resonance frequency of the circuit, calculated using Equation (19), is 107.18 Hz. Given that the difference between the estimated resonance frequency and the actual resonance frequency is 0.12 Hz, we can conclude that the approach taken is valid in this scenario and that the value of 2.2 mF for the capacitor can be used.

**Equivalent Series Resistance** – The formula which can be used to determine the specification for the maximum ESR of the output capacitor is [6]:

$$\Delta V_{pp} = \Delta I_{pp} \times R_{esr} \tag{24}$$

Equation (24) holds in the case that both of the following conditions also hold:

$$RC > \frac{T_{on}}{2} \quad RC > \frac{T_{off}}{2}$$

Since in the case of the converter for this project $T_{on}$ and $T_{off}$ are complementary, these two conditions can be condensed into one:

$$RC > \frac{T_{sw}}{2} \tag{25}$$

In Equation (25), $T_{sw}$ is the switching period of the PWM signal. Given $R = 10\Omega$, $C = 2.2$ mF, and $T_{sw} = 32\mu s$, the condition in (25) can be checked.

$$5 \cdot 2.2 \times 10^{-3} > \frac{32 \times 10^{-6}}{2} \quad \longrightarrow \quad 22 \times 10^{-3} > 16 \times 10^{-6}$$

Since the condition from (25) holds, the Equation (24) can be used to accurately calculate the voltage ripple due to the ESR of the capacitor. Using this equation, a maximum rating on the ESR of the capacitor can be given. For instance, assuming a maximum ripple current of 100 mA in the worst case scenario, and a desired maximum load voltage ripple of 50 mV, the ESR of the output capacitor should be less than $0.5\Omega$.

# 9 PI Controller Design

## 9.1 Definitions

The transfer functions of the controller $C(s)$, the open loop system $L(s)$, and the closed loop system $G_{cl}(s)$ are given below:

$$C(s) = k_p + \frac{k_i}{s} \tag{26}$$

$$L(s) = C(s) \times H(s) \tag{27}$$

$$G_{cl}(s) = \frac{L(s)}{1 + L(s)} \tag{28}$$

## 9.2 Requirements

To summarize the requirements set forth in Section 3.1, the settling time of the system should be less than 0.5s, the overshoot of the system should be less than 5%, and the steady state error should be zero. As aditional stability requirements, the phase margin of the open loop system should be more that 60 °and the gain margin should exceed 40 dB. The requirements expressed mathematically:

$$\begin{aligned} t_s &\leq 0.5[s] \\ O_M &\leq 1.05 \\ e_{ss} &= 0 \\ \phi_M &\geq 60[^\circ] \\ G_M &\geq 40[dB] \end{aligned}$$

### 9.3 Design Method

The method for tuning the proportional and integral gains for a buck converter which proved useful is outlined in the following steps.

1. Start with $k_p = 0$, and raise the integral gain, $k_i$ until the output oscillates strongly around the setpoint
2. Raise $k_p$ from zero incrementally until the oscillations are gone
3. If the initial undershoot is too high, raise the value of $k_i$ and continue to tune $k_p$
4. Repeat this process until the desired behavior is achieved

After the tuning process was concluded, the proportional gain was 0.75, and the integral gain was 700. A bode plot of both the open loop and closed loop system is shown in Figure 12. In Figure 13, it is apparent that the closed loop system has a phase shift of $-48°$ at the gain crossover frequency, while the open loop system has a phase shift of $-90°$. This means that the phase margin for the closed loop system is 132°, and the phase margin for the open loop system is 90°. Since both of these phase margins are larger than the minimum required phase margin of 60°, the controller satisfies this design requirement. The closed loop system also has infinite gain margin because the phase shift of the open loop system never crosses $\pm 180°$.

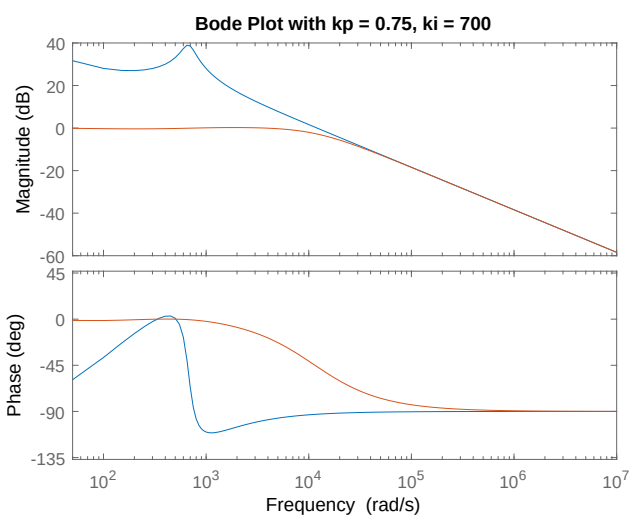



Figure 12: Bode plot of the open loop system (shown in blue) and the closed loop system (shown in red)

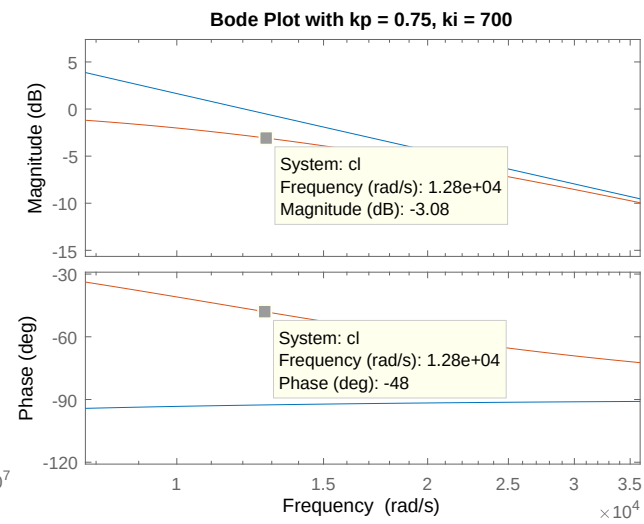



Figure 13: The same bode plot zoomed in to specifically show the bandwidth and phase margin of the system

## 10 Implementation Details

### 10.1 Current Sensing

The measurement of the inductor current will be realized using the INA240A1 Current Sense Amplifier from Texas Instruments. The INA240A1 chip works by placing a resistor in series with the inductor and amplifying the voltage drop of the resistor, effectively mapping the current into a

voltage. Later, the voltage on the output pin on the chip can be used to calculate what the current through the inductor is.

The voltage drop of the resistor can be read by the chip because resistor is then connected in parallel with the input of the differential amplifier, and hence is also called the shunt resistor. The INA240A1 chip has a voltage amplification factor of 20 [7]. Furthermore, the chip has two reference inputs which can be used to change the operating mode of the amplifier. The default setting, and the setting used for experimentation, is referencing the voltage across the shunt resistor with half the supply voltage [8]. This means that the output voltage of the chip is 20 times the differential voltage of the amplifier plus half the supply voltage. The expression for the output voltage of the chip, using the terminology of the datasheet, is:

$$V_{out} = 20 \times (V_{in+} - V_{in-}) + \frac{V_s}{2} \tag{29}$$

Equation (29) implies that the output voltage of the chip, $V_{out}$, is bound on the lower end by $\frac{V_s}{2}$. This is due to the fact that in no circumstance of the operation of the circuit will the difference $V_{in+} - V_{in-}$ be negative. On the other hand, the output voltage of the chip is restricted on the upper end by the supply voltage given to the chip. According to the maximum rating of the INA240A1 chip, the maximum supply voltage is 5.5 Volts [9]. Interestingly, this leads to a limit to the current which can be read by the INA240A1 chip, which is a function of the choice of shunt resistor. The shunt resistor chosen has a value of 0.1Ω, so the maximum current that the chip can map into a voltage without clipping the output:

$$\begin{aligned} V_{in+} - V_{in-} &= I_L \times R_{shunt} \\ I_{max} &= \frac{V_s - 0.5V_s}{20 \times R_{shunt}} = \frac{2.75\ Volts}{20 \times 0.1\Omega} \\ I_{max} &= \frac{2.75 Volts}{2\Omega} = 1.375\ [Amperes] \end{aligned} \tag{30}$$

Assuming that the current flowing through the inductor of the buck converter circuit does not exceed the limit given in Equation (30), the expression for the current given the output voltage of the chip is:

$$\begin{aligned} I &= \left( \frac{V_{out} - \frac{V_{supply}}{2}}{20 \times 0.1\Omega} \right) \\ I &= \frac{V_{out} - \frac{V_{supply}}{2}}{2} \quad [Ampere] \end{aligned} \tag{31}$$

## 10.2 Analog to Digital Conversion

Since the controller will be implemented on a microcontroller board, the input to the controller must be represented in a digital format. The conversion between the analog voltage and the digital number representation is done by a device called an Analog to Digital Converter (ADC). Since the ADC runs on a microcontroller board, it must be considered that the conversion takes a non-trivial amount of time to complete. This means that the conversion cannot be considered to be done instantaneously, rather that the conversion takes a set number of clock cycles and thus operates at a predictable pace. On the Arduino, the clock frequency is 16 MHz, running off of a crystal

oscillator. When running in free-running mode, the ADC on board has a prescale factor on the clock of the board of $\frac{1}{64}$, meaning that the clock of the ADC runs at 250 kHz. The choice to operate the ADC in free-running mode forces the conversions to be performed at the highest possible rate. Additionally, each ADC conversion takes 13 clock cycles to complete, meaning that the maximum rate at which a conversion can be done is approximately 19.2 kHz.

The working principle of an ADC is fairly simple: to convert an analog voltage, which can in principle vary continuously, to a digital value, which is restricted to a fixed set of values. This process is known as quantization, because the value measured is considered to vary along discrete quanta. The amount of distinct values the ADC can differentiate between the maximum and minimum values of the ADC is defined by the number of quantization levels, $Q$. This also implies that, depending on the number of quantizations levels, a non-trivial error in the measurement is introduced by quantization. The expression for the smallest change in voltage that can be detected by the ADC, and the expression for the maximum amount of error introduced by quantization, is given as [10, 11]:

$$\Delta V = \frac{V_{max} - V_{min}}{Q - 1} \tag{32}$$

$$e_{max} = \pm \frac{1}{2\,(Q-1)} \times 100[\%] \tag{33}$$

In Equation (32) represents the minimum change in the input voltage , while Equation (35) represents the maximum error introduced by the ADC. In practice, it is conventional to use a 10-bit ADC. Indeed, the ADC on the Arduino board also converts the analog input to a 10-bit value, resulting in $2^{10}$ or 1024 quantization levels. The range of input voltage values on the Arduino ADC is 0V to 5V, meaning that the minimum detectable step size and maximum error are:

$$\Delta V = \frac{5V - 0V}{1024 - 1} = 4.888 \quad [mV] \tag{34}$$

$$e_{max} = \pm \frac{1}{2\,(1024-1)} \times 100[\%] = \pm 4.888 \times 10^{-2} \quad [\%] \tag{35}$$

The expression from the previous section can be rewritten as:

$$I = \frac{N_{ADC} - \frac{V_{supply} \times (Q-1)}{2 \times 5}}{2} \tag{36a}$$

$$I = \frac{N_{ADC} - 512}{2} \tag{36b}$$

The expression to calculate the current of the inductor in Equation (36a) is the most generalized form, where the number of quantization levels and the supply voltage can be changed independently. Expressing the relation between the current and the output value of the ADC in this way clearly shows how the analog reference offset, $\frac{V_{supply}}{2}$, is translated into a quantized value. On the other hand, Equation (36b) shows the reference offset as a constant value subtracted from the number generated by the ADC. The former expression may be desirable if the supply voltage of the INA240A1 chip and/or the number of quantization levels used in the ADC are subject to change. However, for the purpose of the experiments carried out as part of the research of this project, the supply voltage and number of quantization levels will be constant. Therefore, the latter

expression for the current will be used in the code which implements the controller. Depending on the optimization of the code compiler, the program which takes the code of the controller and translates it into Arduino instructions, the latter expression also has the benefit of executing faster.

Another key quantity to understand is the translation factor between the voltage read in by the ADC of the Arduino, and the number it produces, $N_{ADC}$:

$$N_{ADC} = V_{in} \times T \quad where\ T = \frac{1023}{5}$$
$$N_{ADC} = V_{in} \times \frac{1023}{5} \tag{37}$$

Although, the two expressions are likely to result in the same instructions being executed if the compiler efficiently optimizes the code.

## 10.3 Implementation Language

The easiest and most straightforward way to start writing code for Arduino projects is through the Arduino Software Integrated Development Environment (IDE), offered for free download on the Arduino website. The Arduino IDE offers a text editor, as well as built-in tools to allow the developer to verify and upload the code to the Arduino board. Code written for the Arduino in the IDE must be written in the programming language C. The choice of C as the language for Arduino project allows the programmer to write code in a high-level language, while maintaining fast and efficient computation. When programming in C a high degree of control is maintained over what tasks are performed, and how the computer performs these tasks.

As mentioned in the previous section, code for the Arduino must go through the process of being compiled. In this context, compilation refers to the process of translating the source file, written in C, into machine instructions for the ATMega328p, the microcontroller chip on the Arduino.Compiled code runs directly on the machine it was compiled for, in stark contrast to code written in the MATLAB language, which is interpreted by the MATLAB program. Another key way in which the C programming language and the MATLAB language differ is the type system used in each language.

### 10.3.1 Data Types

MATLAB is a dynamically typed language, which means that variable types are checked at execution time. In contrast, C is a statically typed language, meaning that variable types are checked when the program is being compiled. Simply put, data types are a way of telling the computer how to represent a value in memory. In C the data type of a variable must be declared at the same time as the variable itself is declared. It is also allowed to declare multiple variables of the same data type on the same line. Examples of valid variable declarations are provided in the following listing:

```
int a;
char b, c = 0;
float d = 1.5;
```

The fundamental data types in the C programming language can be split into two main categories: floating point and integer types. In C, the keyword '**float**' is used to represent a single-precision floating point number. Floating point numbers can, in most cases, be thought of as being synonymous with decimal point numbers from mathematics.

On the other hand, integer data types are used to store integer values, as one might expect given their name. C offers a variety of integer data types as part of the language, the most obvious of which being '**int**'. Other integer data types which will be used in the Arduino code for this project are '**char**', '**short**', and '**long**'. The C standard library defines each of the integer data types with respect to one another. The size of a '**char**' is always 1 byte, and is always shorter than a '**short**'. An '**int**' is at least as big as a '**short**', and a '**long**' is at least as big as an '**int**'. These relationships can be seen below:

```
// sizeof(char) == 1 byte
sizeof(char) < sizeof(short) <= sizeof(int) <= sizeof(long)
```

### 10.3.2 Arduino Sketches

In order to use the Arduino with an electronics project, code must be written and organized into a source code file known as an Arduino sketch. The canonical format of an Arduino sketch is as follows:

```
1 #include <EternalLibrary.h> //Link an external library
2
3 #define MACRO 21     //Define a macro
4 int global_var;      //Global variable  declaration
5
6 //Note: Global variables can be used everywhere in the document
7
8 void setup();        //Prepares the Arduino for the loop()
9 void loop();         //Called on a continuous loop by the Arduino
```

The above code fragment shows the most common aspects of all code source files written for the Arduino. The green keywords shown in the first and third lines of the above listing are known as preprocessor directives. The C preprocessor is the part of the compiler that transforms the code before it is compiled. The two common preprocessor directives found in Arduino sketches are '**#include**' and '**#define**'.

The '**#include**' statement is often followed by a filename either angle brackets, '<>', or in quotations. Angle brackets are used when the included file is a standard library, or header file, and may also be used in cases when code is being imported from external projects. Quotations are used when the header file to be included is in the same directory as the source file being compiled. In the event that the file is not found in the current directory, then the standard header file directories are checked.

The '**#define**' statement shown in the third line of the listing in this section tells the preprocessor to replace every instance of the word 'MACRO' with the number '21'. This substitution will be performed by the preprocessor throughout the entire source file.

The global variable initialized with the name 'global_var' in line 4 of the above listing is an example of a variable declared in the global scope. The necessary thing to understand about variable scope in C for the purpose of this project is that a variable declared in global scope can be used anywhere in the program.

The final part of the listing given in this section are the setup and loop functions. Since the Arduino does not have an operating system, the system must manually be prepared for the tasks it will perform in the loop function. The setup function is called only once at the start of the program

execution, while the loop function is called continuously once the Arduino has finished the setup. Normally, the loop function stores the main body of the program running in an Arduno project. Although, as will be explained in Section 10.6, in this project, the controller for the buck converter will not be implemented in the loop function. However, the loop function serves the purpose of allowing a new setpoint to be read in to the controller. This aspect of the Arduino code is detailed in Section 10.7.

## 10.4 Floating Point versus Integer Arithmetic

Since floating point numbers are the standard way in which to encode a decimal value on a computer, they are very useful when performing calculations. In principle, it would be desirable to use floating point numbers in a digital controller application. However, in many cases floating point arithmetic cannot be performed fast enough by the processor of the microcontroller.Part of the reason why floating point operations, or flops, are slow for the processor to compute stems from the fact that the number is stored in an encoded format. Another key reason for the relative slow pace of flops is the degree of accuracy offered by floats.

As an alternative to floating point arithmetic, digital controllers often utilize integer arithmetic. As an immediate benefit to using integers instead of floats, the logic hardware on board the microcontroller is built to operate directly on binary integers. This results in fewer machine instructions being performed and thus contributes to a faster execution time. When using integers to represent the controller, bit shifting can be used as an additional trick to improve the execution time of the control algorithm. Bit shifting is the name given to the operation of shifting all the bits of an integer to the left or the right by some number of spaces. In C, the right shift and left shift operators are represented by,'<<' and '>>', respectively. An example application of this technique is:

```
char test = 32; //Binary representation: 0010 0000

//Left shift is a way of dividing by multiples of 2
char test1 = test >> 1;  //test1 = 16 -- Binary: 0001 0000

//Right shift is another way of multiplying by multiples of 2
char test2 = test << 1; //test2 = 64 -- Binary: 0100 0000
```

## 10.5 Implementing the PI Control Algorithm

The following listing shows the code which is used to realize the digital controller on the Arduino. The code makes use of the '<<' and '>>' operators, which are the bit shift operators.

```
//the volatile keyword will turn off compiler optimizations
volatile long ki = 1, kp = 5;
volatile long i_out, error, integral = 0, ctrlSignal;

  i_out = (adcVal - 532)>>1; //532 was chosen because V_supply
                             //was often 5.3V
  error = setpoint - i_out;
  integral += ki * error;

  ctrlSignal = (5*((kp * error) + integral))>>10;
```

```
11
12    if(ctrlSignal > 255)
13        ctrlSignal = 255;
14    else if(ctrlSignal <= 0)
15        ctrlSignal = 0;
16
17    //Write the result to the proper register
18    //OCR2B = Output Compare Register 2B
19    OCR2B = (unsigned char) ctrlSignal;
```

Assuming 'adcVal' refers to the most recent value measured by the analog to digital converter, the inductor current of the buck converter is calculated on line 5 of the above listing. This line of code makes use of the formula for the inductor current found in Equation (36b), as well as the bit shifting trick explained in Section 10.4. Once the most recent value of the inductor current is calculated, the error of the system can be calculated by taking the difference between the two values (shown in line 7). At which point, the contribution of the integral gain of the controller can be calculated. In discrete systems, the integral term of a PI controller is implemented by keeping track of a running sum, which is identified by the name 'integral' on line 8 of the listing. This is valid because the running sum in a discrete time system is synonymous with an integral starting from $t = 0$ in a continuous time system.

Line 10 of the above listing implements a few steps of the calculation in one line of code. First, the contribution of the proportional component of the PI controller is calculated with the expression, $kp * error$. This value is then immediately added on to the value stored in the variable 'integral'. At this point, the discrete control signal is calculated, although the transformation applied by the ADC from Equation (37) must not be forgotten. In order to undo the transformation of the ADC, the computer multiplies the discrete control signal value by 5 and shifts that value to the right 8 bits, effectively dividing by 1024. In order for the PWM to work, the value of 'ctrlSignal' must be constrained to a value between 0 and 255. Once it is certain that the control signal is a one byte value, it can be sent to the Output Compare Register 2B (OCR2B).

### 10.6 Interrupt Service Routines

An event which causes the execution of the main program to be halted is called an interrupt. In the event of an interrupt, the CPU immediately diverts attention to the interrupt handler function, also known as an Interrupt Service Routine (ISR). The contents of the ISR determines which actions get performed when the corresponding interrupt is called. Interrupt routines can be used to mandate a higher level of priority to the execution of critical tasks in embedded systems. In the case of the implementation of a digital controller on the Arduino board, the control algorithm is implemented in an interrupt routine. The code listing shown in Section 10.5 forms the body of the ISR called by the microcontroller. The first few lines of the ISR used in this project are as follows:

```
1 ISR(ADC_vect){
2   digitalWriteFast(DEBUG_PIN, true); // marker pin high
3
4   adcVal=ADCL; // store lower byte ADC
5   adcVal+=ADCH<<8; // store higher bytes ADC
6
7 //More code...
```

The first line of the above listing defines the ISR as being triggered by the completion of an analog to digital conversion. This means that each time an analog to digital conversion is done, the ISR is called, the debug pin is set HIGH, and the value measured by the ADC is stored in the integer 'adcVal'. Since the ADC is already running at the fastest possible rate, triggering the controller calculation on the completion on an ADC conversion ensures that the control signal is updated as fast as possible. The argument of the ISR, as shown in line 1 of the listing, is ADC_vect. This means that the ISR will be called every time an analog to digital conversion of the voltage on pin A0 of the Arduino. This means that the update rate of the PWM duty cycle value is the rate at which ADC conversions occur, which is 19.2 kHz (Section 10.2).

## 10.7 Dynamically Assigning the Setpoint

In order to obtain results that demonstrate that the controller remains stable when the setpoint is changed, the code written for the controller must support user Input/Output (I/O) operations. Optimally, the setpoint could be passed to the Arduino as an integer which represents the setpoint in terms of milliAmperes. The code necessary to read the setpoint value from the user will be implemented in the '`loop()`' function. This is due to the fact that, for the purpose of the digital controller implementation in this project, it is unnecessary to react upon a new setpoint as fast as possible.

On the Arduino, the main interface between the user and the program being executed is through the 'Serial Monitor'. Interactions with the Arduino are mediated through a family of 'Serial' functions. The particular function of interest is the '`Serial.readBytes`' function, which provides the ability to read a set number of characters from the Serial input into memory. In order for this function to work, a memory buffer must be available for the function to store the characters from input into.

Once the characters from the serial input have been read into memory, they are still encoded in the ASCII format. Therefore, the characters in the memory buffer must be converted into the integer the user inteded. This can be handeled by the standard library function, '`atoi`', which stands for "alphabetic to integer". Then, once the integer result is obtained from '`atoi`', it must be converted into the same units as the ADC value, such that the two can be compared. Otherwise, the error signal produced by taking the difference between the setpoint and the ADC value would be incorrect. To ensure that the setpoint is given the right value such that the controller regulates properly, the integer converted using '`atoi`' must be multiplied by $\frac{1023}{5\times1000}$. The code implementation of reading values from input is presenting in the following listing:

```
//Only execute the following code if new input is available
if(Serial.available() > 0)
{
  char buffer[5] = {0}; //Statically allocate 5 bytes

  //Read those bytes from input
  setpoint = Serial.readBytes(buffer,5);

  setpoint = atoi(buffer)/5;            //Divide by 5
  setpoint += 2 * (setpoint / 100);     //Account for inaccuracy

  //A trick to safely discard any remaining characters from input
  while(Serial.available())
    Serial.read();
```

```
15 }
```

On line 9 of the above code the character string read into the buffer is converted into an integer and subsequently divided by 5. This operation serves the purpose of transforming the setpoint integer value into a value comparable with the ADC output. Since it is mathematically imprecise to neglect the rest of the transformation namely, multiplying by 1023 and dividing by 1000, the inaccuracy is compensated for in the next line of code. To compensate for inaccuarcy, the value of two is added for each multiple of 100 that 'setpoint' surpasses (i.e if 'setpoint' were 800, the value 8 will be added and 'setpoint' becomes 808). While processing the value of 'setpoint' in this way is faster than executing the proper mathematical transformation, $\frac{1023}{5\times1000}$, no tests were performed to determine whether this difference in speed is relevant. Alternatively, lines 9 and 10 in the above listing can be replaced the following line:

```
1   setpoint = (atoi(buffer)<<10) / 5000;
```

# 11 Results

In most cases, the results of the simulations and experiments were taken using the same value of resistance connected across the output of the buck converter. This was done in order to standardize the results as much as possible, such that the results from the simulation and experimental domains can be compared one-to-one. In the event that the resistor used in a particular simulation or experiment differs from 10Ω, this will be stated in the explanation of the result.

## 11.1 MATLAB Simulations

The diagram used to obtain the Simulink simulations can be found in Appendix D.1.

### 11.1.1 Step Response

Both Figures 14 and 15 show the step response behavior of the system according to the Simulink simulation. In both simulations the setpoint chosen was 1A of current. The rise time of the system according to the Simulink simulation is about 0.25 ms. While Figure 14 indicates that the system has an initial undershoot, the Figure 15 shows that the average value of the output current signal stays between 105% and 95% of the setpoint at startup.

### 11.1.2 Step Input Disturbance

In order to test the robustness of the controller against a change in the step input, a simulation was devised where the setpoint would be set to a few different values over the course of 50 ms. The initial setpoint is 250 mA, which will last for 5 ms. The setpoint will then follow this sequence: 500mA for 10ms, 1A for 15ms, 250mA for 10ms, and finally 500mA for 10ms. This sequence is shown in Figures 16 and 17.

The result of conducting the simulation with the standard value of load resistor 10Ω is shown in Figures 16 and 17. Both figures show that the controller is robust against the setpoint changing at any point during its operation. Figure 17 is the result of running the same simulation with all setpoint values divided in half. Although it may be difficult to tell, the juxtaposing Figures 16 and 17 is meant to illustrate the point that the overshoot of the system is dependent on the magnitude of the setpoint. This phenomenon will be explained in the Discussion. In order to make this point clear, another simulation was conducted with a load resistor value of 2Ω.

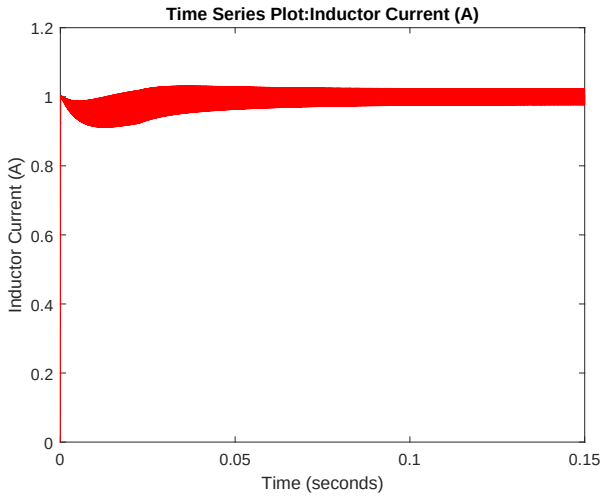



Figure 14: Step response of the inductor current

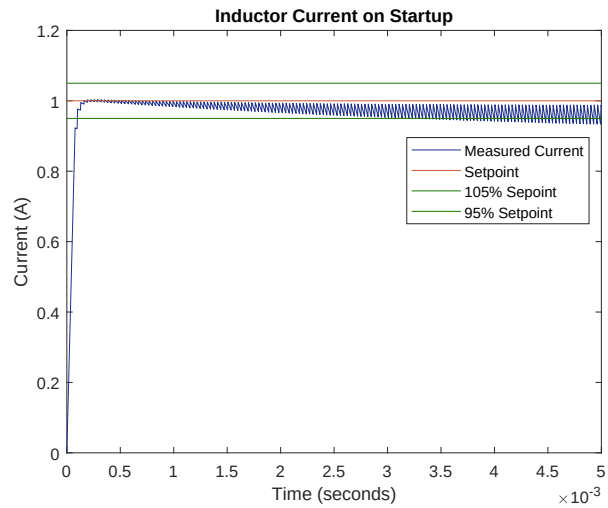



Figure 15: Behavior of the system in the first few milliseconds

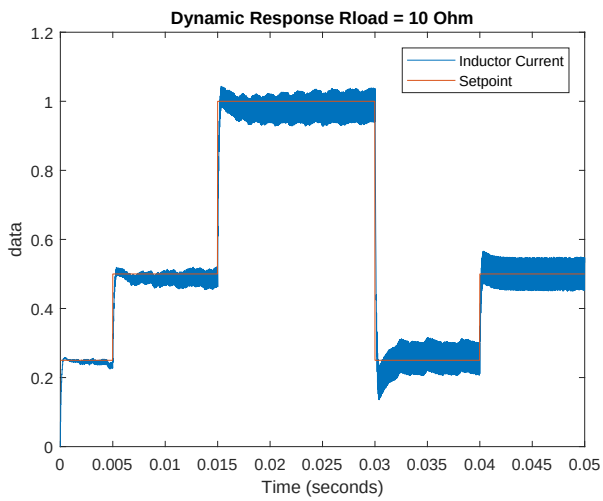



Figure 16: $R_{load} = 10\Omega$, max current 1A

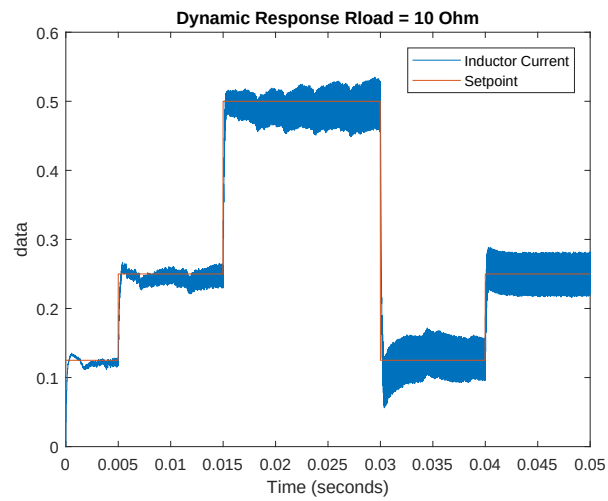



Figure 17: $R_{load} = 10\Omega$, max current 0.5A

Figure 18 is a recreation of the test shown in Figure 16, using the same setpoint values. In contrast, Figure 19 shows the results when all setpoint values are multiplied by 5. In this test, the maximum setpoint given to the system is 5A, which is theoretically feasible given an input voltage of 12V. This test was done only to show the property that the overshoot of the controller is dependent on the magnitude of the setpoint. This test cannot be done on the experimental setup because the maximum current the sensor can read is 1.375A, from Equation (30).

## 11.2 LTspice Simulations

The LTspice model used to obtain the circuit simulation results can be found in Appendix D.2.

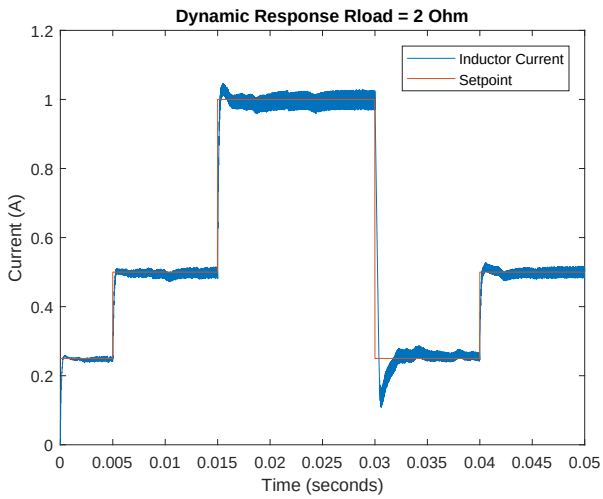



Figure 18: $R_{load} = 2\Omega$, max current 1A

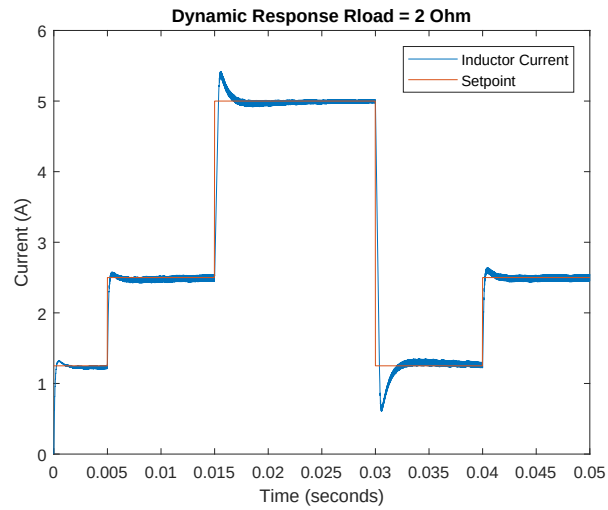



Figure 19: $R_{load} = 2\Omega$, max current 5A

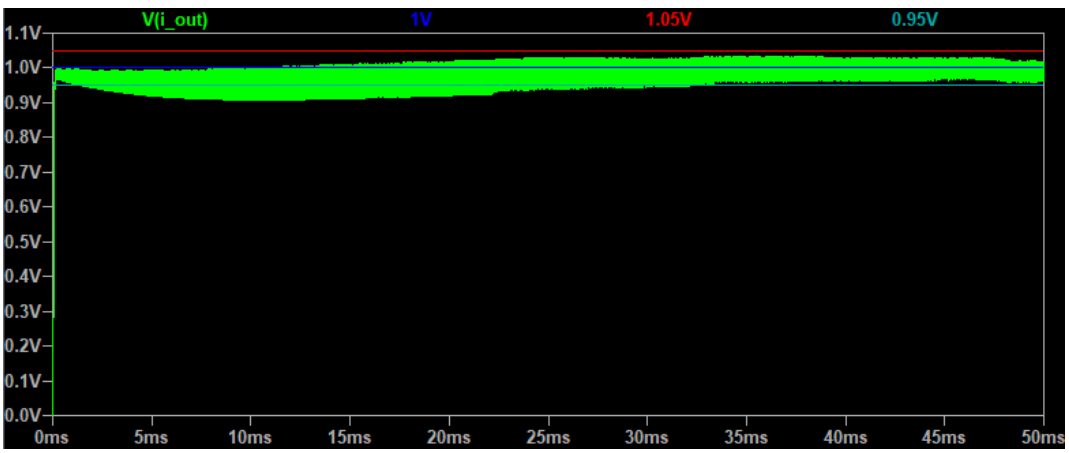



Figure 20: Step response when simulated in LTspice with $R_{load} = 10\Omega$

### 11.2.1 Step Response

Figure 20 shows the result of the LTspice simulation testing the system under the same conditions in Section 11.1.1. Therefore, Figure 14 and Figure 20 are meant to show the same behavior. In order to get a clearer view of the behavoir of the system during the first few milliseconds of operation, the test from Figure 15 was recreated in LTspice. The results of this test can be seen in Figure 21.

Similarly to Figure 15, Figure 21 shows that the system has no overshoot when the setpoint is set to 1A with an operating load of 10Ω. The undershoot of the system can be shown to go as low as 95% of the initial setpoint. The rise time of the system can be estimated to be 0.2 ms.

### 11.2.2 Step Input Disturbance

In order to show that the circuit is likely to be stable when the step input is disturbed, this condition must be tested in LTspice. In an effort to standardize the results from Simulink to LTspice, the same sequence of setpoint values and time intervals from the Simulink tests were used. The results from the LTspice simulation show that the circuit is indeed likely to be stable as

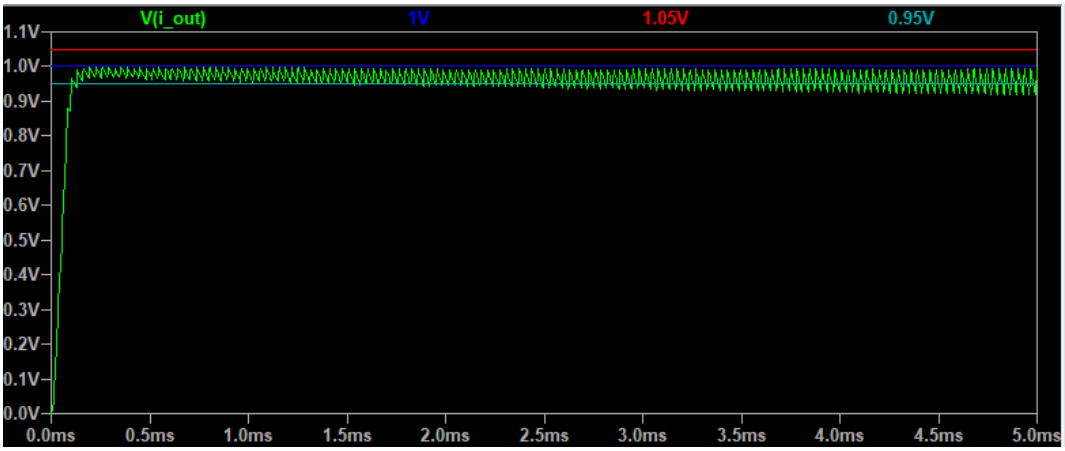



Figure 21: First 5 milliseconds of step response when simulated in LTspice with $R_{load} = 10\Omega$

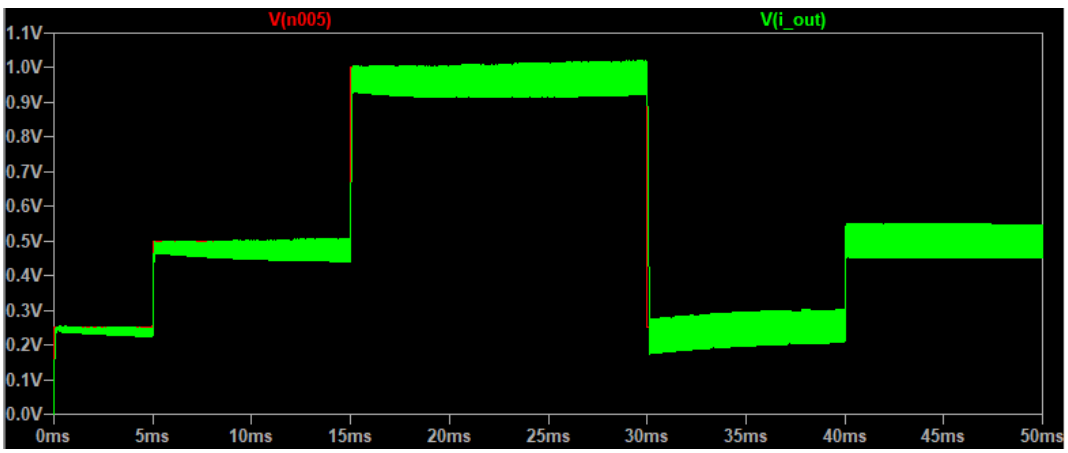



Figure 22: Dynamic response of the system when the setpoint changes level $R_{load} = 10\Omega$

designed. Each time the setpoint was changed there is little discernible over- or undershoot to the setpoint value, as shown in Figure 22.

### 11.2.3 Load Disturbance Rejection

In principle, the load connected to a power regulator, such as a buck converter, is subject to change as electrical devices connected to the power network are switched on and off. Therefore, testing the scenario where the load resistor changes dramatically and instantaneously is considered in this section. The load resistor connected to the buck converter was changed instantaneously from $10\Omega$ to $1\Omega$ once the system reached a steady state.

In Figure 23, the current of the inductor, $I_{out}$, is shown when the load resistor changes from $10\Omega$ to $1\Omega$. The output current begins to rise at the moment the resistance dropped, reaching a maximum value of 1.1A. However, within 25 ms the system returns to its steady state output current.

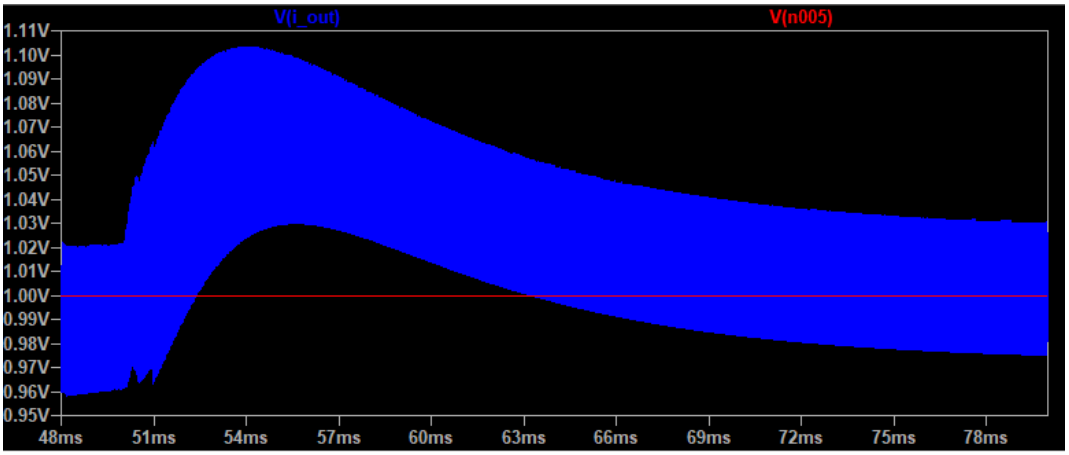



Figure 23: Load disturbance rejection of the system, $R_{load}$ is being switched from 10Ω to 1Ω at the 50 ms mark

### 11.2.4 Dynamic Setpoint and Load Disturbance

While the previous test showed the circuit in a more realistic scenario where the load resistance is subject to change, the parameters of that test leave out one potential case: when the load and the setpoint change at the same time. In order to examine whether the controller will remain stable under the least ideal conditions the resistance values were set to transition at the moment when the setpoint also changed. The results of this test are shown if Figure 24.

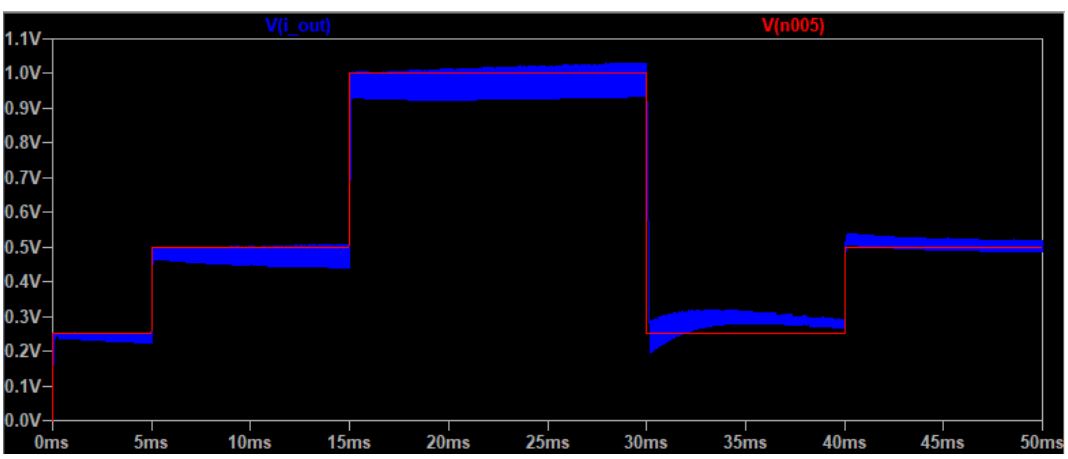



Figure 24: Dynamic input reference and load disturbance, at $t = 15ms$ $R_{load}$ goes from 10Ω to 5Ω. At $t = 30ms$, $R_{load}$ goes from 5Ω to 1Ω

As it can be clearly seen in Figure 24 the system remains stable even when the setpoint and load resistor change at the same time. This final simulation test can be used to infer that the real system, implemented in the lab should also be very robust.

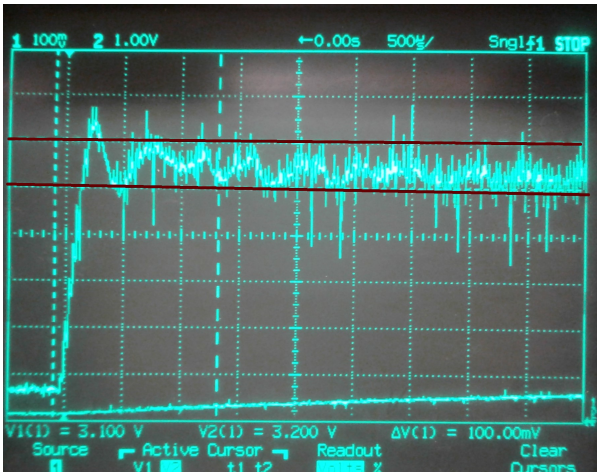



Figure 25: Startup response of the closed loop system experimental setup, $R_{load} = 10\Omega$, setpoint is 250 mA

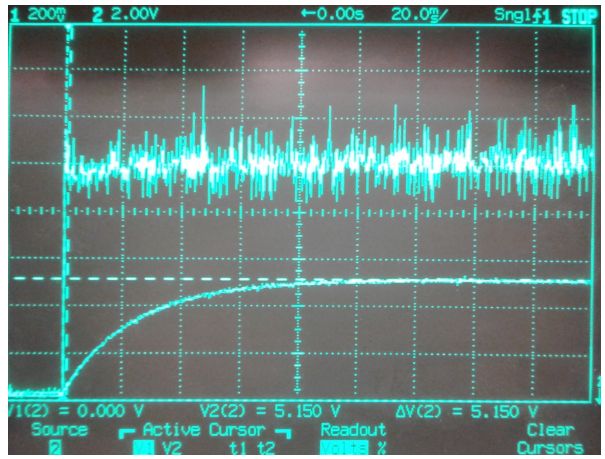



Figure 26: Step response of the closed loop system, setpoint is 500 mA

## 11.3 Experimental Results

### 11.3.1 Constant Load and Setpoint

Figure 25 and 26 show the step response of the real buck converter system implemented in the lab. From Figure 25, the timescale of the horizontal axis is $500\mu s/div$. The setpoint in this test is set to 250mA. The voltage shown on the oscilloscope is the output voltage of the INA240A1 current sense amplifier. Given a supply voltage of 5.3V, the expected output voltage of the current sensor can be calculated by rearranging Equation (31):

$$V_{out} = 2 \cdot I + \frac{V_{supply}}{2}$$
$$V_{out} = 0.5V + \frac{5.3V}{2} = 3.15V$$

The voltage values corresponding to 95% and 105 % of the setpoint value are calculated in the same way, yielding 3.125V and 3.175V, respectively. The horizontal cursors are located roughly at those voltages to aid in determining the rise time of the experimental setup. These cursors are highlighted by the brown lines drawn over the image. Based on Figure 25, the rise time of the real system can be estimated to be 0.2 ms. The results from Figure 25 also show that the system overshoots the setpoint value in its initial rise. Considering the peak of the voltage shown in the oscilloscope output is approximately 3.275V, the peak current throught the output of the buck converter is 312.5mA. This means that the initial overshoot is 125% of the setpoint value in this test. The settling time, the time after which the system stays within ±5% of the setpoint, and thus between the brown lines in Figure 25, occurs after $500\mu s$. Figure 26 shows the experimental setup reaching a steady state given a setpoint of 500mA.

### 11.3.2 Constant Load, Varying Setpoint

The test results in Figure 27 were designed to mimic the results in Figures 16 and 24. Since the setpoint in the experimental setup is read in according to user input, it was impossible to

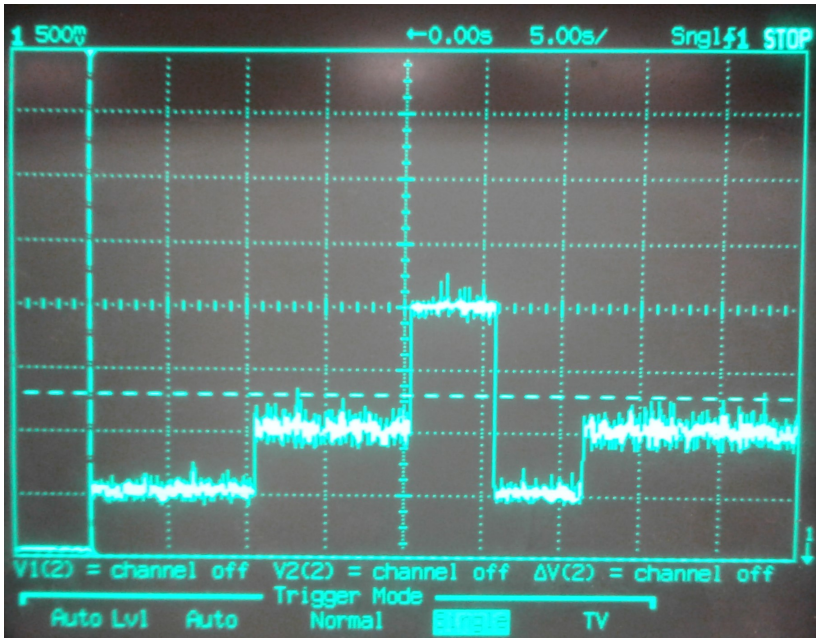



Figure 27: Dynamic response of the system with varying setpoint

recreate the experiment on the same timescale as given in the simulations. However, the same current setpoint values were chosen for testing. Additionally, the same value of the load resistor, 10Ω, was used in this experiment. In Figure 27, each vertical division represents 500mV, which due to Equation (31), translates to a difference of 250mA per division. Therefore it can be seen on the oscilloscope output that the sequence of setpoint values tested are, in order: 250mA, 500mA, 1A, 250mA, and 500mA. From the output on the oscilloscope it is clear that the system remains stable under this test condition.

### 11.3.3 Load Disturbance

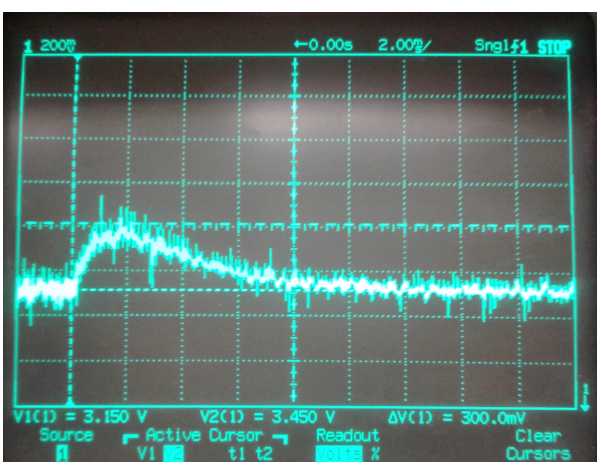



Figure 28: Load disturbance rejection. $R_{load}$ goes from 10Ω to 1Ω

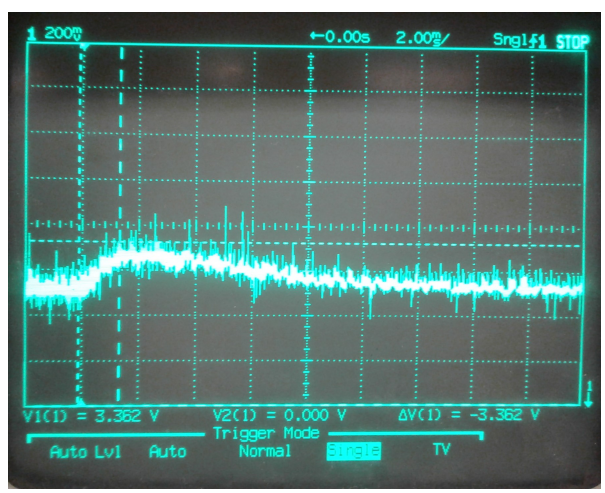



Figure 29: Load disturbance rejection. $R_{load}$ goes from 10Ω to 2Ω

The final battery of tests conducted on the experimental setup have to do with testing the controller for its ability to reject load disturbances. In the oscilloscope output photos given in Figures 28 and 29, the timescale on the horizontal axis is 2ms per division. In Figure 28, the controller returns the circuit to its steady state output current 8ms after the initial disturbance. While in Figure 29, the controller returns the circuit to the desired steady state between 8ms to 10ms after the load disturbance.

# 12 Discussion

The PI controller has been shown to meet the phase and gain margin requirements in Section 9. The remaining requirements which needed to be tested in simulation and experimentally were the settling time of the system, the maximum overshoot, and the steady state error. The simulated step response of the buck converter system shows that, under the conditions tested, the settling time of the system is 0.25ms and 0.2ms in Simulink and LTspice, respectively. Because the simulations indicate that the average value of the current stays within 105% and 95% of the setpoint value after the initial rise, the settling time and rise time of the simulations can be considered to be the same. Both of these values are within the 0.5s settling time requirement. In the lab experiments, the buck converter is shown to have a settling time of 0.5ms. Both the controllers implemented in simulation and the controller implemented in the lab meet the settling time requirement.

The steady state requirement of the PI controller, namely that the controller should regulate the system with zero steady state error has been shown to hold in simulation as well as in experiments conducted. Based on the simulations conducted in both Simulink and LTspice, the maximum overshoot under practical conditions also stays under the required 105% maximum overshoot specification. However, the controller built in the lab was shown to produce 125% overshoot during the initial rise of the system. Therefore, the PI controller implemented in the simulations pass all the design requirements set out in Section 9.

From Figures 16, 22, and 27 the PI controller has been shown to maintain stability when the input reference is disturbed. Furthermore, Figures 23, 28, and 29 show that the system remains stable when the output load connected to the buck converter is altered.

The experimental results can be said to reflect the simulations insofar as the rise time of the simulations and the experimental setup are all roughly the same. The step response of the experimental setup was shown to exhibit behavior differing from the behavior expected from the simulations. This is due to the fact that the simulations were implemented assuming that the controller could be modeled as an analog controller. The update rate of the controller built in the lab is driven by the maximum rate at which the ADC can function, which is approximately 19kHz. Therefore, the assumption that the controller can be modeled as an analog controller is invalid. In order to fully optimize the behavior of the controller built in the lab, the design of a discrete controller should be considered.

# 13 Recommendations

## 13.1 Code Refactoring

If the code used in this Arduino project is to be used again, the code should be refactored to make the use of fixed point integers more transparent to the programmer. As it can be seen in Section 10.5, the code written for this project does not explicitly convert the $k_i$ and $k_p$ values into a fixed point format. Therefore, the controller values that the user declares are treated as if they

were already converted. This means that the actual controller coefficients being used are obfuscated from the programmer. This issue can be fixed by adding a macro function 'toFixed' which takes in a decimal value, stored as a '**float**', and the number of fraction bits, and returns the fixed point integer representation of that decimal value. The solution can be found in the following listing:

```
//The following code translates a float into a fixed point number
#define toFixed(decimal,FRACTION_BITS) ((int)(decimal * \
                                          (1 << FRACTION_BITS)))

double kp = 2.3, ki = 0.11;

volatile long kp_int = toFixed(kp,10), \
              ki_int = toFixed(ki,10);
```

In order to verify the accuracy of this implementation of converting floating point values into fixed point, a small program was written. For an arbirary number of test cases, the program converts a floating point value into the fixed point representation (32,10). Then the resulting fixed point integer is converted back into a floating point number. To compare the original and the result of converting the number twice, the program calculates the difference between the two numbers. The output of the program is a report stating the average difference between the original number and the twice converted number. The source code of this program, as well as an example usage of the program, can be found in Appendix F.

### 13.1.1 Standardizing ADC and Setpoint Readings

Additional considerations must also be made for converting the value read in by the ADC and the setpoint to the proper fixed point representation. Due to the conversion factor being $T = \frac{1023}{5}$, as shown in Equation (37), in order to convert the ADC value into a fixed point representation with 10 fractional bits, one must simply multiply the ADC value by 5. The fixed point precision, given by the number of fractional bits, can then be changed by simply shifting the result either to the right or left. To convert the setpoint, read from the user in milliAmps, to the proper fixed point value, one must first multiply by the fixed point offset, $2^n$ where $n$ is the number of fractional bits, and divide the result by 1000. Example code implementing these conversions is given in the following listing, assuming the current adcVal and setpoint have already been read:

```
//Converting the value of from the ADC

//If the desired fixed point value has 10 fractional bits,
//simply multiply the adcVal by 5
adcVal *= 5;

//Convert the input read in from the user:
setpoint = (setpoint * (1<<FRACTIONAL_BITS)) / 1000;
```

## 14 Conclusion

The PI controller was tuned by setting the proportional gain to zero initially, raising the integral gain until the system becomes unstable. Then the proportional gain was increased incrementally

until the desired step response was achieved. The design parameters were checked to ensure that the resulting controller met specifications. Chief among those concerns is the stability requirements for phase and gain margin, the controller satisfied the design specifications. Then simulation models in Simulink and LTspice were made for the purpose of investigating the controller's behavior. In Simulink the State Space model can be implemented directly into the simulation, whereas LTspice is used to obtain a circuit simulation of the system. The Simulink simulation is useful when attempting to verify the mathematical model. The use of LTspice is necessary to verify the selection of the electrical components because LTspice will provide a more accurate simulation of the circuit. The controller was shown to remain stable in simulations, even when the input reference was disturbed. The system was also shown to be robust against disturbances in the resistive load. While the results from Simulink were comparable with the results from LTspice, the experimental results differed from the simulations. This is because the controller implemented in both the Simulink and LTspice simulations was an analog controller. The controller implemented in the lab was a digital controller, and the sample rate and computation rate of the Arduino microcontroller is insufficient for it to behave like an analog controller at the switching frequency chosen.

## References


[1] R. Singh and K. Shenai, "Dc microgrids and the virtues of local electricity," *IEEE Spectrum*, vol. 6, 2014.

[2] B. Scott, S. Gregory, C. Spyridon, *et al.*, "Dc microgrids scoping study—estimate of technical and economic benefits," *LosAlamos National Laboratory, Brookheaven*, 2015.

[3] M. Cucuzzella, S. Rosti, A. Cavallo, and A. Ferrara, "Decentralized sliding mode voltage control in dc microgrids," in *American Control Conference (ACC), 2017*, pp. 3445–3450, IEEE, 2017.

[4] A. R. Hevner, "A three cycle view of design science research," *Scandinavian journal of information systems*, vol. 19, no. 2, p. 4, 2007.

[5] Fairchild Semiconductor International, Inc., *NPN Epitaxial Silicon Transistor*, Aug. 2002.

[6] S. P. Singh, "Output ripple voltage for buck switching regulator," *Application Note (SLVA630A), Texas Instruments, October*, 2014.

[7] Texas Instruments Inc., *INA240 High- and Low-Side, Bidirectional, Zero-Drift, Current-Sense Amplifier With Enhanced PWM Rejection*, Feb. 2018.

[8] Texas Instruments Inc., *INA240 High- and Low-Side, Bidirectional, Zero-Drift, Current-Sense Amplifier With Enhanced PWM Rejection*, Feb. 2018. p. 14.

[9] Texas Instruments Inc., *INA240 High- and Low-Side, Bidirectional, Zero-Drift, Current-Sense Amplifier With Enhanced PWM Rejection*, Feb. 2018. p. 4.

[10] J. P. Bentley, *Principles of measurement systems.* Pearson Education India, 1988. p. 249.

[11] J. P. Bentley, *Principles of measurement systems.* Pearson Education India, 1988. p. 251.

# Appendices

## A Average Model Derivation

$$A_{on} = \begin{bmatrix} -\frac{R_{sensor}}{L} & -\frac{1}{L} \\ \frac{1}{C} & -\frac{1}{RC} \end{bmatrix} \quad , \quad C_{on} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

Given $A_{on} = A_{off}$ and $C_{on} = C_{off}$:

$$\begin{aligned} A &= A_{on}d + A_{off}\left(1-d\right) \\ &= A_{on}d + A_{on} - A_{on}d \\ &= A_{on} \end{aligned} \qquad \begin{aligned} C &= C_{on}d + C_{off}\left(1-d\right) \\ &= C_{on}d + C_{on} - C_{on}d \\ &= C_{on} = \begin{bmatrix} 1 & 0 \end{bmatrix} \end{aligned}$$

$$B_{on} = \begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix} \quad , \quad B_{off} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

The average of B calculated in the same way as the average of A and C:

$$\begin{aligned} \bar{B} &= \begin{bmatrix} \frac{V_s}{L} \\ 0 \end{bmatrix} d + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \left(1-d\right) \\ &= \begin{bmatrix} \frac{d}{L} \\ 0 \end{bmatrix} \end{aligned}$$

# B Voltage to Current Transfer Function

$$
\begin{aligned}
H(s) &= C(sI - A)^{-1}B \\
&= \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} s + \frac{R_{sensor}}{L} & \frac{1}{L} \\ -\frac{1}{C} & s + \frac{1}{R_{load}C} \end{bmatrix}^{-1} \begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} s + \frac{1}{R_{load}C} & -\frac{1}{L} \\ \frac{1}{C} & s + \frac{R_{sensor}}{L} \end{bmatrix} \begin{bmatrix} \frac{V_s}{L} \\ 0 \end{bmatrix} \times \frac{1}{\left(s + \frac{R_{sensor}}{L}\right)\left(s + \frac{1}{R_{load}C}\right) + \frac{1}{LC}} \\
&= \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \frac{V_s}{L}\left(s + \frac{1}{R_{load}C}\right) \\ \frac{V_s}{LC} \end{bmatrix} \times \frac{1}{s^2 + s\left(\frac{1}{R_{load}C} + \frac{R_{sensor}}{L}\right) + \frac{1}{LC}} \\
H(s) &= \frac{V_s}{L} \frac{s + \frac{1}{R_{load}C}}{s^2 + s\left(\frac{1}{R_{load}C} + \frac{R_{sensor}}{L}\right) + \frac{1}{LC}}
\end{aligned}
$$

## B.1 Phase Shift of the RLC System

Argument of $H(s)$, considering $s = j\omega$:

$$
\begin{aligned}
H(j\omega) &= \frac{V_s}{L} \frac{j\omega + \frac{1}{R_{load}C}}{(j\omega)^2 + j\omega\left(\frac{1}{R_{load}C} + \frac{R_{sensor}}{L}\right) + \frac{1}{LC}} \\
\arg\left(H\left(j\omega\right)\right) &= \arg\left(\frac{V_s}{L} \frac{j\omega + \frac{1}{R_{load}C}}{(j\omega)^2 + j\omega\left(\frac{1}{R_{load}C} + \frac{R_{sensor}}{L}\right) + \frac{1}{LC}}\right) \\
&= \arg\left(j\omega + \frac{1}{R_{load}C}\right) - \arg\left((j\omega)^2 + j\omega\left(\frac{1}{R_{load}C} + \frac{R_{sensor}}{L}\right) + \frac{1}{LC}\right) \\
\arg\left(H\left(j\omega\right)\right) &= \arctan\left(\frac{\omega}{R_{load}C}\right) - \arctan\left(\frac{\omega\left(\frac{1}{R_{load}C} + \frac{R_{sensor}}{L}\right)}{\frac{1}{LC} - \omega^2}\right)
\end{aligned}
$$

## B.2 Open loop transfer function derivation

$$
\begin{aligned}
L(s) &= C(s)H(s) \\
&= \left(k_p + \frac{k_i}{s}\right) \cdot \frac{V_s}{L} \frac{s + \frac{1}{R_{load}C}}{s^2 + s\left(\frac{1}{R_{load}C} + \frac{R_{sensor}}{L}\right) + \frac{1}{LC}} \\
&= \frac{V_s}{L} \cdot \frac{(k_p s + k_i)\left(s + \frac{1}{R_{load}C}\right)}{s^3 + s^2\left(\frac{1}{R_{load}C} + \frac{R_{sensor}}{L}\right) + s\frac{1}{LC}} \\
&= \frac{V_s}{L} \cdot \frac{s^2 k_p + s\left(k_i + \frac{k_p}{R_{load}C}\right) + \frac{k_i}{R_{load}C}}{s^3 + s^2\left(\frac{1}{R_{load}C} + \frac{R_{sensor}}{L}\right) + s\frac{1}{LC}}
\end{aligned}
$$

# C Resonance of an RLC

Given the RLC network in Figure 10, the transfer function can be calculated as follows:

$$H(j\omega) = \frac{Z_{RC}}{Z_{RC} + Z_L}\ , \quad where\ Z_{RC} = Z_R//Z_C = \frac{R}{1 + j\omega RC}$$

$$\begin{aligned}
&= \frac{\frac{R}{1+j\omega RC}}{\frac{R}{1+j\omega RC} + j\omega L} \times \frac{1 + j\omega RC}{1 + j\omega RC} \\
&= \frac{R}{R + j\omega L\left(1 + j\omega RC\right)} \\
&= \frac{1}{1 + j\omega \frac{L}{R} - \omega^2 LC}
\end{aligned}$$

## C.1 Magnitude and Resonance Derivation

$$|H(j\omega)| = \frac{1}{\sqrt{\left(1 - \omega^2 LC\right)^2 + \left(\omega \frac{L}{R}\right)^2}}$$

The resonance frequency is found at the maximum of the above function. As a shortcut, the frequency at which minimum value of the expression under the square root in the denominator is found will be the same frequency at which the maximum transfer is found.

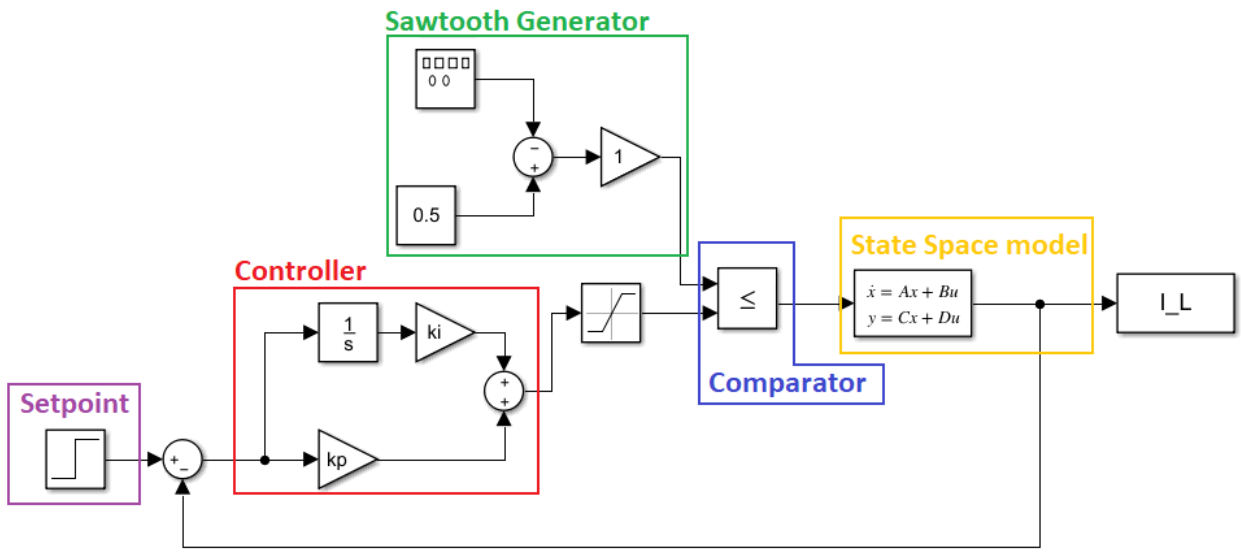
$$\begin{aligned}
\frac{d}{d\omega}\left(\left(1 - \omega^2 LC\right)^2 + \omega^2\left(\frac{L}{R}\right)^2\right) &= 0 \\
2\left(1 - \omega^2 LC\right)\left(2\omega LC\right) + 2\omega\left(\frac{L}{R}\right)^2 &= 0 \\
4\omega LC - 4\omega^3 L^2 C^2 + 2\omega\frac{L^2}{R^2} &= 0 \\
2\omega\left(2LC - 2\omega^2 L^2 C^2 + \frac{L^2}{R^2}\right) &= 0
\end{aligned}$$

The last line implies the trivial solution $\omega = 0$ as the resonance frequency, however a non-trivial solution can still be calculated by finding the zeros of the polynomial expression.

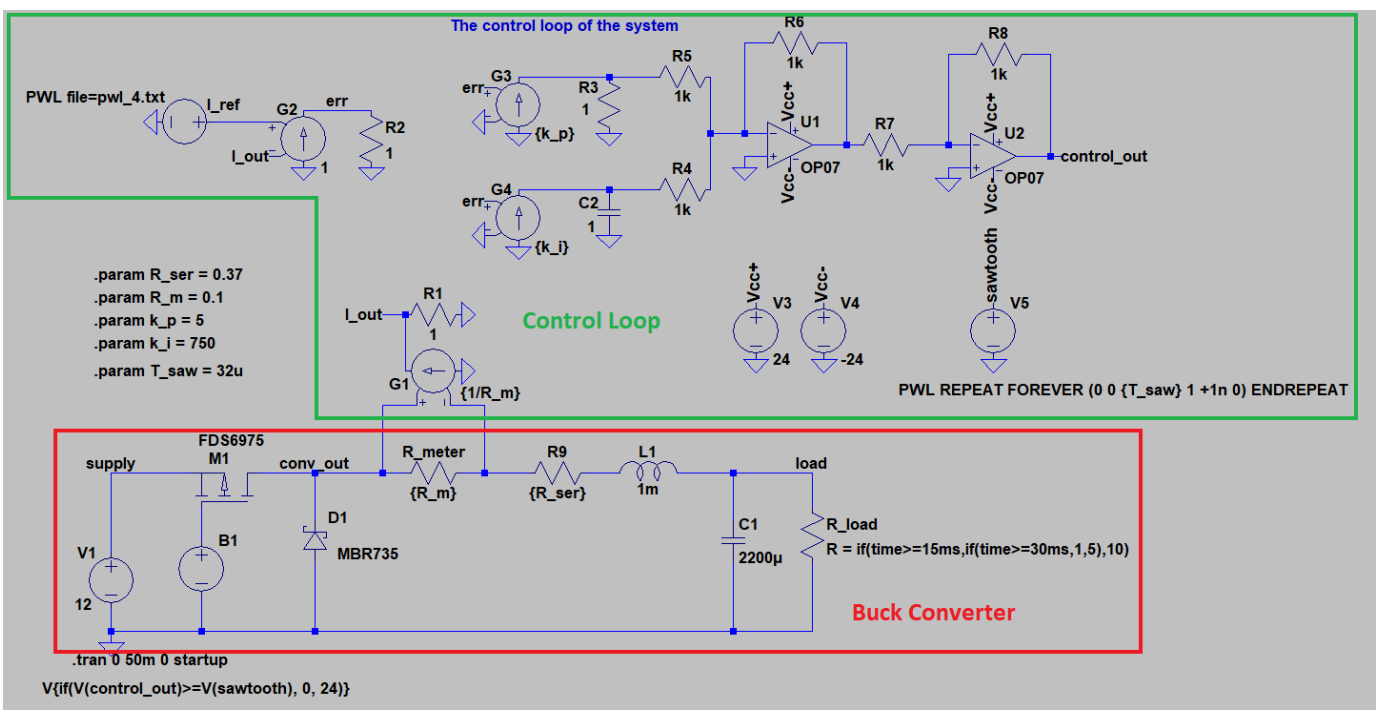
$$\begin{aligned}
0 &= 2LC - 2\omega^2 L^2 C^2 + \frac{L^2}{R^2} \\
2\omega^2 L^2 C^2 &= 2LC + \frac{L^2}{R^2} \\
\omega^2 &= \frac{1}{LC} + \frac{1}{2\left(RC\right)^2} \\
\omega_{res} &= \sqrt{\frac{1}{LC} + \frac{1}{2\left(RC\right)^2}}
\end{aligned}$$

# D Simulation Diagrams

## D.1 MATLAB Simulink Diagram




## D.2 LTspice Diagram



## E All Arduino Code

```
//#include "Arduino.h"
#include <digitalWriteFast.h>
  //digitalWriteFast.h  -- library for high performance digital
  //                       reads and writes by j.r.raines
  //see http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1267553811/0
  //and http://code.google.com/p/digitalwritefast/


#define ADC_CH0 0   //MUX3:0: Analog Channel Selection Bits
#define PWM_PIN 3   //Timer2 generated PWM output pin
#define DEBUG_PIN 4 // debug output pin.
#define LED_PIN 13  // Arduino LED output pin.


//The setup function is called once at startup of the sketch
void setup()
{
    // Add your initialization code here
    pinMode(DEBUG_PIN, OUTPUT);
    pinMode(PWM_PIN, OUTPUT);
    pinMode(LED_PIN, OUTPUT);

    // when using Serial.print() in the function loop()
    Serial.begin(115200);

    // initialize ADC for continuous sampling mode
    DIDR0 = 0x3F; // digital inputs disabled for ADC0D to ADC5D
    bitSet(ADMUX, REFS0); // Select Vcc=5V as the ADC reference
    bitClear(ADMUX, REFS1);
    ADMUX = (ADMUX & B11110000)|ADC_CH0; //selects the ADC channel
    bitSet(ADCSRA, ADEN);  // AD-converter enabled
    bitSet(ADCSRA, ADATE); // auto-trigger enabled
    bitSet(ADCSRA, ADIE);  // ADC interrupt enabled

    // ADC prescaler division factor = 64 (free running mode)
    // ADC clock runs at 16MHz/64=250kHz
    // One conversion takes 13 clock periods.
    // fsampl = 250.000/13 = 19230.769Hz
    bitClear(ADCSRA, ADPS0);
    bitSet(ADCSRA, ADPS1);
    bitSet(ADCSRA, ADPS2);
    // set trigger source to 'free running'
    bitClear(ADCSRB, ADTS0);
    bitClear(ADCSRB, ADTS1);
    bitClear(ADCSRB, ADTS2);
```

```
    // start conversion
    bitSet(ADCSRA, ADSC);

    // In case the ADC is synced to timer0:
    // set timer0 prescale factor to 1 instead of 64
    // NOTE: delay(), micros() and millis()
    // functions are running faster!
    //TCCR0B = _BV(CS00);                //1
    //TCCR0B = _BV(CS01);                //8
    //TCCR0B = _BV(CS00) | _BV(CS01);    //64 =default
    //TCCR0B = _BV(CS02);                //256
    //TCCR0B = _BV(CS00) | _BV(CS02);    //1024

    // set timer2 PWM frequency to 31372.55Hz,
    // instead of the default 490.20Hz
    TCCR2B = (TCCR2B & B11111000) | B00000001;

    sei(); // enable interrupts

    analogWrite(PWM_PIN, 1);
    // dummy write to start PWM. After this the pulse width can
    // be updated directly via the OCR2B register for pin3.
}

volatile long adcVal, kp = 100, ki = 15, setpoint = 0;
volatile long integral = 0, i_out, error = 0, ctrlSignal=0;
int n = 0;

// The loop function is called in an endless loop
void loop()
{
  digitalWriteFast(LED_PIN, true);

  if(Serial.available() > 0)
  {
    char buffer[5] = {0};
    setpoint = Serial.readBytes(buffer,5);
    setpoint = atoi(buffer)/5;

    setpoint += 2*(setpoint/100);
  }

  n++;
  if( n >= 2500 )
  {
      Serial.print("setpoint:   ");
      Serial.print(setpoint);
      Serial.print("   adcVal:   ");
```

```
      Serial.print(adcVal);
      Serial.print("   iout:   ");
      Serial.print(i_out);
      Serial.print("   err:   ");
      Serial.print(error);
      Serial.print("   ctrlSignal:   ");
      Serial.println(ctrlSignal,DEC);
      n = 1;
  }

     digitalWriteFast(LED_PIN, false);// toggle LED on and off
}



/* ***************************************************************
 * ISR - ADC
 * Captures the selected ADC channel.
 * ***************************************************************
 */
ISR(ADC_vect){
  digitalWriteFast(DEBUG_PIN, true); // marker pin high

  // read 10 bits value from ADC from
  // the current selected input channel
  adcVal=ADCL; // store lower byte ADC
  adcVal+=ADCH<<8; // store higher bytes ADC

  // Implement your controller here if fast enough.
  // Keep a watch on the execution time of the
  // ISR with the oscilloscope on the debug pin!

  i_out = (adcVal - 531)>>1;
  error = setpoint - i_out;
  integral += ki * error;

  ctrlSignal = (5*((kp * error) + integral))>>10;

  if(ctrlSignal > 255)
      ctrlSignal = 255;
  else if(ctrlSignal <= 0)
      ctrlSignal = 0;

  OCR2B = (unsigned char) ctrlSignal; // write PWM value.

  digitalWriteFast(DEBUG_PIN, false); // marker pin low
} // (runtime about 1.6us to readout the adc)
```

# F Converting Float to Fixed Point

## F.1 Program Usage

The basic usage of the program to check the accuracy of the fixed point conversion is to call the program with the number of random test cases it should generate as its only argument. In this case the terminal output will look like:

```
> accuracy.exe 5
Case     | Decimal Value | Converted     | Difference
--------+---------------+---------------+------------
1        | 0.1025        | 0.1016        | 0.0009
2        | 15.8350       | 15.8350       | 0.0000
3        | 46.1675       | 46.1670       | 0.0005
4        | 47.9225       | 47.9219       | 0.0006
5        | 66.2500       | 66.2500       | 0.0000
--------+---------------+---------------+------------

Report:

         Avg. Deviation: 0.000422
```

Additionally, there is the option to only have the average deviation value printed to the terminal output. This is desirable when the number of test cases is large, because the bottleneck of the program execution is printing the values to `stdout`. The table of values is not shown if the "–silent" argument is passed as the program's second argument. An example application of this option:

```
> accuracy 100000 --silent
Report:

         Avg. Deviation: 0.000469
```

## F.2 Source Code

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define toFixed(x,FRACTION_BITS) ((int)(x * (1<<FRACTION_BITS)))
#define fixedToDouble(y,FRAC_BITS) ((double) y /(1<<FRAC_BITS))
int compare(double *a, double *b)
{
    return *a > *b ?  1 :
           *a == *b ? 0 :
                     -1;
}

int main(int argc, char *argv[])
```

```
{
    //Give cases and silent default values
    int cases = 100, silent = 0;

    if(argc > 1)
    {
        cases = atoi(argv[1]);
        if(argc == 3)
            silent = (strcmp(argv[2],"--silent") == 0) ? 1 : 0;

        if(cases > 0)
        {
            double *rand_vals = malloc(cases*sizeof(double));
            for(int i = 0; i < cases; i++)
                rand_vals[i] = (double)rand() / 400;

            //Sort the values so the output will be organized
            qsort(rand_vals,cases,sizeof(double),compare);

            if(silent == 0)
            {
                printf("Case\t| Decimal Value\t|");
                printf(" Converted\t| Difference\n");
                printf("--------+---------------+");
                printf("---------------+------------\n");
            }

            double sum = 0.0;
            for(int i = 0; i < cases; i++)
            {
                double converted = fixedToDouble(
                                    toFixed(rand_vals[i],10),10);

                double diff = rand_vals[i] - converted;
                sum += diff;
                if(silent == 0)
                {
                    printf("%i\t| %2.4f\t|",i+1,rand_vals);
                    printf(" %2.4f\t| %2.4f\n",converted,diff);
                }
            }
            if(silent == 0)
            {
                printf("--------+---------------+");
                printf("---------------+------------\n\n");
            }

            //Report the result
```

```
            printf("Report:\n\n\tAvg. Deviation: %f\n",sum/cases);

            //Free the memory allocated using malloc
            free(rand_vals);
        }
        else
            printf("The number of cases given is invalid.\n");
    }
    else
        printf("Improper usage of the program\n\n");

    return 0;
}
```