VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui

School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

Self Tuning Buck Converter

Niels Clayton

Supervisors: Daniel Burmester and Ramesh Rayudu

Submitted in partial fulfilment of the requirements for
Bachelor of Engineering with Honours.

Abstract

Switch-mode power supplies are commonly used in a wide variety of consumer and professional appliances to transform DC voltages with high efficiency. One such switch-mode supply is the buck converter, which steps down a DC voltage. The current buck converter design process requires that a specific output filter be designed around the switching frequency of the converter, the required output voltage, and the desired inductor ripple. This filter design process often results in the selection of non-standard components that are difficult to purchase or manufacture, increasing costs and leading to design compromises. This project will develop a platform that allows for observation and control of the inductor current ripple by modulating the switching frequency of the converter. This will allow engineers to design buck converters directly for the specified inductor current ripple their application can tolerate, eliminating the issues of designing this output filter.

Contents

1 Introduction **1**
- 1.1 Project Motivation . . . 1
- 1.2 Project Goals . . . 1

2 Background **3**
- 2.1 Pulse Width Modulated Signal Generation . . . 3
  - 2.1.1 Analogue PWM Signal Generation . . . 3
  - 2.1.2 Digital PWM Signal Generation . . . 4
- 2.2 Buck Converters . . . 4
  - 2.2.1 Buck Converter Design . . . 5
- 2.3 Current Sensing . . . 6
  - 2.3.1 Current Sense Amplification . . . 6
  - 2.3.2 Hall Effect Sensors . . . 6
- 2.4 Control Systems . . . 7

3 Design **8**
- 3.1 System Specifications and Architecture . . . 8
  - 3.1.1 System Architecture . . . 9
- 3.2 PWM Signal Generator . . . 10
  - 3.2.1 Analogue PWM Generator Design . . . 10
  - 3.2.2 Digital PWM Generator Design . . . 10
- 3.3 System State Sensing . . . 11
  - 3.3.1 Output Voltage Sensing . . . 11
  - 3.3.2 Current Sensor Selection . . . 12
  - 3.3.3 Average Inductor Current Sensing . . . 12
  - 3.3.4 Peak Inductor Current Sensing . . . 12
- 3.4 Control System . . . 14
  - 3.4.1 Output Load Voltage Controller . . . 14
  - 3.4.2 Inductor Current Ripple Controller . . . 15

4 Implementation **16**
- 4.1 PWM Signal Generation . . . 16
- 4.2 System State Sensing . . . 16
  - 4.2.1 Output Voltage Sensing . . . 16
  - 4.2.2 Inductor Current Sensing . . . 17
- 4.3 Control System . . . 18
- 4.4 Full system implementation . . . 19

5 Evaluation **20**
5.1 PWM Generation 20
5.1.1 PWM Generator Subsystem Evaluation 21
5.2 System State Sensing 21
5.2.1 Output Voltage Sensing 21
5.2.2 Inductor Current Sensing 21
5.2.3 State Sensing Subsystem Evaluation 23
5.3 Control System 23
5.3.1 Control Subsystem Evaluation 24

6 Conclusions **25**
6.1 Future Work 26

A System Specification Derivation **29**

B PWM Generation Figures **32**

C Peak Detector Figures & Tables **35**

D Control System Figures **43**

E Schematic & Printed Circuit Board **49**

F Breadboard Implementation **51**

G Source Code **53**

Chapter 1: Introduction

Historically power distribution has been primarily in the form of AC (Alternating current). This is credited to the fact that AC power is simple to produce, and made it easy to step up and down the voltages efficiently with transformers [1]. However, with the invention of solid-state electronics such as the metal–oxide–semiconductor field-effect transistor or MOSFET for short, it has become possible to efficiently step up and step down direct current (DC) voltages. This has been achieved by the invention of the switch-mode power supply, which has facilitated the continued reduction of size and increase in efficiency of electronics [2].
Today, switch-mode power supplies can be found in a wide variety of consumer and professional electronics, with some examples being laptops, phones, and any form of DC charger. Their widespread usage when compared to other DC-DC converters such as linear regulators can be attributed to their far greater efficiency. One such switch-mode power supply is the buck converter, which will step down a DC input voltage to a lower DC output voltage.

1.1 Project Motivation

Although buck converters are a widespread technology, they are not without their limitations and drawbacks. The current buck converter design process requires that a specific output filter be designed around the switching frequency of the converter, and the desired inductor ripple current. This filter design process will often result in the converter requiring discrete passive components that are non-standard and hard to source. This will usually result in the designer having to make compromises in their design for either the cost or the performance of the converter.
Another drawback of this design process is the static nature of both the filter and the switching components once they have been selected. This results in the buck converters desired inductor current ripple only being achieved at a very specific designed output voltage or load. This means that with current buck converter designs varying the desired output voltage or varying the output load will cause the inductor current ripple to vary. This is an issue, as very few loads are static and will not change during their operation.

1.2 Project Goals

The goal of this project is to develop a testing platform through which the effects of variable buck converter switching frequency on inductor current ripple can be observed and controlled. The aim of this is to eliminate the need to design the output stage of the buck converter by implementing a control system to maintain a specified inductor current ripple, and output voltage.

To achieve this goal, a proof of concept buck converter will be designed to operate as the testing platform. This converter will require a variable frequency pulse width modulated

(PWM) signal generator, and inductor current ripple sensing. This will allow for direct manipulation and control of inductor current ripple. The converter will also need to implement the full functionality of existing buck converter designs, including controlled output voltage regulation.
It is also important to note that as this project aims to produce a testing platform to be used as a proof of concept, there will be continued work undertaken after the projects conclusion. Because of this, it is important that the design and implementation undertaken in this project considers the ease of use and future expansion of the project.

The following list of specifications outlines the requirements the designed test platform will meet:

1. Operate with a 12V DC input supply voltage
2. Provide a selectable DC output voltage between 3V and 10V
3. Provide an output voltage precision of at least $\pm 5\%$ of the targeted output voltage
4. Provide a selectable inductor current ripple between 20% and 50% of the total output current
5. Provide a variable converter switching frequency between $1k$Hz and $100k$Hz
6. Provide an inductor current ripple precision of at least $\pm 5\%$ of the target inductor current ripple
7. Operate with variable load sizes between 10Ω and $20\ \Omega$

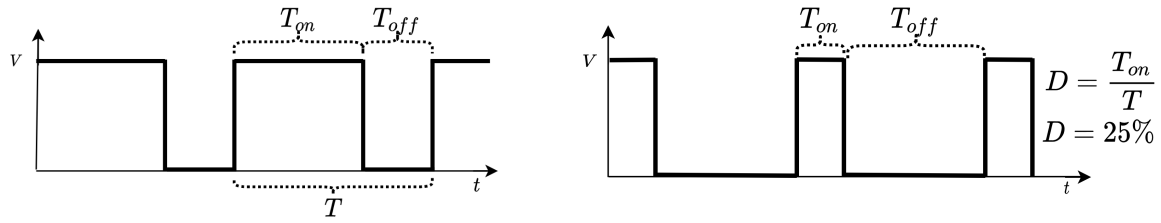
Chapter 2: Background

A literature research was performed to inform design decision made in this project, and to evaluate any existing research. This chapter will discuss buck converter design factors and topologies, as well as the various different methods of PWM generation, current sensing, and control implementation. In performing this literature research, we searched Google Scholar, Engineering Village, and Te Waharoa to find designs that utilised variable frequency PWM.
These searches returned no research relevant to the designs of this project, with the only related work focusing on the electromagnetic noise reduction using randomised frequency modulation [3, 4]. Because of this, research was instead performed to inform the design of the buck converter and the generation of PWM signals.

2.1 Pulse Width Modulated Signal Generation

Pulse width modulation (PWM) is a digital signal generation technique shown in Figure 2.1a, in which the Period T of the signal is held constant, while the ratio of its logic high period T_{on} to logic low period T_{off} is modulated. This ratio of high period to the low period is referred to as the duty cycle of the PWM signal and is often expressed as a percentage, this can be seen in Figure 2.1b.

PWM signals are used in a wide variety of applications for both digital and analogue electronics. PWM is often used to generate analogue signals from digital components by varying the average voltage of the digital PWM signal over time [5]. PWM is also used to control the switching elements contained within switch mode power supplies using this same principle, as discussed in Section 2.2. With regard to this project, we will be looking to generate a PWM signal that can be modulated in both duty cycle and frequency.



(a) Pulse width modulated signal

(b) Duty cycle calculation

Figure 2.1: Pulse width modulated signal characteristics

2.1.1 Analogue PWM Signal Generation

Designing a PWM signal generator using analogue components has three distinct stages required to generate the signal. These stages can be seen in Figure 2.2, and include clock

generation, triangle wave generation, and signal comparator stages [6].

The clock generation stage generates a square wave clock signal at a set frequency. This is usually done using a quartz crystal oscillator, or another form of resonating oscillator circuit. The triangle wave generating state must take the clock signal from the previous stage, and produce a triangle wave of the same frequency. This stage is most often done using a standard op-amp integrating circuit with unity gain at the resonating frequency of the clock source. The final signal comparator stage will convert this triangle wave into a PWM signal. Using a comparator, a reference voltage can be applied to the non-inverting input, and then the triangle wave can be applied to the inverting input. This will produce a pulse train with the same frequency as the clock source, where the period of T_{on} and T_{off} is set by the reference voltage.

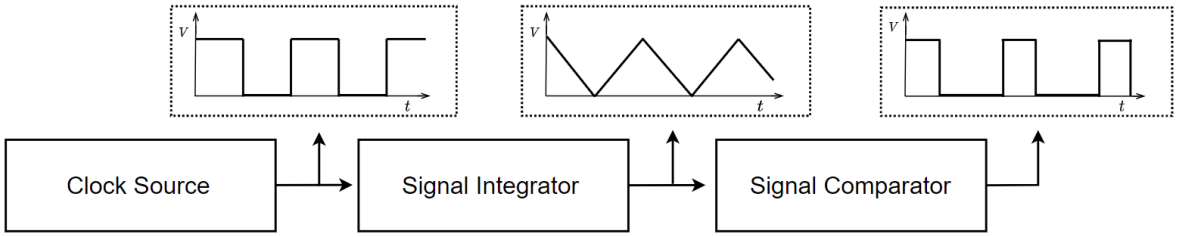


Figure 2.2: Stages of analogue PWM generation

2.1.2 Digital PWM Signal Generation

Designing a PWM signal generator with digital components is less complex than the method described in Section 2.1.1, and can be done using either a microcontroller or a Field Programmable Gate Array (FPGA). By using an internal timing register that is continually incrementing at a known period we can set a period for our PWM. It is possible to vary the period of the PWM by simply increasing or decreasing the timing registers clock. Next, by toggling a digital I/O when a compare variable is equal to the current value of the timer, we are able to generate a PWM signal with a variable duty cycle [7]. This timing architecture can be simply implemented within an FPGA, but it can also commonly be found within the hardware of most microcontrollers. However, It should be noted that the range of achievable frequencies and duty cycle accuracy will be dependant on an individual microcontrollers clock speed and internal register sizes.

2.2 Buck Converters

The buck converters is a variant of a switch mode power supply that steps down a DC input voltage to a DC output voltage. They are commonly used in a wide variety of consumer and professional appliances such as laptops, phones, and chargers due to their high efficiency compared to other DC-to-DC step down converters such as linear regulators [8].

The basic operational components of a buck converter can be seen below in Figure 2.3. From this we see that a buck converter has three main elements, the input voltage source, two switching components, and an output filter across the load. In the case of Figure 2.3, the first switching component is an actively controlled switch such as a MOSFET or transistor, and the second a passive switching diode. This configuration of an active and a passive switch is known as the non-synchronous buck converter topology, if the passive diode

were to be replaced with a second active switch the topology would be considered synchronous. Although both topologies function under the same fundamental principles, the non-synchronous topology is easier to implement with the drawback of higher losses and therefore lower efficiency.

It can also be seen from Figure 2.3 that a buck converter has two operating states that are controlled through the activation of these switching components. By toggling these switching components at high speed though the use of PWM, we can control the current flowing through the inductor of the output filter. By controlling this current we are also able to directly control the current through, and voltage across the output load of the converter. Using this, buck converters will often have a feedback control system in their design to be able to actively control and regulate the output voltage during usage. This controller will vary the duty cycle of the the switching PWM signal, thereby varying the output voltage of the buck converter as shown in Equation (2.1).

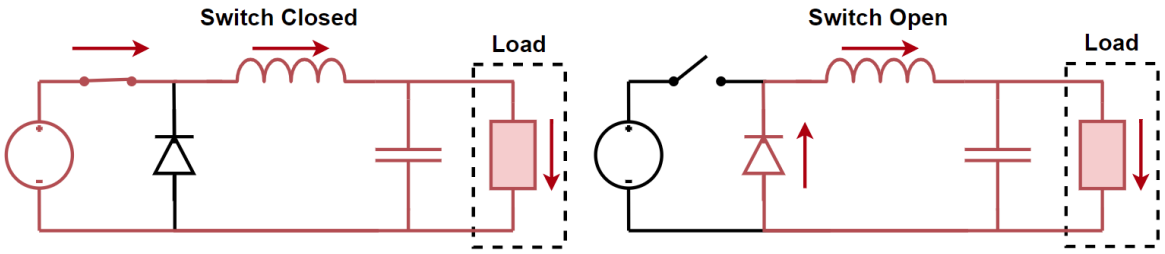


Figure 2.3: Operating states of a buck converter

2.2.1 Buck Converter Design

The design of a common buck converter has two primary considerations, the output voltage of the converter V_0, and the inductor current ripple of the converter Δi_L. These considerations can be specified by designing the buck converter using Equation (2.1) & Equation (2.2) [9, 10].

When designing a buck converter the first design specification that must be met is the output voltage. In Equation (2.1) the output voltage can be directly related to the input voltage V_{in} and the switching duty cycle D. Using this equation it is possible to directly set the output voltage of the buck converter by varying this duty cycle.

$$V_o = D \cdot V_{in} \tag{2.1}$$

Once the output voltage has been specified, the inductor current ripple can be calculated and specified with Equation (2.2). This equation allows for the inductor current ripple to be directly related to the inductor size L, and the PWM switching frequency f_s. This allows the the specification of the inductor current ripple through the varying of these two values.

$$\Delta i_L = \frac{V_o \cdot (1 - D)}{L \cdot f_s} \tag{2.2}$$

These two equations will be used to inform the designs and specifications of this project, and will be discussed in detail in Section 3.1.

2.3 Current Sensing

Current sensors are transducers that converter an input current to an easily measured output voltage that is proportional to the current through the sensor. Current sensors operate on one of two principles, Ohm's law, Faraday's and Ampere's law [11]. Ohm's law describes how when current flows through a resistive element a voltage drop will occur. Sensors operating under this principle will measure the voltage drop across a known resistance, and are referred too as direct sensors as they directly effect the circuit. Faraday's and Ampere's law describes how a changing electric flux will induce a changing magnetic field, and a changing magnetic field will induce a changing magnetic flux. Sensors operating under this principle will use the magnetic field induced by the current through the sensor to produce a voltage proportional to that current, and are referred to as indirect sensors as they make no direct contact with the circuit being sensed.

2.3.1 Current Sense Amplification

Current sense voltage amplification sensors are a form of direct sensor, working on the principle of Ohm's law $I = \frac{V}{R}$. By sensing the voltage dropped across a known value resistive element (Often called the current shunt), it is possible to calculate the current that is flowing through the element. This form of current sensing is easy to implement in theory, however it has the effect of altering the system being sensed by adding a resistive load, and thereby increasing the losses of the system.
To mitigate the effects of this sensing on a circuit, a smaller resistive load can be used. However, this will also decrease the measurable voltage across the load, and therefore decrease the precision of a taken measurement. Because of this, shunt based current sensors are often paired with an operational amplifier of known gain. This combination allows for accurate amplification of the shunts voltage drop, increasing the precision of measurements, and allowing for drastically smaller shunt resistors.

2.3.2 Hall Effect Sensors

Hall effect sensors are a form of indirect sensor, working on the principles of Faraday's and Ampere's law as discussed above. They function by measuring the magnetic flux density, meaning that they are commonly used to sense the presence of a magnets or magnetic fields. However, since a current flowing though a wire will produce a magnetic flux, they can also be used to measure current flow.
Due to this operation, they are able to sense a current without contacting or altering the circuit. This means that they are commonly used in the measurement of high voltage or high current circuits, as they will remain completely isolated from the circuit being sensed. This provides large safety advantages, eliminates sensing power losses, and can often decrease the complexity of the sensing circuit.
However, hall affect sensors do suffer from a lack of precision. Due to their operation, their measurements will constantly be offset by any ambient magnetic flux within the surrounding environment. This means that they are not commonly used for the sensing of small signal currents, as the noise floor of the Earth's own magnetic field can often be larger than the signal being sensed [12].

2.4 Control Systems

Control systems are integral to the operation of any system for which you wish to achieve a desired outcome. In their simplest, a control systems has the purpose of obtaining a desired output from a system, with a desired performance, when a given input is provided.

There are two main topologies of control system, open loop and closed loop. An open loop control system requires no feedback of the output, and therefore no sensing. Because of this, open loop controllers are often fast to implement within a system because of this. However, it should be noted that they possess no knowledge of the systems state and are therefore unable to correct for errors such as external disturbances.
On the other hand, closed loop controllers require feedback from the current output of a system, and therefore will require sensing to be implemented. This feedback provides the ability to compensate for errors in the systems output, with the drawback of add complexity.

The performance of a specific control system is often discussed in terms of it's steady state settling time, and it's stead state error. The steady state settling time of a controller describes the time it takes for a controller to reach it's final value ('steady state') when provided with a given input. The slower the steady settling time of a controller, the slower it will react to a given input, and vice versa. The steady state error of a controller describes the error between the desired output and the final output of the system once it's steady state has been achieved.

Chapter 3: Design

The design processes outlined within this report will draw heavily on the background research discussed in Chapter 2. These designs aim to effectively implement the outlined system requirements discussed by Section 1.2 in a robust and repeatable manner.

3.1 System Specifications and Architecture

Section 1.2 outlines a list of specifications that this project must achieve. For each item in this list, design specifications can be derived to ensure that our designs are capable of meeting the outlined requirements. This list of specifications will also help to shape the architectural approach taken on this project.

Requirement 1: Operate with a 12V DC input supply voltage

To meet this requirement, the buck converter must be designed to be 12V DC compatible. this will require all designed hardware, including additional digital and analog circuity to be capable of operating with this supply.

Requirement 2: Provide a selectable DC output voltage between 3V and 10V

To meet this requirement, the designed buck converter must have the ability to change the duty cycle of it's PWM switching signal. This will allow for variation of the output voltage based on Equation (2.1).

Requirement 3: Provide an output voltage precision of at least $\pm 5\%$ of the targeted output voltage

To meet this requirement, the designed buck converter must have a control system to regulate the output voltage. To achieve a steady state error of less than $\pm 5\%$, feedback should be implemented, requiring output voltage sensing. We can also specify that a minimum duty cycle resolution of 1.25% will be required to achieve this. For the derivation of these values, refer to Appendix A.

Requirement 4: Provide a selectable inductor current ripple between 20% and 50% of the total output current

To meet this requirement, the designed buck converter must have the ability to change the frequency of it's PWM switching signal. This will allow for variation of the inductor peak to peak current ripple based on equation Equation (2.2).

Requirement 5: Provide a variable converter switching frequency between 1kHz and 100kHz

To meet this requirement, the PWM switching frequency must be selectable across the specified range. This, in tandem with requirement 4, allow for the specification of the inductor range for which these requirements can be met. For the derivation of these values, refer to Appendix A.

Requirement 6: Provide an inductor current ripple precision of at least $\pm$5% of the target inductor current ripple

To meet this requirement, the designed buck converter must have a control system to regulate the inductor current ripple. To achieve a steady state error of $\pm$5%, feedback should be implemented, requiring inductor current ripple sensing. We can also specify that a minimum frequency selection resolution of 200Hz, as well as a minimum current ripple measurement resolution of a resolution of 15mA will be required to achieve this. For the derivation of these values, refer to Appendix A.

Requirement 7: Operate with variable load sizes between 10Ω and 20Ω

To meet this requirement, all components of the buck converter must be designed to handle the maximum output current of 1A that will occur with a 10Ω load at 10V. It will also be important that any current sensing is designed to operate within the range defined by these loads. Finally it is required that control systems are implemented to maintain requirements 3, & 6 as the load varies.

3.1.1 System Architecture

Based on the specifications outlined in the above section, an architecture that outlines the system design has been developed. This architecture has been divided into three subsystems that must each be developed. This architecture system can be seen in Figure 3.1.

Subsystem one implements a PWM generator. From requirements 2, & 5 it has been identified that a PWM generator capable of varying both the duty cycle and the frequency of it's output signal independently must be designed. This system can be seen in Figure 3.1 outlined in green.

Subsystem two implements system state sensing. From requirement 3 & 6 it has been identified that an output voltage sensor, and an inductor current ripple sensor must be designed. This system can be seen in Figure 3.1 outlined in blue.

Subsystem three implements two control systems. From requirements 3, 4, 6, & 7 it has been identified that control systems should be designed to ensure that the specified steady state errors of the output voltage and inductor current ripple are met.
One control system will take the output load voltage measurement, and control the PWM duty cycle. The other control system will take the inductor current ripple, and control the PWM frequency. This system can be seen in Figure 3.1 outlined in red.

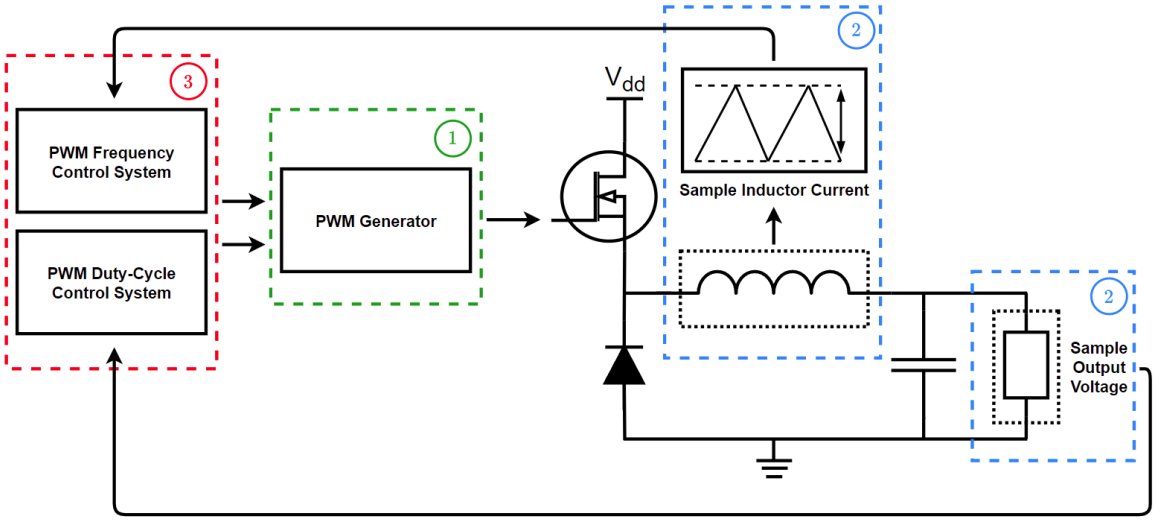


Figure 3.1: System architecture design, with three internal subsystems

3.2 PWM Signal Generator

In the design of the PWM generator, both the analogue and digital topologies discussed in Section 2.1 were considered, and designed for. Each of these designs presented pros and cons that would affect the overall design. This section will discuss these design choices, and finalise the design of the PWM signal generator.
A successful design will meet all of the requirements outlined by Section 3.1. Therefore the design must be capable of duty cycle variation with a resolution of 1.25%, as specified by requirements 2 & 3, as well as switching frequency variation between 1kHz & 100kHz with a resolution of 200Hz, as specified by requirements 4, 5, & 6. A successful design will also adhere to the ease of use and future expansion goal specified in Section 1.2.

3.2.1 Analogue PWM Generator Design

It has been discussed in Section 2.1.1 that the design of an analogue PWM signal generator requires three distinct stages. These stages are signal clock generation, signal integration, and signal comparison, as shown in Figure 2.2.
Designs for these sections were implemented within the circuit simulation software LTSpice, see Appendix B. From these simulations it was identified that the signal integration stage of the design functions as a first order low pass filter, where integration occurs beyond the filter's cut-off frequency. This causes a 40dB attenuation of the output signal across the required frequency range. Due to this, the design was identified to be unsuitable for this system as it was unable to meet requirements 4, 5, & 6.

3.2.2 Digital PWM Generator Design

As discussed in Section 2.1.2, The design of a digital PWM signal generator is based on the toggling an output when a timer reaches a given comparison value. This design can be achieved with either discrete logic designed within an FPGA, or using existing hardware within a microcontroller.

FPGA PWM Generator

The operation of an FPGA allows for the design and implementation of customised logic. Because of this, designs utilising FPGA's would easily be able to meet the specified requirements for the PWM signal generator.
However, the designs integration into the system architecture described in Figure 3.1 must also be considered. A PWM generator implemented in an FPGA will be required to either implement or provide an interface with the control subsystem, as well as the state sensing subsystem. These interfaces must also be designed, greatly increasing the complexity of this specific subsystem. Due to this added complexity, it was identified that this design was not suitable for an initial proof of concept, and would not adhere to the ease of use goal.

Microcontroller PWM Generator

The selection of the microcontroller is highly dependant on the clock frequency and internal PWM peripheral architecture. Since these vary greatly between different processors, a selection of microcontroller data-sheets were reviewed to identify their specifications.
The microcontrollers reviewed were selected based on the ease of use of their available development environments. This included AVR, STM8, Espressif, and teensy based microcontroller development boards.
From this review it was identified that the ESP32 microcontroller [13] would be best suited to this project. This microcontroller is capable of outputting a maximum PWM frequency of $125kHz$ when a duty cycle resolution of 9 bits is selected. This will provide a total of 512 selectable duty cycle values, for a resolution of 0.2%, while providing a selectable duty cycle frequency between 1Hz, & 125kHz.
Based on these specifications, it is clear that this design meets the duty cycle requirements of 2 & 3, as well as the frequency requirements of 4, 5, & 6, and is therefore suitable.

3.3 System State Sensing

The design of the state sensing system will include the output voltage sensor and the inductor current ripple sensor. These sensors will be responsible for providing the feedback data for the control systems, illustrated in Figure 3.1. To provide a peak to peak inductor current measurement as a percentage of the output current, both the average and peak inductor currents must be measured. Based on these measurements, the inductor current ripple percentage can be calculated.
A successful design will meet all the requirements outlined by Section 3.1. Therefore the output voltage sensor must be capable of measuring voltages between 3V & 10V with a resolution of 150mV, as specified by requirements 2 & 3. It is also required that the average inductor current, and the peak inductor current ripple measurements achieve a resolution of 15mA, for frequencies between 1kHz & 100kHz, as specified by requirement 4, 5, 6, & 7.

3.3.1 Output Voltage Sensing

The output voltage of the buck converter will be a DC voltage between 3V & 10V. As there are no high frequency components to this signal, there are no bandwidth requirements on it's sampling. Due to this the internal analog to digital converter (ADC) of the ESP32 microcontroller will be used for sampling. This 12 bit ADC can measure an input voltage range of 0V to 3V [13], therefore the output voltage will have to be measured through a voltage divider. This divider will provide an output ratio of $\frac{10}{3.33}$, allowing for maximal use of the ADC's input range. This will provide a theoretical measurement resolution of 2.5mV.

Based on these specifications, this design is capable of measuring output voltages between 3V and 10V, with a resolution of 2.5mV, meeting the outlines requirements 2 & 3.

3.3.2 Current Sensor Selection

To maintain design simplicity, both the average inductor current, and the peak inductor current ripple will be measured from the same sensor. Due to this, a hall effect based current sensor is not suitable for this application. It was discussed in Section 2.3.2 that they lack precision due to a high noise floor, and would be unable to meet requirements 4 & 6.
Because of this, a current shunt based sensor with a defined gain and selectable shunt resistance has been designed, specifically the LT1999-50 with a 60mΩ shunt. This current sense amplifier has a specified gain of 50V/V, a typical gain error of 0.2%, and a bandwidth of 2MHz [14]. This sensing solution will provide a voltage output between 0.67V & 3.72V across the total current sensing range, with a resolution defined by the gain error of 50μA at the output. Refer Appendix A for the calculation of these values. The datasheet also specifies that there will be no attenuation of the output gain for frequencies of 100kHz or below.
Based on these specifications, this design is capable of measuring the full current input range specified in requirements 4 & 7, while surpassing the resolution and frequency specifications of requirements 5, & 6.

3.3.3 Average Inductor Current Sensing

To obtain the average inductor current ripple, the output of the previously designed current sense amplifier can be low pass filtered. By designing a first order passive low pass filter with a cut-off frequency of 10Hz, we can be sure to attenuate any selectable switching frequency by at least 40dB. This will provide a simple to measure DC voltage between 0.45V and 3V. This can be measured using the internal ADC of the ESP32 providing a theoretical resolution of 700μV, or 244μA. Refer Appendix A for the calculation of these values.

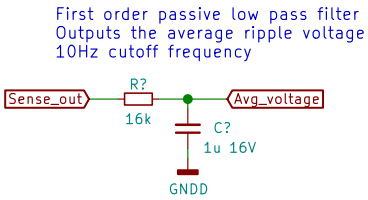


Figure 3.2: Average current 10Hz cut-off low pass filter

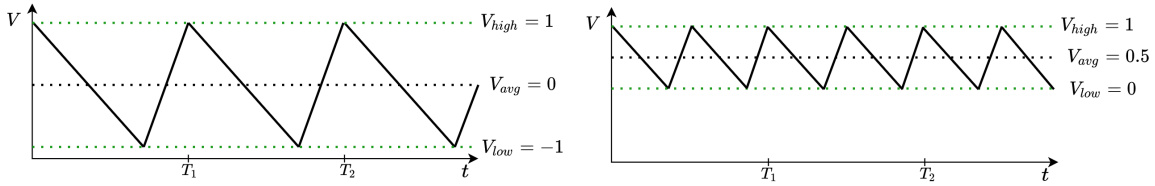
3.3.4 Peak Inductor Current Sensing

Based on the bandwidth requirement of 1kHz to 100kHz, it was identified that sampling the output signal of the current sensors to identify the peaks would be impractical. To achieve this a sampling rate of at least twice the highest frequency component would be specified by the Nyquist theorem, and a recommended minimum of one decade above is considered common practice, requiring a sampling rate of 1Mhz.
Due to this, designs were undertaken to simplify the sampling requirements. The following designs will take direct input from the current sensor, and aim to output a DC voltage that is representative of the input current peak values.

Precision Rectifier Peak Voltage Detection

This design is based around the operation of a signal precision rectifier, and the knowledge that the inductor current ripple will be symmetrical and triangular in shape [9]. Due to this symmetry, we know that the average inductor ripple current over a period will be zero, shown in Figure 3.3 (a). After rectification of the signal, the the ripple peak to peak value

will halve, the frequency will double, and the new signal average will be half the new ripple peak to peak, shown in Figure 3.3 (b).



(a) Pre-rectification current sense output of 1V pk to pk, average voltage of 0V

(b) Post-rectification current sense output of 1V pk to pk, average voltage of 0.5V

Figure 3.3: Average voltage output of a symmetrical signal, before and after rectification

From here, we can obtain the signal average by applying a passive a low pass filter with a cut off of 10Hz to the output, producing a sampleable DC voltage. To obtain the full original signal peak to peak value, this signal can then amplified with a gain of 4. The full design can be seen in Figure 3.4.

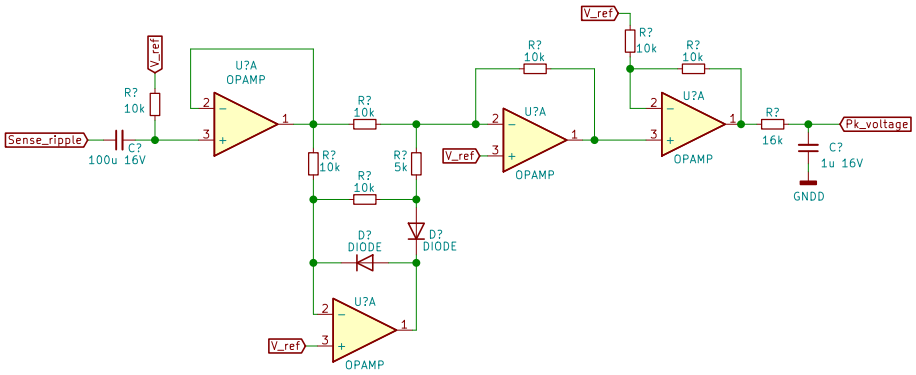


Figure 3.4: Precision rectifier peak detector designed circuit schematic

Sample and Hold Peak Voltage Detection

This design is based on existing voltage peak detection designs [15, 16]. The design operates on a similar principle to a sample and hold unit, charging a signal capturing capacitor through a diode such that it will not discharge. In order to remove the voltage drop across the charging diode, it is placed within the forward path of an operational amplifier buffer. The output voltage is then buffered again to allow for sampling. The full design can be seen in Figure 3.5.

Both of the discussed designs have been simulated in LTSpice, and have had their operation confirmed, refer to Appendix C for output plots. From this it was identified that both designs would be able to meet the resolution specification of 15mA, and bandwidth specification of 1kHz to 100kHz outlined in requirement 4, 5, 6, & 7.

It should also be noted however that the design of the precision rectifier based peak detector is much more complex than that of the sample and hold peak detector. Because of this, it was decided that the sample and hold based peak detector would be most suitable for this project, as it meets all required specifications, and best adheres to the ease of use goal.

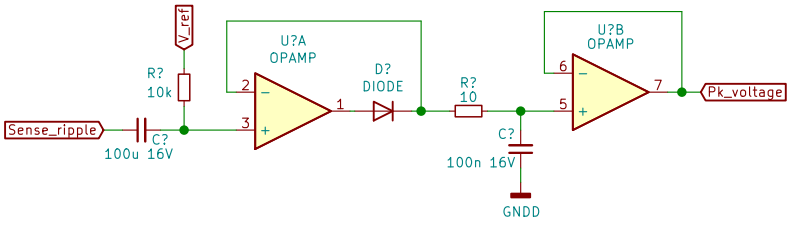


Figure 3.5: Sample & hold peak detector designed circuit schematic

Current sensing digitisation

Due to the operation of the design discussed in Section 3.3.4, the output DC voltage will be imposed on a DC refrence voltage. To eliminate this DC bias from the output, a dedicated external ADC with a differential input can be employed. This will allow for the removal of the bias at the digitisation of the signal. The selected ADC is the MAX11644, which provides a 12bit resolution, an internal reference of 4.096V, and the required differential input. Based on these specifications, this design is capable of measuring input voltages between 0V and 4.096V, with a resolution of 8mV. This will give a current sensing resolution of 2.6mA, meeting the specifications of requirement 6.

3.4 Control System

When designing the output voltage and inductor current ripple control systems, a successful design will meet all the requirements outlined by Section 3.1. From this we can identify that both controllers must provide a steady state error that is less $\pm 5\%$ of the targeted value, as specified in requirements 3 & 6.

3.4.1 Output Load Voltage Controller

To achieve the desired output voltage steady state error, a feedback controller with an integrating term should be used. To provide this, the design will implement a proportional, integral, & differential (PID) controller. This controller will be provided with the buck converter output voltage as an input, this will then be converted to the theoretical PWM duty cycle required to produce that output based on Equation (2.1). This will then be used to calculate the controller error and output the new PWM duty cycle.
To design this, a mathematical model of a buck converter [17] was found and simulated in Matlab. This model was then discretised such that it could be implemented on a microcontroller, and a PID controller was then designed with the '`PIDTune()`' function. A step response simulation of this model can be found in Appendix D.
Due to the control topology used in this design, this design will be able to provide an output voltage steady state error of 23mV based on the discrete voltage step size from the PWM generator. This will provide a worst case error of $\pm 0.78\%$, meeting the $\pm 5\%$ specified in requirement 3.

3.4.2 Inductor Current Ripple Controller

To achieve the desired inductor current ripple steady state error, the same topology implemented for the output voltage controller will be implemented. This PID controller will take the system's inductor current ripple as a percentage of the average current, and then calculate the error between it and the target value. From here it will then output the new PWM switching frequency.
Due to the control topology used in this design, this design will be able to provide an inductor current ripple steady state error of less than $\pm 5\%$, as specified in requirement 6.

Chapter 4: Implementation

4.1 PWM Signal Generation

The PWM signal generator subsystem was designed around the Espressif ESP32 microcontroller. For ease of development around this microcontroller, a pre-built breadboard compatible development board was purchased for the implementation.

Using this development board, a PWM hardware driver was developed to facilitate the implementation of this subsystems core functionality as specified in Section 3.1. This driver implements three core functions, '`PWM_setup()`' to initialise the PWM hardware, '`PWM_set_duty()`' to select a new duty cycle, and '`PWM_set_frequency`' to select a new frequency. The full driver implementation, refer to Appendix G.

After the software implementation had been completed, it was identified that the 3.3V digital output of the microcontroller would be incapable of driving the buck converters switching power MOSFET. To resolve this, the IR2125 high side N-channel gate driver IC was purchased to drive the MOSFET from the PWM generators output signal. This final circuit was then implemented and it's functionality was tested on a breadboard. Please see Appendix E for the designed schematic, and Appendix F for the implemented breadboard circuit.

4.2 System State Sensing

4.2.1 Output Voltage Sensing

The output voltage sensing design discussed in Section 3.3.1 was implemented on a prototyping breadboard. Refer to Appendix E for the designed schematic, and Appendix F for the implemented breadboard circuit.
Using the ESP32 development board, an ADC hardware driver was developed to facilitate the measurement of the sensor output. This driver implements the core functionality of the ADC though two functions, '`init_adc()`' to assign the input IO and measurement resolution, and '`read_adc()`' to take an ADC measurement. Using these two functions this driver then also facilitates the computation of a rolling average with the '`rolling_average()`' function, and voltage conversion with the '`adc_conversion()`' function. The full driver implementation can be found in Appendix G.

When initially implementing the ADC conversion function, it was assumed that that the output conversion of the ADC was linear. Once this had been implemented, it was observed that there were large inaccuracies in the voltage conversion.
From this it was identified that the internal ADC was non-linear, and would require calibration before meeting the required specifications. To achieve this, ADC measurements were taken for a selection of known input voltages ranging from 100mV to 3V in 500mV

steps. These ADC measurements were then plotted against their respective input voltages using Matlab, and a forth order polynomial was fit to the data, shown in Figure 4.1. This polynomial was then used for the conversion of the raw ADC reading to a voltage, greatly improving the ADC's accuracy.

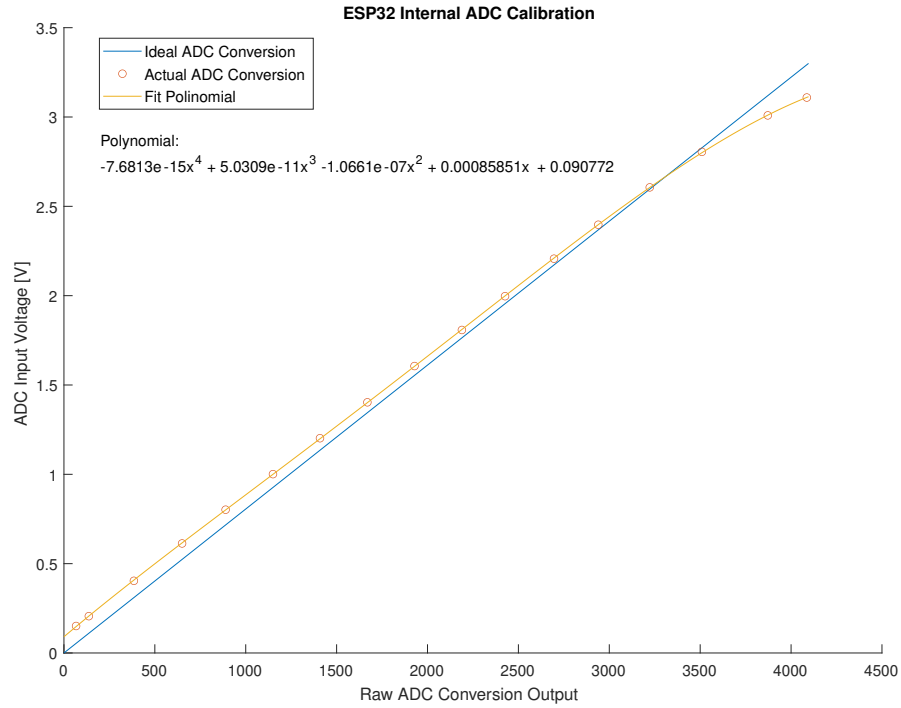


Figure 4.1: ESP32 internal ADC calibration data with forth order polynomial fit

4.2.2 Inductor Current Sensing

Average Inductor Current Sensing

The average inductor current sensing design discussed in Section 3.3.3 was implemented on a prototyping breadboard. Refer to Figure 4.2 for the designed schematic, and Appendix F for the implemented breadboard circuit.
This design also utilises the ESP32 internal ADC, which was previously implemented in Section 4.2.1. Therefore no further implementation was required for the functionality of this design.

Peak Inductor Current Sensing

The peak inductor current sensing design discussed in Section 3.3.4 was initially implemented on a prototyping breadboard, using an LT084CN operational amplifier (op amp) provided by the lab technicians.
On implementation it was observed that the functional principles of the design held true, as the design maintained a DC output voltage that was dependant in the input peak to peak ripple. However, it was observed that the accuracy of the design was much lower than expected when compared to the design simulations. The operation of this original design can be seen in Appendix C.

It was identified that the limiting factor within the design was the bandwidth and slew-rate of the input stage op amp seen in Figure 3.5. This op amp is responsible for charging the sampling capacitor and negating the forward bias diode drop, and therefore it's bandwidth and slew-rate will dictate the bandwidth of the design.
Based on this, the input op amp was replaced with the OPA2830, as it provided a greatly improved bandwidth of 230MHz and a slew rate of 500V/μs. The original design was also altered to place a potentiometer in series with the sampling capacitor, as well as adding a high impedance resistor in parallel with this capacitor. This allows for the final designs performance to be tuned through the potentiometer, and any overshoot to be quickly dissipated by the parallel discharge resistor. This final design of the schematic can be seen in Figure 4.2, and the implemented breadboard circuit can be found in Appendix F.

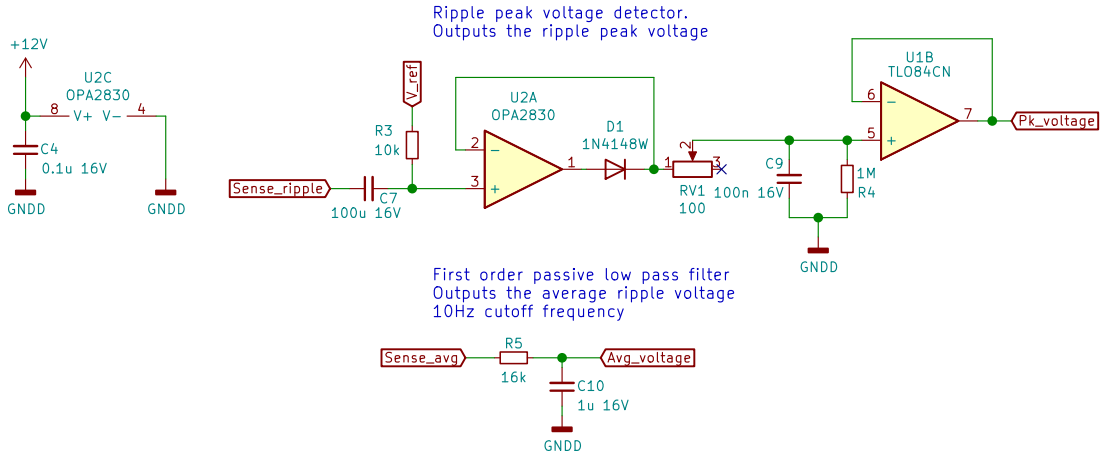


Figure 4.2: Finalised peak detector schematic, with updated op amp selection and performance improvements

4.3 Control System

The control system design discussed in Section 3.4 has been implemented in the form of a generic PID based control system library. This control system implementation provides a structure in which all of the controller variables and states are stored. This simplifies the initialisation of the controller, as the proportional, integral, derivative, and time period constants can all be specified at the structure initialisation. This implementation also allows for the creation and management of multiple controllers simultaneously, meeting requirements 2 & 4.
This control library also provides more advanced safety features for the system, including selectable minimum and maximum output limits, and selectable integrator windup limits.

This library implements two core functions which provide the full functionality of a digital PID control loop. '`PID_init()`' initialises and sets up the controller, and '`PID_update()`' calculates the new output of the controller. The full driver implementation can be found in Appendix G.

4.4 Full system implementation

Software implementation

The full system software implementation is built upon the ESP-IDF framework, and utilises a FreeRTOS to provide tasks and scheduling.
The ESP-IDF Framework was chosen over other supported frameworks due to it's increased control of the ESP32 available hardware peripherals, which was required for the designs system to meet it's specified requirements from Figure 3.1.
The use of the ESP-IDF framework also facilitated the use of FreeRTOS, providing access to a task scheduler that would greatly decrease the structural complexity of this system. This scheduler allows each subsystem within the design to operate from within a task, independent of all others. therefore, this implementation reduces the complexity of further developments within this project, without limiting the functionality of the microcontroller platform. The full system software implementation can be found in Appendix G.

Hardware implementation

To facilitate the ease of use of the fully designed system, a printed circuit board (PCB) was designed and implemented using the KiCAD software platform. This PCB implements all finalised designed elements discussed in Chapter 3, and provides multiple footprints for the buck converter output filter to allow for expedited testing and varying filters. For the full system schematic, refer to Appendix E.

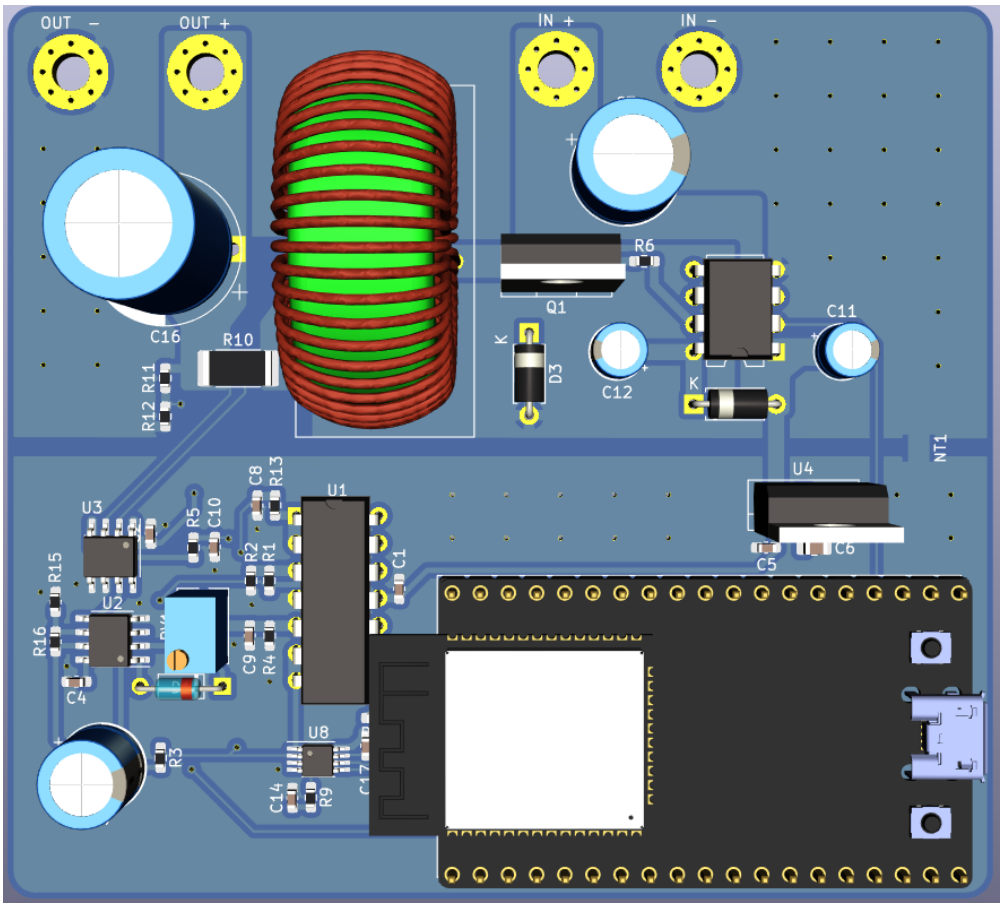


Figure 4.3: Full system printed circuit board implementation

Chapter 5: Evaluation

This evaluation will evaluate the the success of the designed and implemented components in achieving their specified requirements as outlined in Chapter 3.

5.1 PWM Generation

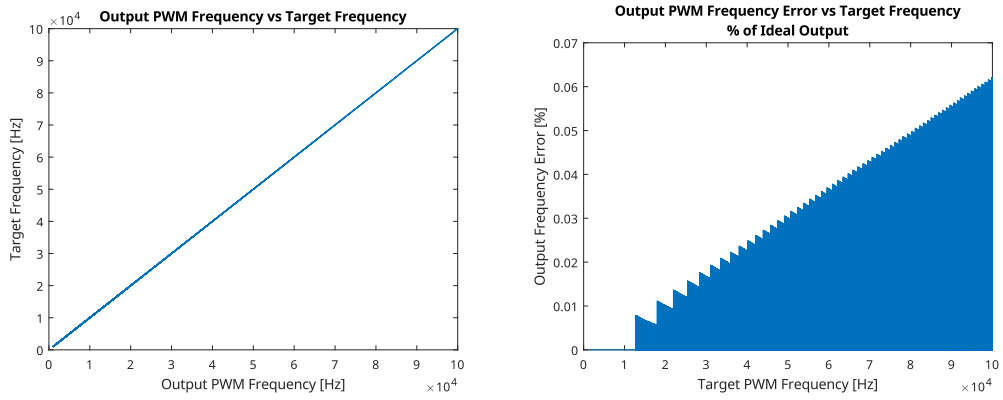
It was specified in Section 3.2 that a successful PWM generator subsystem must be capable of duty cycle variation with a resolution of 1.25%, as specified by requirements 2 & 3. The successful subsystem must also be capable of varying the switching frequency between 1kHz & 100kHz with a resolution of 200Hz, as specified by requirements 4, 5, & 6.

Duty Cycle Variation

The design discussed in Section 3.2.2 outlines it's capabilities for selectable PWM duty cycle, with a resolution of 0.2%. This precision was verified through visual inspection of oscilloscope measurements at various selected duty cycles, as well as numerically, plotting each targeted duty cycle against the systems output.

Switching Frequency Variation

The design discussed in Section 3.2.2 outlines it's capabilities for selectable PWM frequency between 1Hz, & 125kHz. This frequency range was verified through visual inspection of oscilloscope measurements at various selected frequencies across the designed range.



(a) Output PWM frequency plotted against target frequency

(b) Output PWM frequency error as a percentage of target frequency

Figure 5.1: PWM frequency selection

It was also specified that the design would provide a PWM frequency selection error of no more than 200Hz for a provided frequency. This was evaluated numerically by plotting the

target output frequency against the systems provided output, and can be seen in Figure 5.1 (a). To allow for easier visualisation of this error, it was calculated and plotted as a percentage of the target frequency Figure 5.1 (b). From this data, it was found that the maximum output frequency error of the system was 64Hz, or 0.064% of the target output.

5.1.1 PWM Generator Subsystem Evaluation

The PWM generator subsystem was required to provides a duty cycle selection resolution of at least 1.25%, and a frequency selection resolution of at least 200Hz, across the 1kHz to 100kHz range. The performed tests show that the implemented subsystem meets these specifications, providing a duty cycle selection resolution of 0.02%, and a frequency selection resolution of 64Hz. For the full range of PWM duty cycle and frequency test, measurements, and plots, refer too Appendix B

5.2 System State Sensing

It was specified in Section 3.3 that a successful state sensing subsystem will provide two sensing elements. The first, an voltage sensor capable of measuring voltages between 3V & 10V with a resolution of 150mV, specified by requirements 2 & 3. The second, an inductor current sensor capable of measuring average current and the peak current ripple with a resolution of 15mA, for frequencies between 1kHz & 100kHz, as specified by requirement 4, 5, 6, & 7.

5.2.1 Output Voltage Sensing

The design discussed in Section 3.3.1 outlines it's capability of measuring a DC voltage between 3V & 10V, with a measurement resolution of 2.5mV. This measurement precision was evaluated by comparing a selection of known input voltages between 3V and 10V to their measured outputs from the implemented design.

From this testing it was identified that a final precision of 10mV was achieved by the design, with signal noise from the ADC presenting as the main design limitation. Although the evaluated resolution of the design does not meet it's designed specification, it still provides a resolution 15 times greater than required.

5.2.2 Inductor Current Sensing

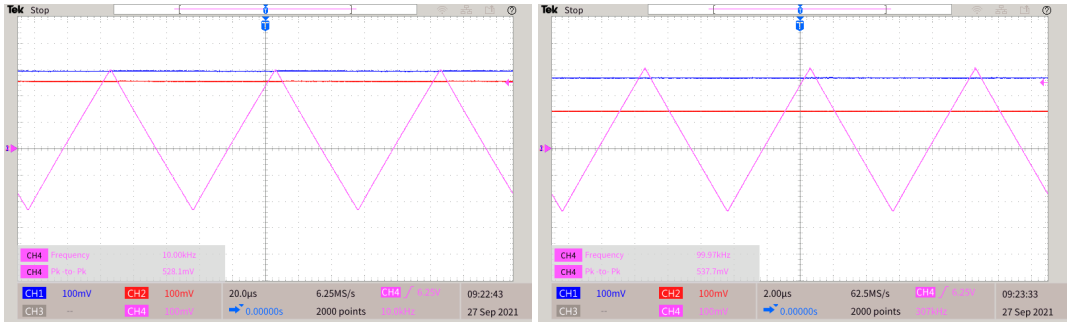
Average Inductor Current

The design discussed in Section 3.3.3 outlines it's capability to provide an average current measurement for frequencies between 1kHz and 100kHz, with a resolution of 244μA. The frequency range and resolution were evaluated by comparing a selection of known input voltages between 100mV and 3V to their measured outputs from the implemented design. This was then repeated for frequencies of 1kHz, 50kHz, and 100kHz.

From this testing it was identified that a final precision of 3.5mV was achieved by the design across all frequencies, with similar design limitations as the output voltage sensing. This provided a final average current measurement resolution of 1.1mA.

Peak Inductor Current

The design discussed in Section 3.3.4 outlines it's capability to provide a peak current measurement for frequencies between 1kHz and 100kHz, with a resolution of 15mA. As discussed in Section 4.2.2, the operation of the original design did not approach the resolution suggested by the simulations, and as such a redesign was undertaken to improve this.
The precision and functional frequency range of both the initial and final designs were evaluated and compared. In this evaluation, a known peak to peak voltage was input to the design, and compared to it's output DC voltage for a range of frequencies from 1kHz to 100kHz. This was then repeated for input voltages of 150mV, 500mV, and 1500mV, spanning the entire possible input range from the system. An oscilloscope screenshot of this testing for an input of 500mV, and frequencies of 10kHz and 100kHz can be seen in Figure 5.2.

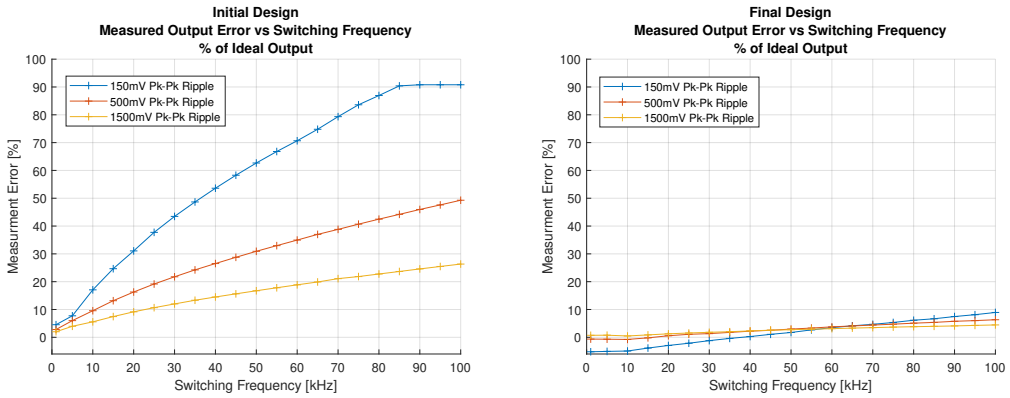


(a) Initial (Red) & final (Blue) peak detetor outputs for a $10kHz$ 500mV triangle input

(b) Initial (Red) & final (Blue) peak detetor outputs for a $100kHz$ 500mV triangle input

Figure 5.2: Initial and final peak detector design

From this testing, plots comparing the total error of the two designs as a percentage of the input peak to peak signal were generated, and can be seen in Figure 5.3. From these plots it can be observed that there is an increase in the total error for both designs as the input frequency increases, and a decrease in the total error as the input ripple increases. It can also clearly be seen that the final design provides substantially lower error for all input frequencies and peak to peak values, with a largest recorded error of 8.9% when operating at the minimum input ripple and maximum frequency.



(a) Initial design peak current sensing error as a percentage of actual peak current against frequency

(b) Final design peak current sensing error as a percentage of actual peak current against frequency

Figure 5.3: Initial and final peak current sensing design, percentage of actual peak current against frequency

From this testing it was identified that a worst case precision of 13.35mV was achieved by the design with an 100kHz input of 150mV peak to peak. After conversion, this provided a worst case peak current measurement resolution of 4.45mA.

5.2.3 State Sensing Subsystem Evaluation

The state sensing subsystem was required to provide measurements of the buck converters output voltage to a resolution of 150mV, as well as inductor average current and peak current ripple measurements across the 1kHz to 100kHz range to a resolution of 15mA. The performed tests show that the implemented subsystem meets these specifications. This subsystem provides an output voltage measurement resolution of 10mV, an average current sensing resolution of 1.1mA, and a worst case peak ripple sensing resolution of 4.45mA. For the full range of testing, measurements, and plots, refer too Appendix C.

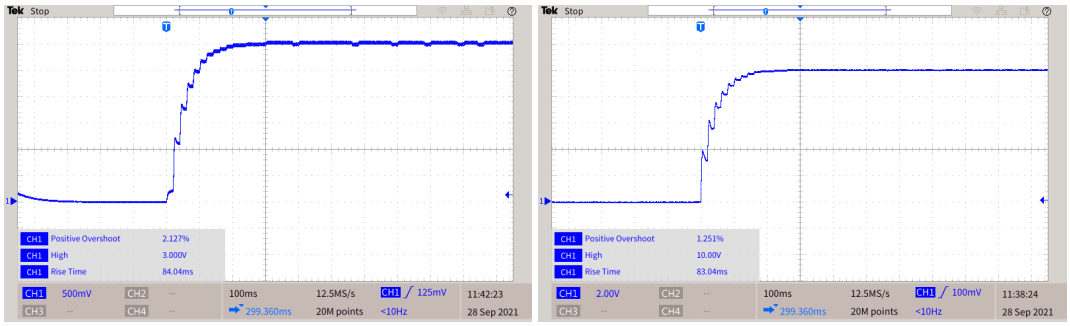
5.3 Control System

It was specified in Section 3.4 that a successful control subsystem will ensure a steady state error of no more than $\pm 5\%$ across the output range of 3V to 10V, as specified by requirement 3. It can be noted that the success of this control subsystem relies heavily on the designed specifications of the PWM generator subsystem to implement the selected duty cycle and frequency, as well as the state sensing subsystem to provide reliable and accurate feedback.

Output Voltage Control

The design discussed in Section 3.4.1 outlines it's capability to provide an output steady state error of 23mV, or $\pm 0.78\%$. This implemented controller was evaluated using a number of methods, characterising the subsystems steady state error and step response across the across the 3V to 10V output range, as well as the controllers supply change rejection.

The step response of the system was evaluated by selecting a system output of 0V DC, and then providing supplying an output target voltage step between 1V DC and 10V DC. From these tests it was observed that there was no output overshoot from the control system, and a consistent settling time of 84ms was observed for all step input values. Oscilloscope screenshots of this controller response can be seen in Figure 5.4.
It was also observed from these tests that the designed steady state error of $\pm 0.78\%$ was achieved, with small output fluctuations of 23mV noticeable in Figure 5.4 (a).

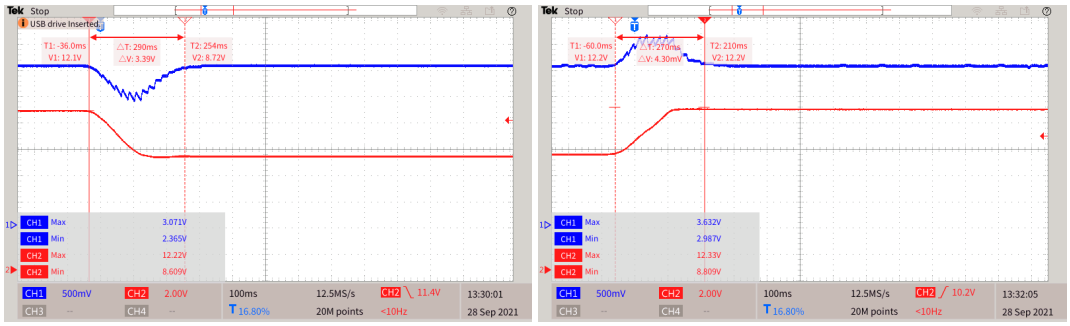


(a) Buck converter output voltage controller 3V step response

(b) Buck converter output voltage controller 10V step response

Figure 5.4: Buck converter output voltage controller step response for full output voltage range

The controller response to supply voltage changes was evaluated by allowing the controller to achieve a steady state, and then varying the input supply between values of 6V and 16V respectively. From these tests it was observed that the controller was consistently able to return the output to it's steady state within a period of 250ms. It was also observed that the designed converter was capable of providing any specified output between 1V and 10V across the full tested supply range, so long as the requested output was at least 500mV below the current supply.



(a) Buck converter output voltage controller response to -4V supply transition

(b) Buck converter output voltage controller response to 4V supply transition

Figure 5.5: Buck converter output voltage controller response to varied input supply

5.3.1 Control Subsystem Evaluation

The control subsystem was required to ensure that the buck converters output voltage steady state error was no larger than $\pm 5\%$ across the output range of 3V to 10V. The performed tests show that the implemented sub system meets these specifications. This subsystem has been proven to provide an output voltage steady state error of $\pm 0.78\%$ across it's full output range, while also providing additional functionality of input supply rejection. This shifts the operating input supply range of all subsystems within this project to between 8V and 16V, as opposed to the static 12V initially specified in requirement 1.
For the full range of testing, measurements, and plots, refer too Appendix D.

Chapter 6: Conclusions

The ultimate goal of this project was to develop a testing platform through which the effects of variable buck converter switching frequency on inductor current ripple could be observed and controlled. This goal was to be approached with an emphasis on its usability and expandability, providing a proof of concept for further development and research to be performed.

With these goals in mind the project was subdivided into three subsystems, each responsible for the operation of a key component of the final systems design. Each subsystem was individually designed, implemented, and evaluated throughout this report.
The first of these subsystem is PWM signal generation, responsible for providing a selectable switching frequency and duty cycle switching signal. This section facilitates buck converter operation, providing the ability to both vary output voltage, and inductor ripple independently.
The second subsystem involves system state sensing, responsible for measuring the current values of various portions of the design. This section facilitates the feedback on current converter states, monitoring the output voltage, the supply voltage, and the inductor current ripple.
The final subsystem provides the control, responsible for maintaining the selected buck converter output voltage and inductor current ripple. This subsystems interacts directly with the previous two, receiving system feedback from subsection 2, and providing output frequencies and duty cycles to subsystem 1.

This project was also subject to a list of requirements, provided in Section 1.2, which was then broken down into final system specifications in Section 3.1. The following list briefly outlines each requirement and it's final implementation within this project.

1. **Operate with a 12V DC input supply voltage**

   Each subsystem was successfully designed to operate with a DC input supply between 8V and 16V.

2. **Provide a selectable DC output voltage between 3V and 10V**

   The implemented system was capable maintaining a selected DC output voltage between 1V and 10V, selectable to one decimal pace.

3. **Provide an output voltage precision of at least $\pm 5\%$ of the targeted output voltage**

   The implemented system was capable maintaining an output steady state error of $\pm 0.78\%$, across the entire output voltage range.

4. **Provide a selectable inductor current ripple between 20% and 50% of the total output current**

   This requirement is pending implementation due to Covid-19 relates delays. Evaluation has been performed on each element of the requirements design, all of which successfully met the individual specifications.

5. **Provide a variable converter switching frequency between 1kHz and 100kHz**

   The implemented system was capable of varying switching frequency between 1kHz, and 100kHz, with a maximum frequency error of 0.064% of the target output.

6. **Provide an inductor current ripple precision of at least $\pm$5% of the target inductor current ripple**

   This requirement is pending implementation due to Covid-19 relates delays. Evaluation has been performed on each element of the requirements design, all of which successfully met the individual specifications.

7. **Operate with variable load sizes between 10Ω and 20 Ω**

   Each subsystem has been successfully designed to operate with variable load sizes between 10Ω and 20 Ω.

6.1 Future Work

Upon full assembly of this projects subsystems into the final designed system, this project lends it self to a wide variety of future research and study.
With the projects motivation being based on the complexity and static nature of buck converter design, research into the characterisation of varying buck converter topologies can be performed. This could allow for distinctions between the design process of synchronous and asynchronous buck converters, exploring possible differences in switching frequency requirements between the two topologies.
It would also be valuable to explore this systems response to non-resistive loads, looking to characterise the different switching requirements for both inductive and capacitive impedances.

Bibliography

[1] E. Earley, "What's the difference between ac and dc?." Online, Sept. 2013.

[2] G. Bocock, "History of switch mode power supplies (smps)." Online.

[3] J. R. Roman, "PWM regulator with varying operating frequency for reduced EMI," Mar. 2001.

[4] Y. L. Familiant and A. Ruderman, "A Variable Switching Frequency PWM Technique for Induction Motor Drive to Spread Acoustic Noise Spectrum With Reduced Current Ripple," *IEEE transactions on industry applications*, vol. 52, no. 6, pp. 5355–5355, 2016.

[5] M. Thoren and C. Steward, "Accurate, fast settling analog voltages from digital pwm signals," techreport, Analog Devices.

[6] J. Caldwell, *Analog Pulse Width Modulation*. texas instruments, June 2013.

[7] S. Colley, "Pulse-width modulation (pwm) timers in microcontrollers." Online, Feb. 2020.

[8] N. Mohan, *Power Electronics: A First Course*. Don Fowley, Oct. 2011.

[9] N. Mohan, *Power Electronics a First Course*, ch. 4: Switch Mode DC-DC Converters: Switching Analysis, Topology Selection and Design, pp. 38–68. Don Fowley, 2012.

[10] B. Hauke, "Basic Calculation of a Buck Converter's Power Stage," tech. rep., Texas Instruments, Aug. 2015.

[11] P. Jain, "Engineers garage - current sensors." Online, June 2012.

[12] Texas Instruments, *Precision Magnetics – Solving the Stability Challenge in Isolated Current Sensing*, Aug. 2020.

[13] Espressif Systems, *ESP32: Technical Reference Manual*, 4.4 ed., 2021.

[14] Analog Devices, *Bidirectional Current Sense Amplifier - LT1999-50*, Sept. 2019.

[15] R. Elliott, "Peak detection circuits." Online, Mar. 2017.

[16] H. Kelley and G. Alonso, "Ltc6244 high speed peak detector," techreport, Analog Devices, 2020.

[17] M. Patil and P. Rodey, *Control Systems for Power Electronics - A Practical Guide*, ch. Chapter 5 - Buck Converter in Closed Loop, pp. 29–40. SpringerBriefs in Applied Sciences and Technology, Springer India, 1 ed., 2015.

Acknowledgments

The author would like to thank Dr Daniel Burmester for his continued help, support and encouragement throughout the course of this project.
The author would also like to thank the ECS VUW technical staff and academics.

Appendix A: System Specification Derivation

PWM Duty Cycle Resolution Calculations

It has been specified that the output voltage error should be within $\pm 5\%$ of the output voltage. This allowable error will be smallest with the smallest output voltage, and will therefore define the smallest allowable duty cycle resolution.

Calculate the allowable output voltage error:

$$\begin{aligned} V_{error} &= 3 \cdot 0.05 \\ &= 0.15V \end{aligned} \tag{A.1}$$

Calculate the minimum number of discrete voltage steps based on Equation (2.1)

$$\begin{aligned} N_{steps} &= \frac{V_{in}}{V_{error}} = \frac{12}{0.15} \\ &= 80 \end{aligned} \tag{A.2}$$

Calculate the resolution in bits:

$$\begin{aligned} N_{bits} &= \log_2(N_{steps}) = \log_2(80) \\ &= 6.3 \approx 7 \end{aligned} \tag{A.3}$$

This gives a minimum duty cycle resolution of 7 bits, or 128 discrete steps.

Minimum & Maximum Inductor Size Calculations

It has been specified that the operating frequency of the buck converter should be between $1k$Hz & $100k$Hz, with an inductor current ripple between 20% & 50%. Based on these specifications, Equation (2.2) can be used to derive the minimum and maximum inductor sizes that the system will continue to function with.

Derivation Constants:

$R_{load} = 10\Omega$, $V_{in} = 12V$, $V_{max} = 10V$, $V_{min} = 3V$, $f_{min} = 1$kHz, $f_{max} = 100$kHz

All calculations will be performed using the minimum allowable inductor current ripple of 20%, calculated below. This will ensure that all greater current ripple percentages are achievable by the system.

$$i_{min} = \frac{V_{min}}{R_{load}} \tag{A.4}$$
$$= 0.3A$$

$$\Delta i_{min} = i_{min} \cdot 0.2 \tag{A.5}$$
$$= 0.06A$$

Minimum Inductor Size

Using Equation (A.6), we see that the minimum achievable indictor size will be at the maximum switching frequency. It can be noted that due to the quadratic nature of this equation, the maximum switching frequency for a constant ripple will occur at an output of $V_o = \frac{v_{in}}{2}$, with a duty cycle of 50%.

$$L = \frac{V_o \cdot (1 - D)}{\Delta i_L \cdot f_s} \tag{A.6}$$

Calculate the minimum inductor value:

$$L_{min} = \frac{\frac{v_{in}}{2} \cdot (1 - 0.5)}{\Delta i_{min} \cdot f_{max}} \tag{A.7}$$

$$= \frac{6 \cdot (1 - 0.5)}{100000 \cdot 0.06}$$
$$= 0.5mH$$

Maximum Inductor Size

From Equation (A.6), we can see that the maximum inductor value will be at the lowest switching frequency, with the greatest output voltage, and therefore the greatest duty cycle.

Calculate the maximum duty cycle:

$$D_{max} = \frac{V_{max}}{V_{in}} \tag{A.8}$$
$$= \frac{10}{12} = 83.3\%$$

Calculate the maximum inductor size:

$$L_{min} = \frac{V_{max} \cdot (1 - D_{max})}{\Delta i_{min} \cdot f_{min}} \tag{A.9}$$

$$= \frac{1 \cdot (1 - 0.83)}{1000 \cdot 0.06}$$
$$= 27.78mH$$

PWM Frequency Resolution Calculations

I need to put this here from desmos: https://www.desmos.com/calculator/oscpnz2ozn

Peak Inductor Current Ripple Sensing Resolution

The sensing resolution must be smaller or equal to the tolerable error of the inductor current ripple.

$$I_{min} = \frac{V_{min}}{R_{load}} = 0.3A \tag{A.10}$$

$$I_{min_{ripple}} = I_{min} * 0.2 = 0.06A \tag{A.11}$$

$$I_{min_{error}} = I_{min} * 0.25 = 0.075A \tag{A.12}$$

$$I_{error} = I_{min_{error}} - I_{min_{ripple}} \tag{A.13}$$

$$= 15mA$$

Appendix B: PWM Generation Figures

Analogue PWM Generation

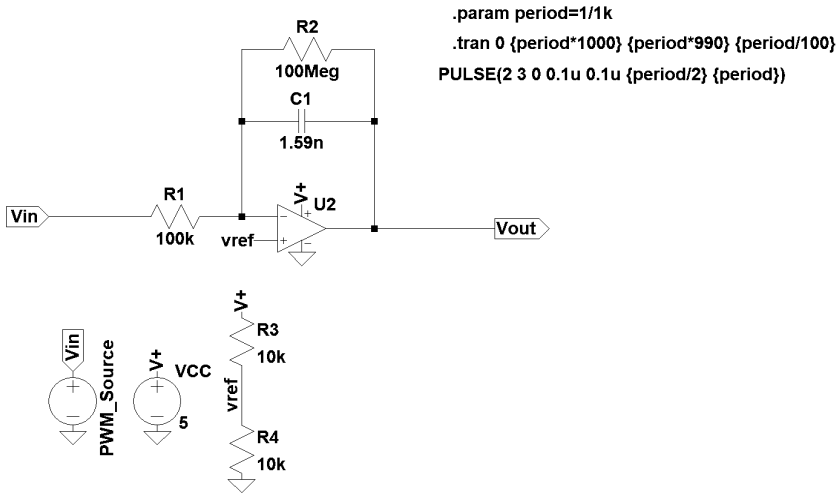


Figure B.1: Operational amplifier integrator LTSpice circuit simulation

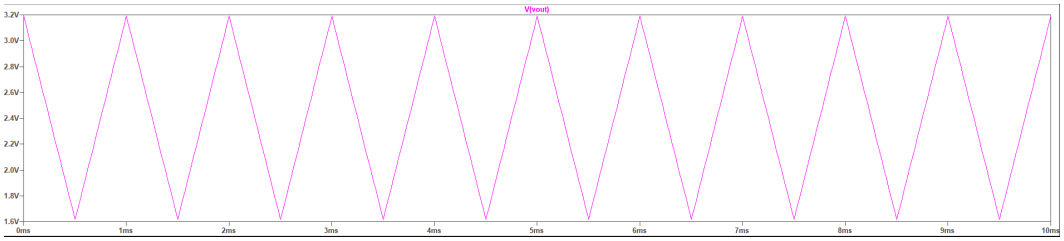


Figure B.2: Operational amplifier integrator simulation with a $1kHz$ square wave input. Output triangle wave amplitude of 1.5V

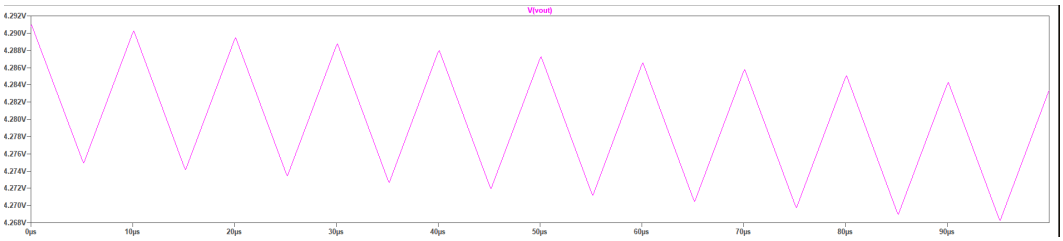
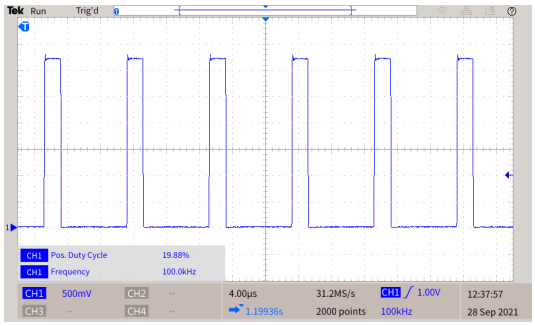


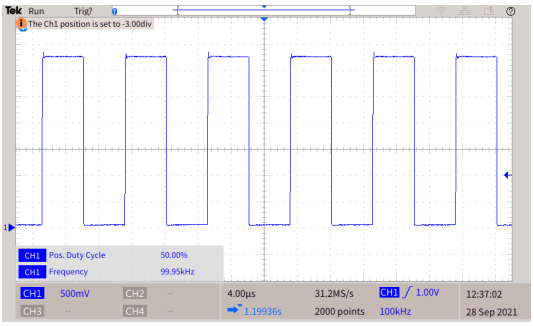
Figure B.3: Operational amplifier integrator simulation with a $1kHz$ square wave input. Output triangle wave amplitude of 20mV

Digital PWM Generation

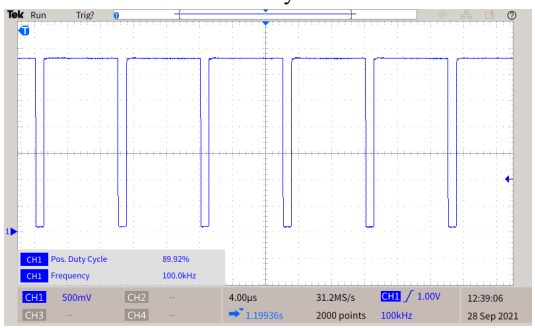
PWM Duty Cycle Control



(a) Digital PWM Generation with a 20% duty cycle



(b) Digital PWM Generation with a 50% duty cycle



(c) Digital PWM Generation with a 90% duty cycle

Figure B.4: Digital PWM: Varying PWM duty cycle with a 100kHz frequency

PWM Duty Cycle Output range

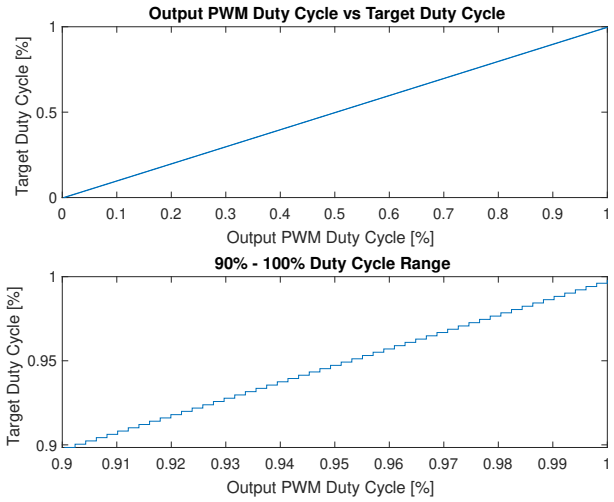
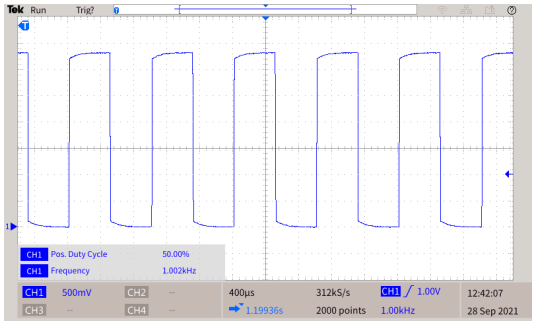
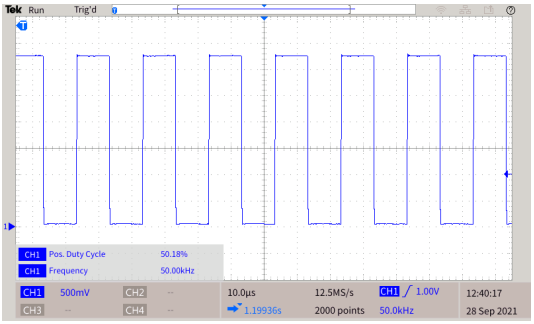


Figure B.5: Implemented PWM generator full duty cycle selection range

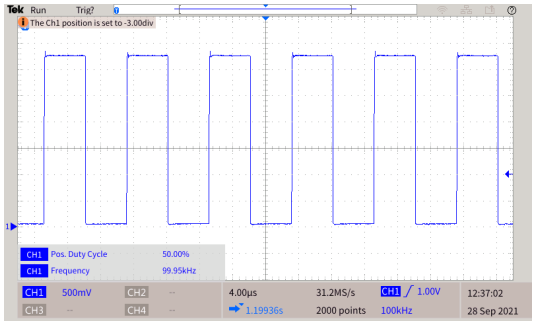
PWM Frequency Control



(a) Digital PWM Generation with a $1kHz$ frequency



(b) Digital PWM Generation with a $50kHz$ frequency



(c) Digital PWM Generation with a $100kHz$ frequency

Figure B.6: Digital PWM: Varying PWM frequency with a 50% duty cycle

Appendix C: Peak Detector Figures & Tables

Precision Rectifier and Sample & Hold Peak Detector Design Simulations

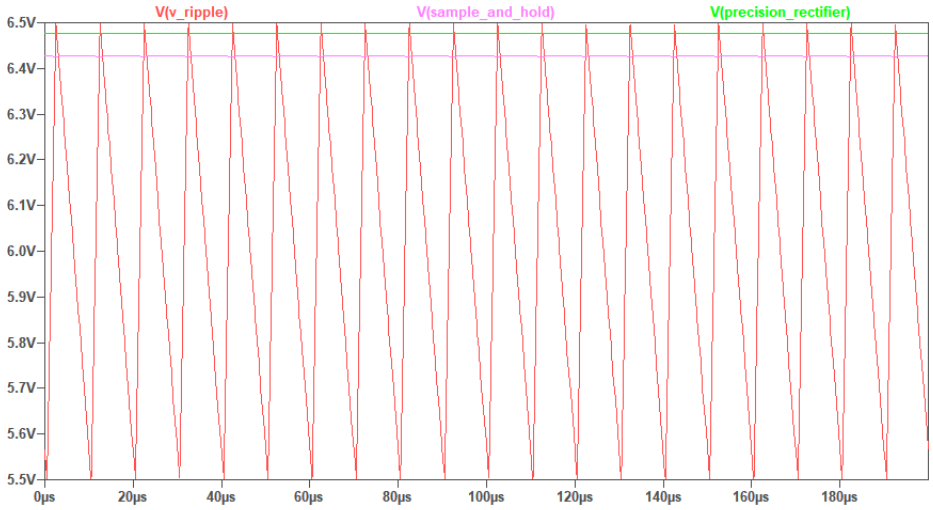
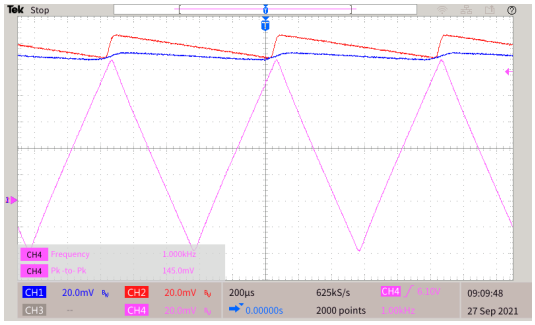


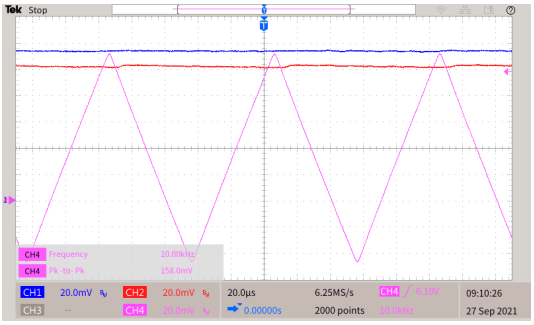
Figure C.1: LTSpice circuit simulation of the precision rectifier and sample and hold peak detection designs.

Initial & Final Peak Detector Output Comparison

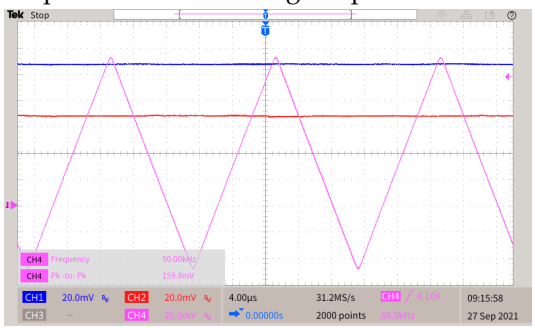
Peak Detector Output for a 150mV Peak to Peak Triangle Input



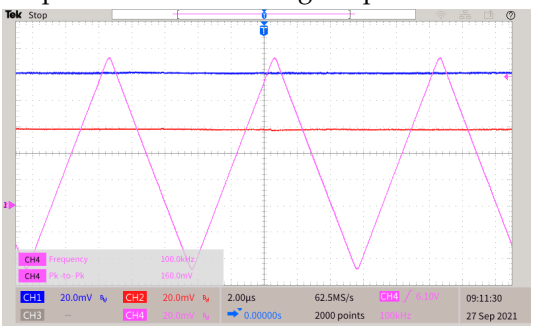
(a) Initial (Red) & final (Blue) peak detetor outputs for a $1kHz$ triangle input.



(b) Initial (Red) & final (Blue) peak detetor outputs for a $10kHz$ triangle input.



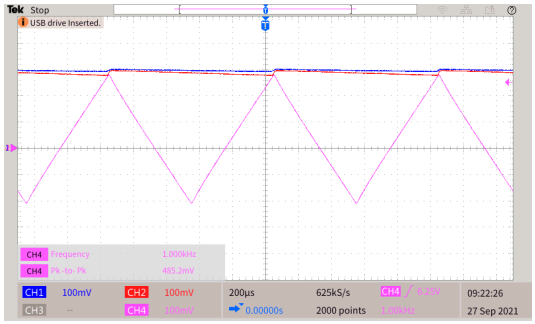
(c) Initial (Red) & final (Blue) peak detetor outputs for a $50kHz$ triangle input.



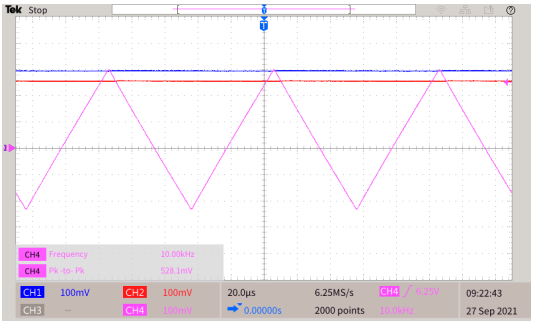
(d) Initial (Red) & final (Blue) peak detetor outputs for a $100kHz$ triangle input.

Figure C.2: Initial & Final Peak Detector Output at varying frequencies for a 150mV peak to peak triangle input.

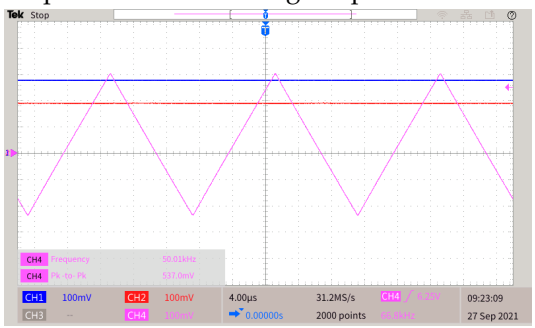
Peak Detector Output for a 500mV Peak to Peak Triangle Input



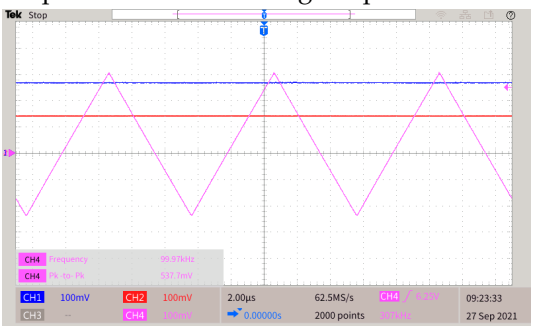
(a) Initial (Red) & final (Blue) peak detetor outputs for a $1kHz$ triangle input.



(b) Initial (Red) & final (Blue) peak detetor outputs for a $10kHz$ triangle input.



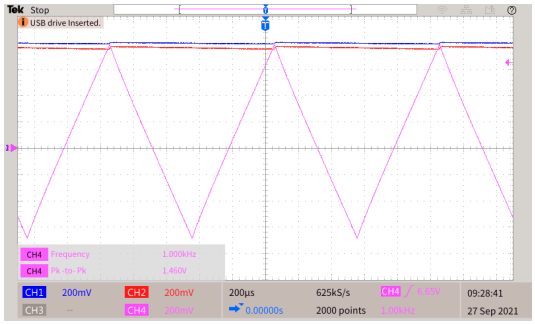
(c) Initial (Red) & final (Blue) peak detetor outputs for a $50kHz$ triangle input.



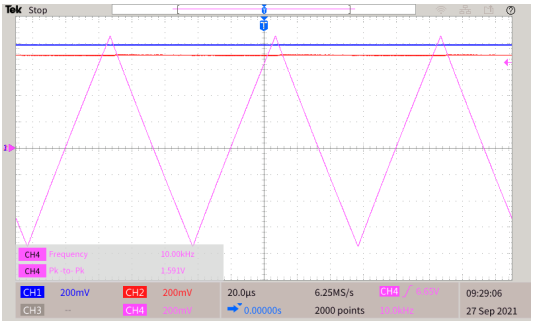
(d) Initial (Red) & final (Blue) peak detetor outputs for a $100kHz$ triangle input.

Figure C.3: Initial & Final Peak Detector Output at varying frequencies for a 500mV peak to peak triangle input.

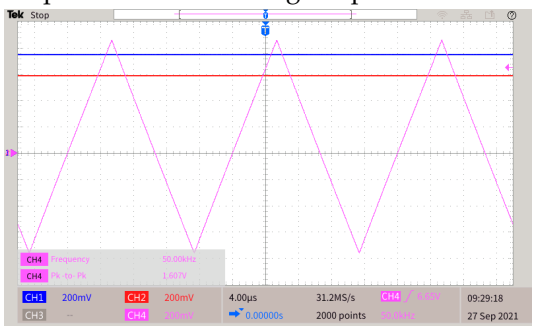
Peak Detector Output for a 1500mV Peak to Peak Triangle Input



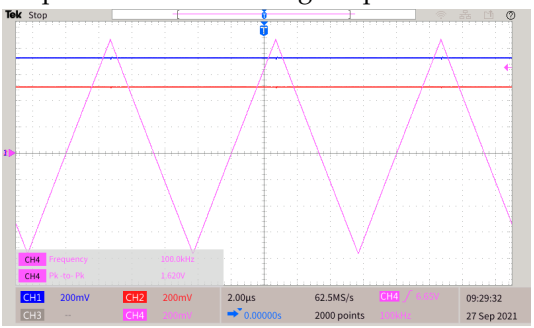
(a) Initial (Red) & final (Blue) peak detetor outputs for a $1kHz$ triangle input.



(b) Initial (Red) & final (Blue) peak detetor outputs for a $10kHz$ triangle input.



(c) Initial (Red) & final (Blue) peak detetor outputs for a $50kHz$ triangle input.



(d) Initial (Red) & final (Blue) peak detetor outputs for a $100kHz$ triangle input.

Figure C.4: Initial & Final Peak Detector Output at varying frequencies for a 1500mV peak to peak triangle input.

Initial & Final Peak Detector Output Error Plots

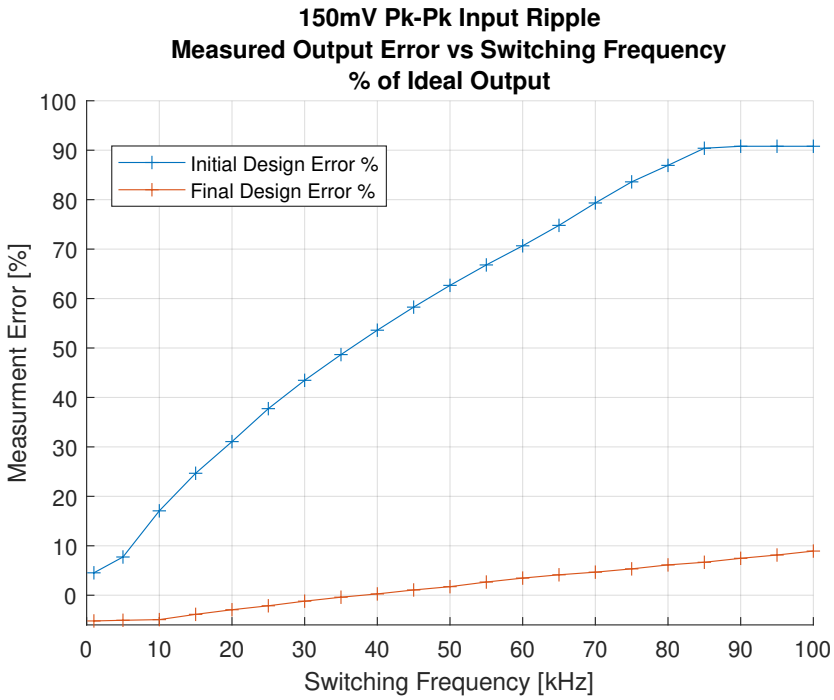


Figure C.5: Initial & final peak detector output error across frequencies for a 150mV peak to peak input. Displayed as a percentage of the ideal output.

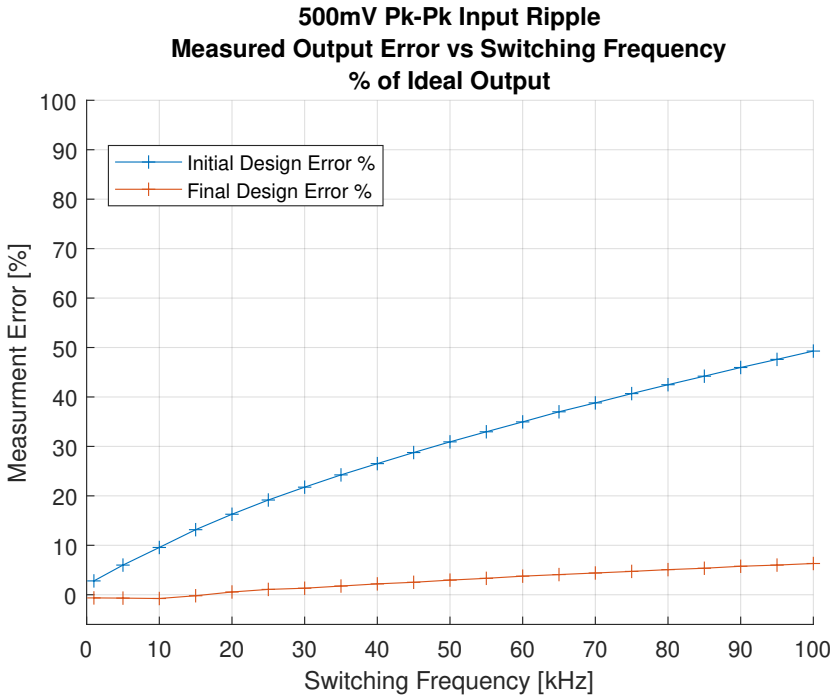


Figure C.6: Initial & final peak detector output error across frequencies for a 500mV peak to peak input. Displayed as a percentage of the ideal output.

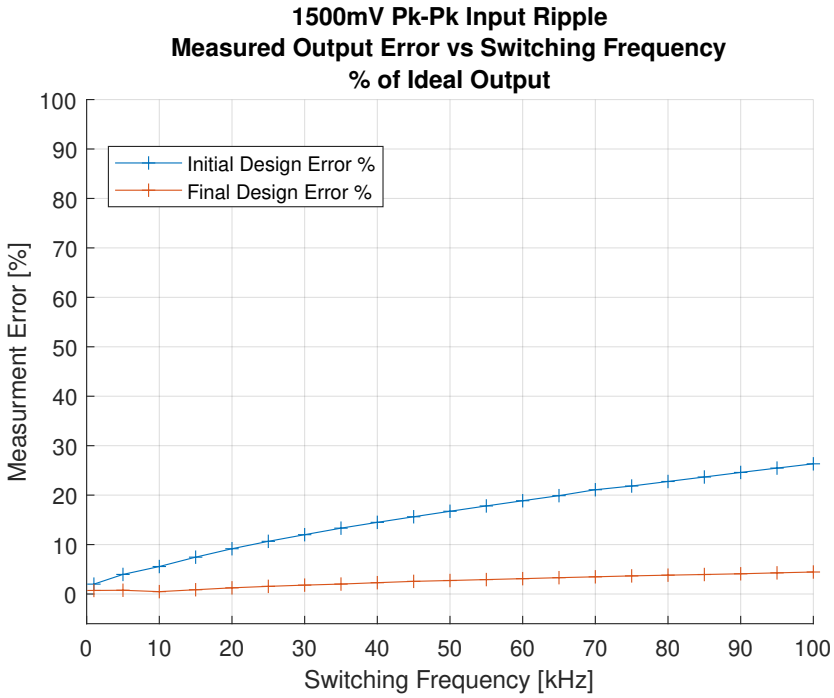


Figure C.7: Initial & final peak detector output error across frequencies for a 1500mV peak to peak input. Displayed as a percentage of the ideal output.

Initial & Final Peak Detector Output Error Tables

150mV Peak to Peak Input Signal Initial and Final Design Peak Detector Output Error Across Frequencies				
Frequency (kHz)	**Final Design Error (mV)**	**Final Design % Error**	**Initial Design Error (mV)**	**Initial Design % Error**
1	-3.90	-5.2	3.40	4.533333333
5	-3.80	-5.066666667	5.80	7.733333333
10	-3.70	-4.933333333	12.80	17.06666667
15	-2.90	-3.866666667	18.50	24.66666667
20	-2.20	-2.933333333	23.30	31.06666667
25	-1.60	-2.133333333	28.30	37.73333333
30	-0.90	-1.2	32.60	43.46666667
35	-0.30	-0.4	36.50	48.66666667
40	0.20	0.266666667	40.20	53.6
45	0.80	1.066666667	43.70	58.26666667
50	1.30	1.733333333	47.00	62.66666667
55	2.00	2.666666667	50.10	66.8
60	2.60	3.466666667	53.00	70.66666667
65	3.10	4.133333333	56.10	74.8
70	3.50	4.666666667	59.50	79.33333333
75	4.00	5.333333333	62.70	83.6
80	4.60	6.133333333	65.20	86.93333333
85	5.00	6.666666667	67.80	90.4
90	5.60	7.466666667	68.10	90.8
95	6.10	8.133333333	68.10	90.8
100	6.70	8.933333333	68.10	90.8

Table C.1: Table of output errors at varying frequencies for both the initial and final peak detection design with a 150mV peak to peak input signal.

500mV Peak to Peak Input Signal Initial and Final Design Peak Detector Output Error Across Frequencies				
Frequency (kHz)	**Final Design Error (mV)**	**Final Design % Error**	**Initial Design Error (mV)**	**Initial Design % Error**
1	-1.60	-0.64	7.00	2.8
5	-1.70	-0.68	15.00	6
10	-1.90	-0.76	23.90	9.56
15	-0.50	-0.2	32.90	13.16
20	1.40	0.56	40.70	16.28
25	2.70	1.08	47.90	19.16
30	3.30	1.32	54.40	21.76
35	4.40	1.76	60.60	24.24
40	5.50	2.2	66.30	26.52
45	6.30	2.52	71.90	28.76
50	7.40	2.96	77.30	30.92
55	8.30	3.32	82.40	32.96
60	9.40	3.76	87.40	34.96
65	10.20	4.08	92.50	37
70	11.00	4.4	97.00	38.8
75	11.80	4.72	101.70	40.68
80	12.70	5.08	106.20	42.48
85	13.40	5.36	110.50	44.2
90	14.40	5.76	114.90	45.96
95	15.00	6	119.00	47.6
100	15.80	6.32	123.20	49.28

Table C.2: Table of output errors at varying frequencies for both the initial and final peak detection design with a 500mV input signal.

1500mV Peak to Peak Input Signal Initial and Final Design Peak Detector Output Error Across Frequencies				
Frequency (kHz)	**Final Design Error (mV)**	**Final Design % Error**	**Initial Design Error (mV)**	**Initial Design % Error**
1	5.40	0.72	15.00	2
5	5.70	0.76	29.70	3.96
10	3.60	0.48	41.70	5.56
15	6.50	0.866666667	55.90	7.453333333
20	9.40	1.253333333	68.70	9.16
25	11.60	1.546666667	79.90	10.65333333
30	13.50	1.8	90.00	12
35	15.10	2.013333333	99.90	13.32
40	17.20	2.293333333	108.70	14.49333333
45	19.20	2.56	117.20	15.62666667
50	20.50	2.733333333	125.50	16.73333333
55	21.90	2.92	133.60	17.81333333
60	23.30	3.106666667	141.50	18.86666667
65	24.80	3.306666667	149.20	19.89333333
70	26.10	3.48	158.20	21.09333333
75	27.50	3.666666667	163.80	21.84
80	28.60	3.813333333	170.70	22.76
85	29.60	3.946666667	177.50	23.66666667
90	30.70	4.093333333	184.40	24.58666667
95	32.10	4.28	191.00	25.46666667
100	33.40	4.453333333	197.50	26.33333333

Table C.3: Table of output errors at varying frequencies for both the initial and final peak detection design with a 1500mV input signal.

Appendix D: Control System Figures

Control System Matlab Simulation

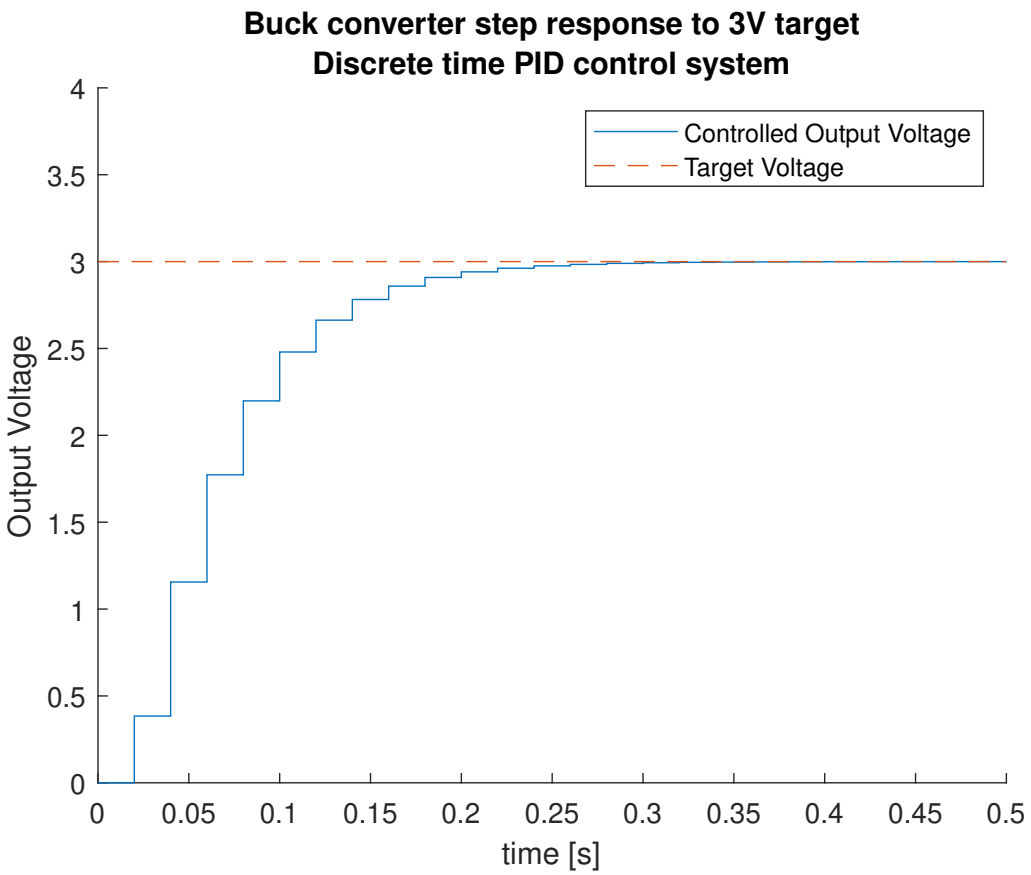
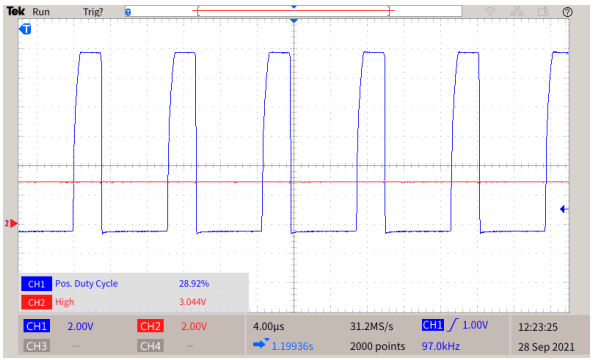
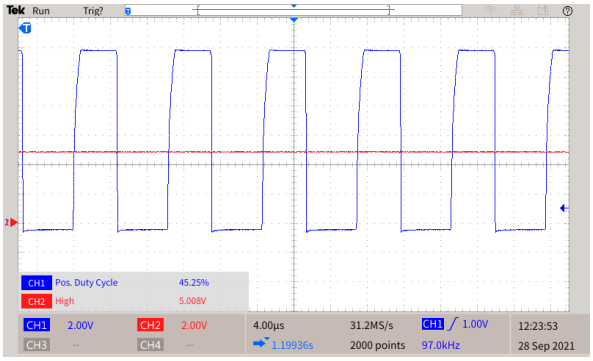


Figure D.1: Matlab simulation of the discrete time controlled buck converter with a 3V step input

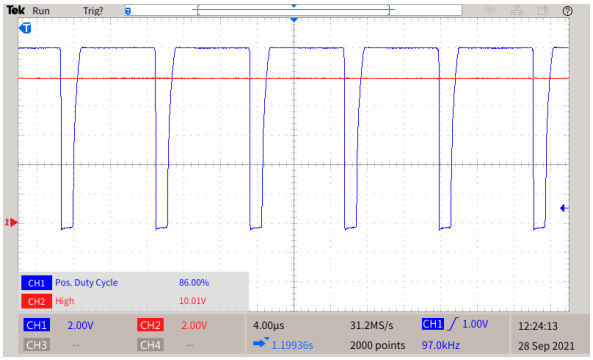
Variable Output Voltage Duty Cycle Control



(a) Controlled duty cycle of 28.9% for a 3V DC output



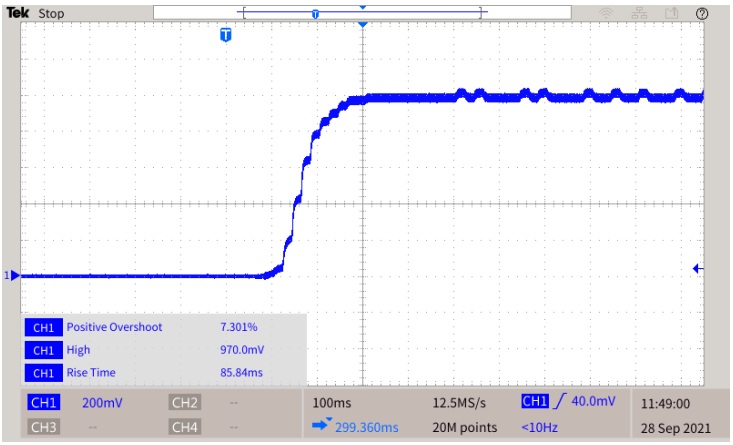
(b) Controlled duty cycle of 45.25% for a 5V DC output



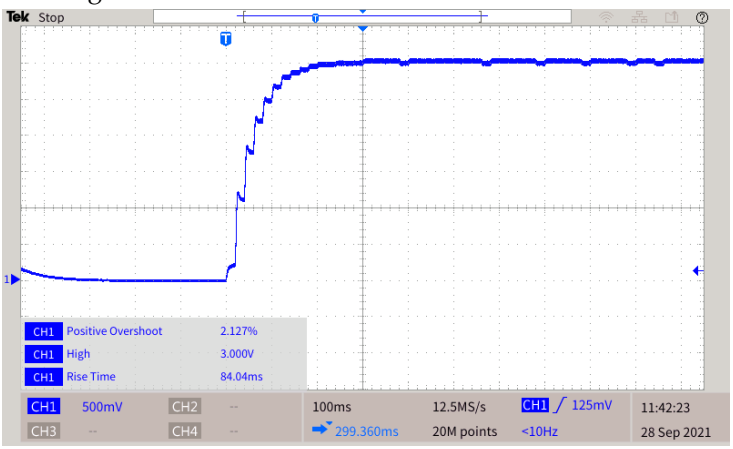
(c) Controlled duty cycle of 86.0% for a 10V DC output

Figure D.2: PWM duty cycle controlled by the voltage output control system (Blue), achieving 3V, 5V and 10V DC (Red).

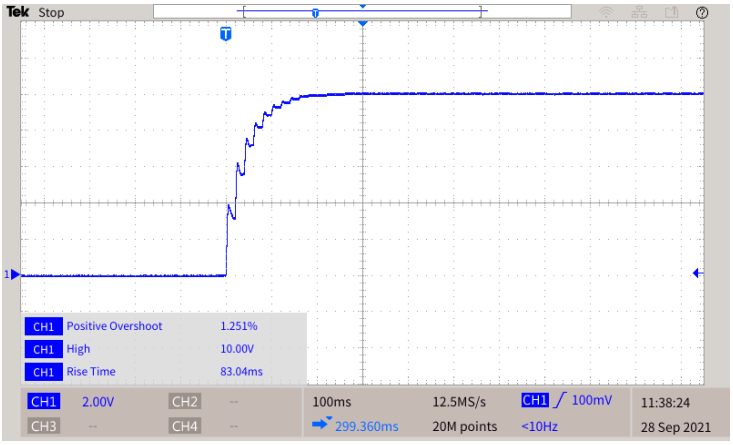
Output Voltage Control System Step Response Rise Time



(a) Output voltage controller 0V to 1V step response with 86ms settleing time.



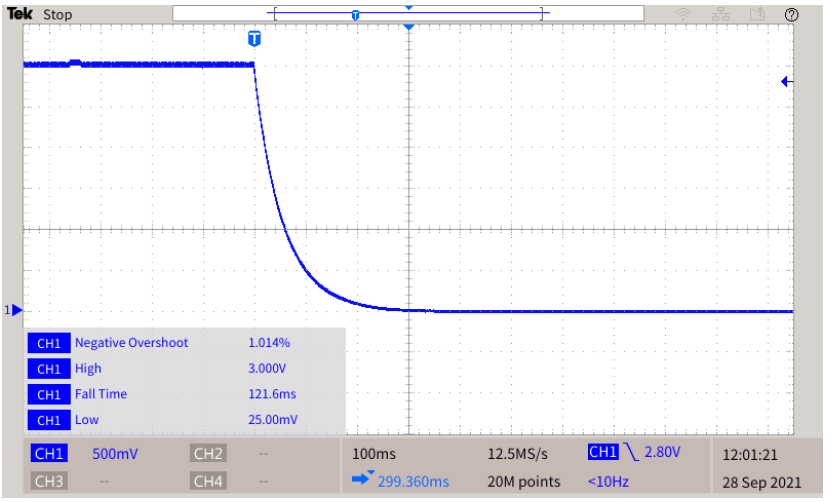
(b) Output voltage controller 0V to 3V step response with 84ms settleing time.



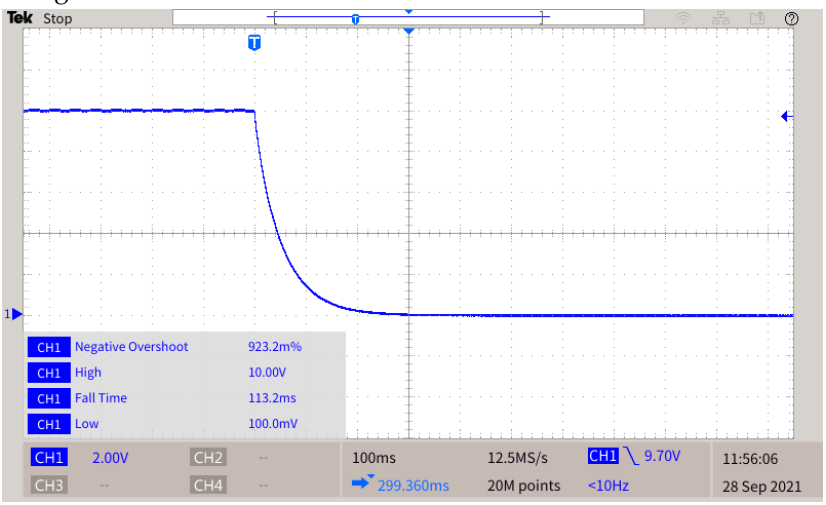
(c) Output voltage controller 0V to 10V step response with 83ms settleing time.

Figure D.3: Output voltage control system step response, reacting to 1V, 3V and 10V output steps.

Output Voltage Control System Step Response Fall Time



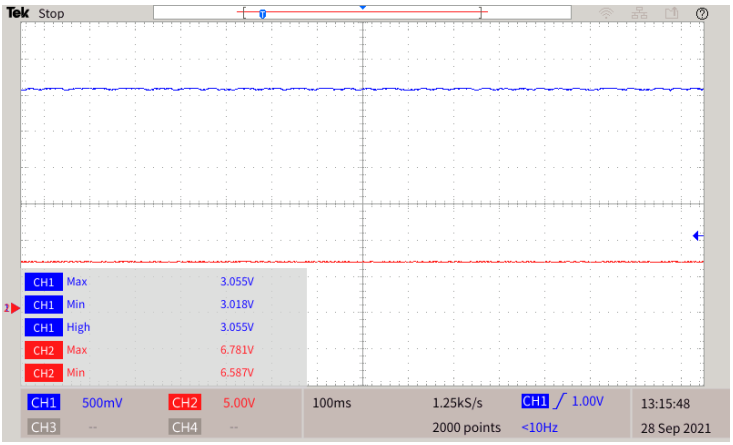
(a) Output voltage controller 3V to 0V step response with 121ms settleing time.



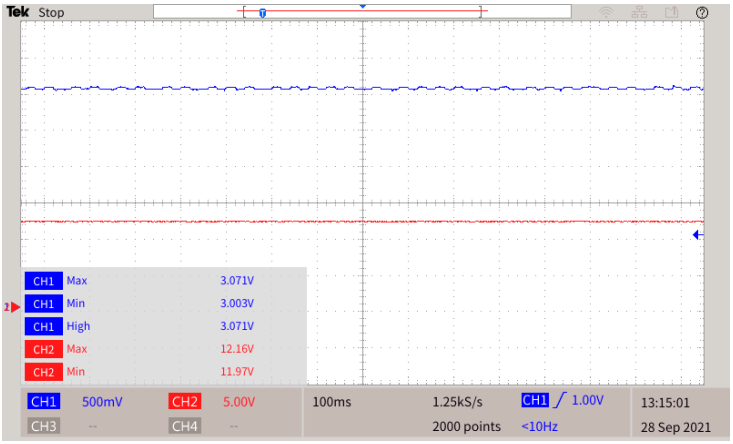
(b) Output voltage controller 10V to 0V step response with 113ms settleing time.

Figure D.4: Output voltage control system step response, reacting to -3V and -10V output steps.

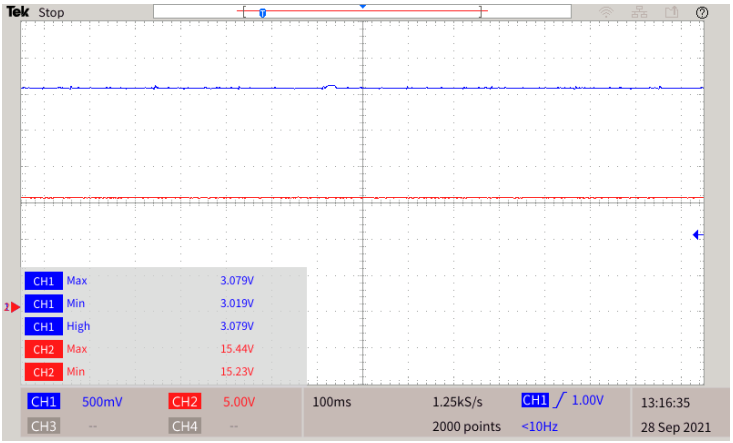
Output Voltage Control System with Variable Supply Voltage



(a) Output voltage controller regulating a 6V supply (Red) for a 3V output (Blue)



(b) Output voltage controller regulating a 12V supply (Red) for a 3V output (Blue)



(c) Output voltage controller regulating a 16V supply (Red) for a 3V output (Blue)

Figure D.5: Output voltage controller regulating variable supply voltages for constant output.

Output Voltage Control System Supply Voltage Step

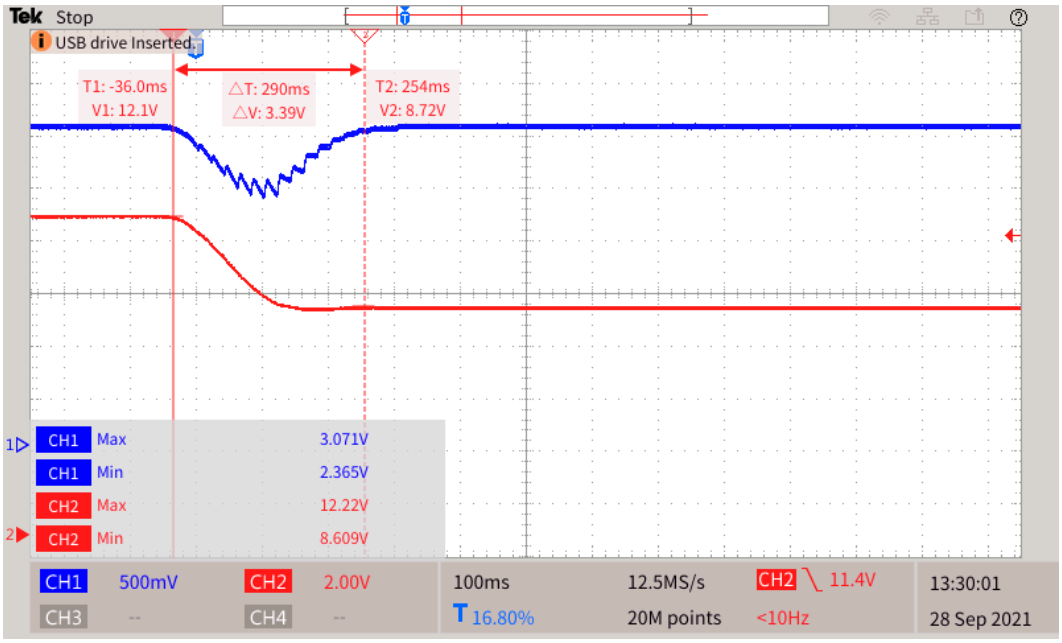


Figure D.6: Output voltage controller response (Blue) to a 12V to 8V supply voltage step (Red), with a recovery time of 290ms

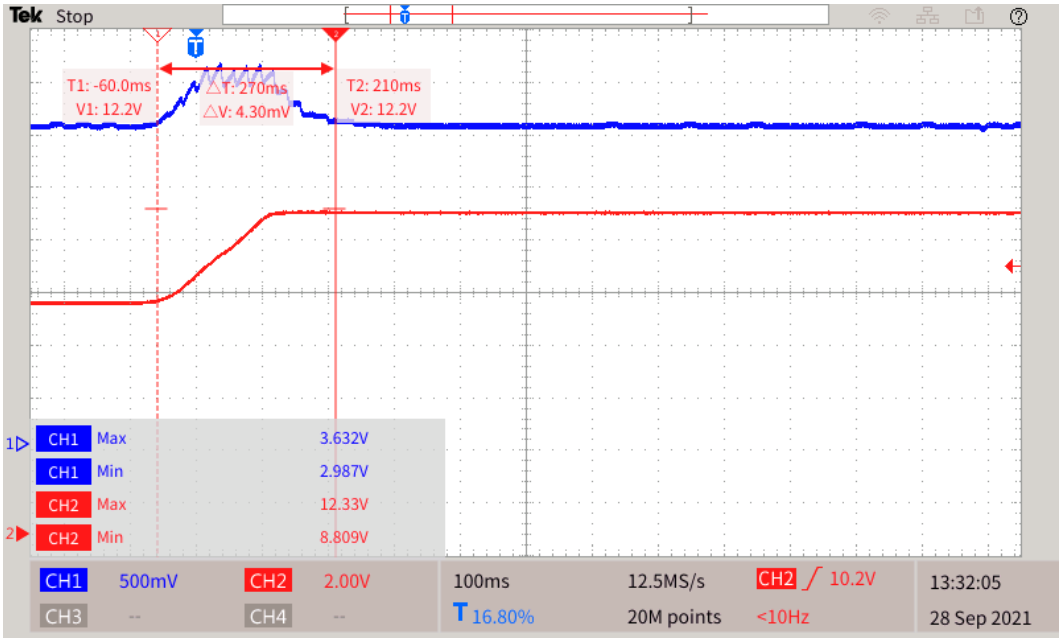


Figure D.7: Output voltage controller response (Blue) for a 8V to 12V supply voltage step (Red), with a recovery time of 270ms

Appendix E: Schematic & Printed Circuit Board

Full System Schematic

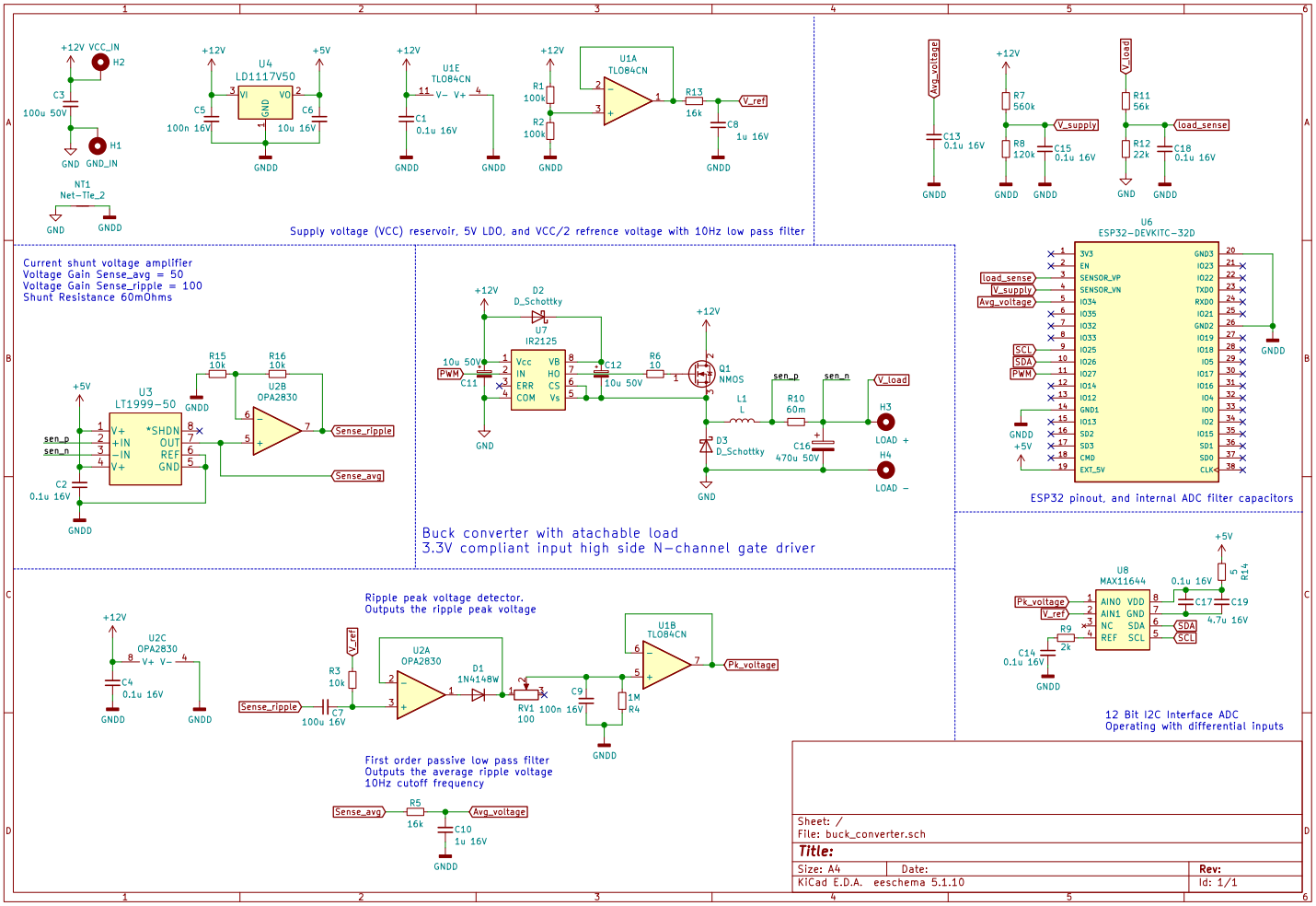


Figure E.1: Final system full schematic

Designed Printed Circuit Board

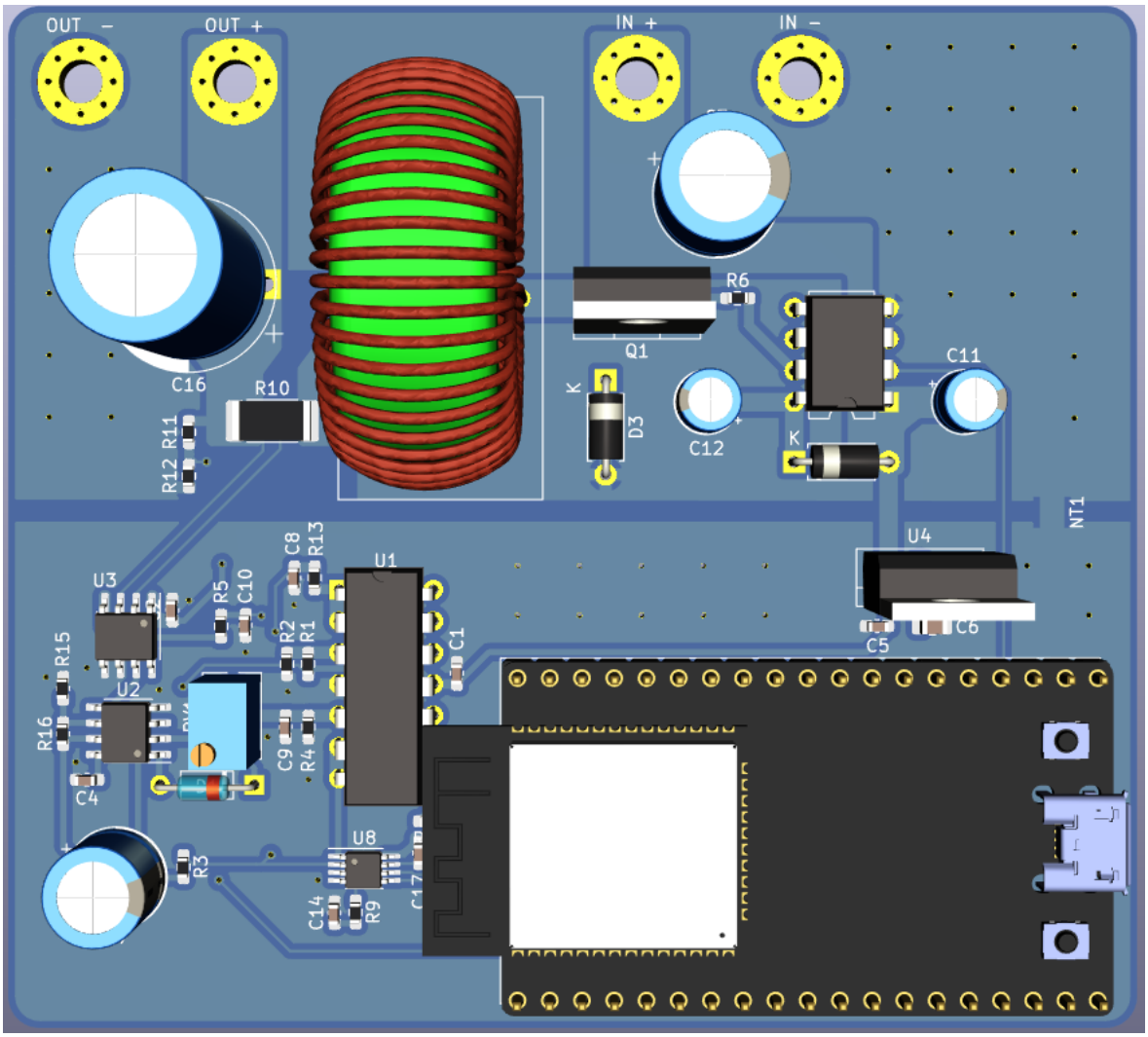


Figure E.2: Final system printed circuit board rendering

Appendix F: Breadboard Implementation

PWM Signal Generator & Control System Breadboard Implementation

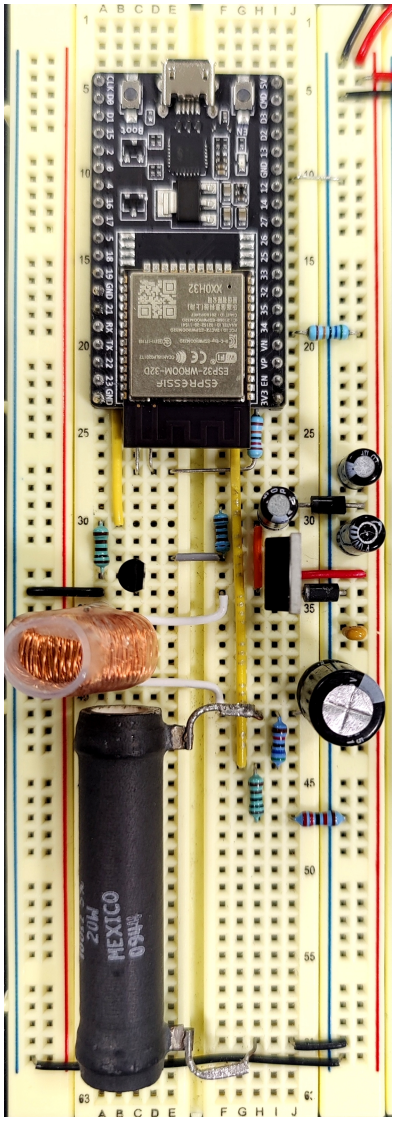


Figure F.1: Photograph of the hardware implementation of both the PWM signal generator, and the feedback measurements for the output voltage control system.

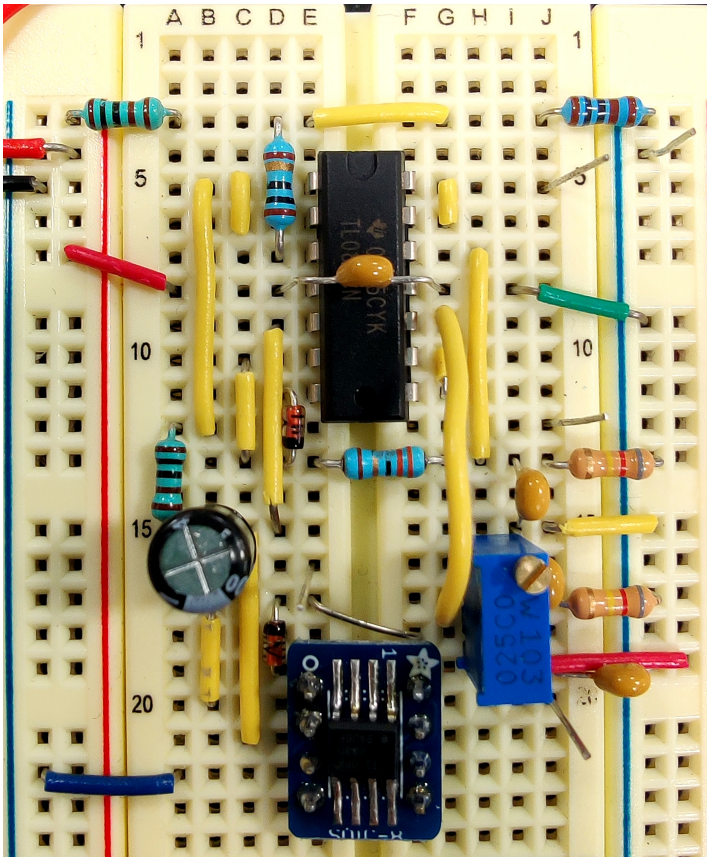


Figure F.2: Photograph of the hardware implementation of the inductor current ripple peak detection circuit.

Appendix G: Source Code

Single Header Libraries: Hardware Drivers

adc.h

```
#ifndef BUCK_CONVERTER_ADC
#define BUCK_CONVERTER_ADC

#include <math.h>
#include "driver/adc.h" // Include the ADC driver

// ADC struct to hold setup data
typedef struct esp_adc
{
    // Rolling average buffer
    uint16_t *buffer;
    uint8_t span;

    // ADC channel to sample
    uint8_t adc_channel;

} esp_adc;

#endif // BUCK_CONVERTER_ADC


#ifdef BUCK_CONVERTER_ADC_IMPL

/*
 * Set up the adc to have a 12bit width, and 11dB attenuation
 */
esp_err_t adc_init(esp_adc* adc){

    // ADC set up
    esp_err_t status; // Status variable to check if adc initialisation was
    successful

    status = adc1_config_width(ADC_WIDTH_BIT_12); // Set adc width to 12 bits
    if(status != ESP_OK) return status;

    status = adc1_config_channel_atten(adc->adc_channel, ADC_ATTEN_11db); // Set
    adc attenuation to 11dB
    return status;
}


```

```
40 /*
41  * Read the raw value from the adc
42  */
43 uint16_t adc_read(esp_adc* adc){
44     // Read the raw value from the provided adc channel
45     return adc1_get_raw(adc->adc_channel);
46 }
47
48 /* Compute the rolling average of the raw ADC values.
49  * The circular buffer will store the index of the next element to be
50  * replaced in it's final index.
51  */
52 float rolling_average(esp_adc* adc, int raw_value){
53
54     // Check if there is an averaging buffer
55     if(adc->span == 0)
56     {
57         printf("Can not compute average with a span of 0\n");
58         return -1;
59     }
60
61     // get the index of the value to replace, and then replace it
62     int next_index = adc->buffer[adc->span];
63     adc->buffer[next_index] = raw_value;
64
65     // Increment the index of the last value and check if it past the end of the
       array
66     next_index++;
67     adc->buffer[adc->span] = (next_index >= adc->span) ? 0 : next_index;
68
69     // Calculate the rolling average
70     float total = 0;
71     for(int i = 0; i < adc->span; i++)
72     {
73         total = total + adc->buffer[i];
74     }
75
76     return total/adc->span;
77 }
78
79 /* Compute the conversion from the adc reading to a voltage.
80  *
81  * The output of the adc has been characterised and then a polynomial has been fit
       to the curve.
82  * This provides an error of < 1% for readings between 200mV and 3100mV
83  */
84  float IRAM_ATTR adc_conversion(float acd_reading)
85 {
86     return (pow(acd_reading , 4) * -7.6813211494455e-15) +
87            (pow(acd_reading , 3) *  5.03088719249885e-11) +
88            (pow(acd_reading , 2) * -1.06609443189713e-7) +
89            (0.00085850726668 * acd_reading) + 0.09077205072441;
90 }
91
92 #endif
```

code/adc.h

pid.h

```
#ifndef BUCK_CONVERTER_PID
#define BUCK_CONVERTER_PID

/*
 * PID Controller code structure designed by https://github.com/pms67 under the MIT licence
 * https://github.com/pms67/PID
 */
typedef struct
{

    /* Controller gains */
    float Kp;
    float Ki;
    float Kd;

    /* Derivative low-pass filter time constant */
    float tau;

    /* Output limits */
    float limMin;
    float limMax;

    /* Integrator limits */
    float limMinInt;
    float limMaxInt;

    /* Sample time (in seconds) */
    float T;

    /* Controller "memory" */
    float integrator;
    float prevError; /* Required for integrator */
    float differentiator;
    float prevMeasurement; /* Required for differentiator */

    /* Controller output */
    float out;

} PIDController;

#endif


#ifdef BUCK_CONVERTER_PID_IMPL

void PID_init(PIDController *pid){

    /* Clear controller variables */
    pid->integrator = 0.0f;
    pid->prevError = 0.0f;

    pid->differentiator = 0.0f;
    pid->prevMeasurement = 0.0f;
```

```
54
55      pid->out = 0.0f;
56  }
57
58
59  void IRAM_ATTR PID_update(PIDController *pid, float setpoint, float measurement){
60
61      // Error Signal
62      float error = setpoint - measurement;
63
64      // Proportional Signal
65      float proportional = pid->Kp * error;
66
67      // Integral Signal
68      pid->integrator = pid->integrator + (pid->Ki * pid->T * (error +
        pid->prevError) * 0.5f);
69
70      /* Anti-wind-up via integrator clamping */
71      if (pid->integrator > pid->limMaxInt){
72          pid->integrator = pid->limMaxInt;
73      }
74
75      else if (pid->integrator < pid->limMinInt){
76          pid->integrator = pid->limMinInt;
77      }
78
79      // Derivative Signal with low pass filtering
80      pid->differentiator = -(2.0f * pid->Kd * (measurement - pid->prevMeasurement)
        /* Note: derivative on measurement, therefore minus sign in front of equation!
        */
81                              +(2.0f * pid->tau - pid->T) * pid->differentiator) /
82                               (2.0f * pid->tau + pid->T);
83
84      /*
85    * Compute output and apply limits
86    */
87      pid->out = proportional + pid->integrator + pid->differentiator;
88
89      if (pid->out > pid->limMax){
90          pid->out = pid->limMax;
91      }
92
93      else if (pid->out < pid->limMin){
94          pid->out = pid->limMin;
95      }
96
97      /* Store error and measurement for later use */
98      pid->prevError = error;
99      pid->prevMeasurement = measurement;
100 }
101
102 #endif //BUCK_CONVERTER_PID
```

code/pid.h

pwm.h

```
#ifndef BUCK_CONVERTER_PWM
#define BUCK_CONVERTER_PWM

// Hardware driver includes
#include "driver/ledc.h"

/*
 * PWM set up defines and helper macros
 */
#define DUTY_RESOLUTION LEDC_TIMER_9_BIT
#define FREQUENCY_MIN   1000
#define FREQUENCY_MAX   100000
#define DUTY_STEPS      512
#define DUTY(X)         ((1.0 - X) * DUTY_STEPS) // Duty cycle calculation macro

#endif


#ifdef BUCK_CONVERTER_PWM_IMPL

/*
 *   Set up the PWM timer and channel structs, and configure the PWM output
 */

void PWM_setup(ledc_timer_config_t *pwm_timer, ledc_channel_config_t *pwm_channel,
    uint32_t frequency, float duty_cycle){

    // Select and set up the PWM clock source and initial frequency
    ledc_timer_config_t timer = {
        .duty_resolution = DUTY_RESOLUTION,    // resolution of PWM duty
        .freq_hz = frequency,                  // frequency of PWM signal
        .speed_mode = LEDC_HIGH_SPEED_MODE,    // timer mode
        .timer_num = LEDC_TIMER_0,             // timer index
        .clk_cfg = LEDC_AUTO_CLK,              // Auto select the source clock
    };

    // Select and set up the PWM output channel and initial duty cycle
    ledc_channel_config_t channel = {
        .channel    = LEDC_CHANNEL_0,          // PWM output channel (0 - 4)
        .duty       = DUTY(duty_cycle),        // PWM duty cycle
        .gpio_num   = 23,                      // PWM output GPIO
        .speed_mode = LEDC_HIGH_SPEED_MODE,    // PWM output mode
        .timer_sel  = LEDC_TIMER_0             // PWM timer index
    };

    *pwm_timer = timer;
    *pwm_channel = channel;

    ledc_timer_config(pwm_timer);
    ledc_channel_config(pwm_channel);
}

/*
 *   Set a new PWM duty cycle for a given PWM channel
```

```
 */

esp_err_t PWM_set_duty(ledc_channel_config_t *pwm_channel, float duty_cycle){

    // Set the new PWM Duty Cycle of the given channel configuration
    ledc_set_duty(pwm_channel -> speed_mode, pwm_channel -> channel,
    DUTY(duty_cycle));

    // Update the PWM channel with the new configuration
    return ledc_update_duty(pwm_channel -> speed_mode, pwm_channel ->channel);
}


/*
 *   Set a new PWM frequency for a given PWM channel
 */

esp_err_t PWM_set_frequency(ledc_channel_config_t *pwm_channel, ledc_timer_config_t
    *pwm_timer, uint32_t frequency){

    if(frequency > FREQUENCY_MAX || frequency < FREQUENCY_MIN)
    {
        printf("Specified input frequency not within possible range: %d - %d\n",
    FREQUENCY_MIN, FREQUENCY_MAX);
        return ESP_ERR_INVALID_ARG;
    }

    // Set the new PWM frequency of the given channel and timer configuration
    return ledc_set_freq(pwm_channel -> speed_mode, pwm_timer -> timer_num,
    frequency);
}

#endif // BUCK_CONVERTER_PWM
```

code/pwm.h

Application Code

buck_converter.h

```
#ifndef BUCK_CONVERTER
#define BUCK_CONVERTER

// Project includes
#include "pwm.h"
#include "adc.h"
#include "pid.h"

// Standard includes for math and print functions
#include <stdio.h>
#include <math.h>
#include <string.h>

// RTOS includes for error messages and task handling
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/queue.h"
#include "esp_err.h"

void init_buck();
void control_loop(void *pvParameters);

/*
 * PWM timer and channel structs
 */
ledc_timer_config_t pwm_timer;     // Create the PWM timer struct
ledc_channel_config_t pwm_channel; // Create the PWM channel struct

/*
 * Constants for ADC conversion
 */
static const float LOAD_R1 = 56237.0f; // Voltage divider values from buck output
    load
static const float LOAD_R2 = 22035.0f + 995.2f;

static const float SUPPLY_R1 = 565300.0f; // Voltage divider values from buck
    output load
static const float SUPPLY_R2 = 119700.0f;

// Internal esp ADC structs
esp_adc v_supply;

/*
 * Constants, functions, and variables for PID controller
 */
static const float VO = 3.0f;  // The desired initial output of the converter.

// Controller gains
static const float KP = 0.16f;
static const float KI = 17.1f;
static const float KD = 0.0f;

```

```
51 // Controller Sample time period in ms
52 static const float TS = 2.0f;
53
54 #endif // BUCK_CONVERTER
```

code/buck_converter.h

buck_converter.c

```c
// Include single header library function implementations
#define BUCK_CONVERTER_PWM_IMPL
#define BUCK_CONVERTER_ADC_IMPL
#define BUCK_CONVERTER_PID_IMPL

#include "buck_converter.h"

void init_buck()
{
    // PWM set up
    PWM_setup(&pwm_timer, &pwm_channel, 50000, 0);
}

/*
 * Output voltage PID control loop.
 * This function initialises the PID controller, and the internal ADC on the ESP32.
 * It will then enter the main control loop, and will compute the following:
 *
 *      - Read the current buck converter load voltage. This is Measured using the
 *        ESP32 ADC, through the voltage divider defined by LOAD_R1 and LOAD_R2.
 *
 *      - Calculate the equivalent duty cycle that would be used to produce the
 *        current measured load voltage.
 *
 *      - Pass the target duty cycle, and current duty cycle through the PID
   controller
 *
 *      - Update the buck converter duty cycle with the new controller output
 *
 *      - Delay the task until the selected sample period defined by TS has elapsed
 */

void control_loop(void *pvParameters)
{
    // Setup an input queue for receiving new target voltages
    QueueHandle_t target_voltage_queue = *(QueueHandle_t *)pvParameters;

    // Load voltage ADC set up and local variables
    uint16_t supply_voltage_buffer[5] = {0}; // Supply voltage rolling average
    buffer

    // Variables for storage and conversion of load voltage
    uint16_t load_adc_raw;
    float load_adc_voltage;
    float load_voltage;
    float measurment_duty;

    // Variables for storage and conversion of
    uint16_t supply_adc_raw;
    float supply_adc_voltage;
    float supply_voltage;

    // Initialise the ADC structs
    esp_adc v_load = {// Create adc struct for reading the load voltage
```

```
                    // adc input channel 0 (GPIO 36)
                    .adc_channel = 0,

                    // do not allow for a rolling average
                    .span = 0,
                    .buffer = NULL};

esp_adc v_supply = {// Create adc struct for reading the supply voltage
                     // adc input channel 6 (GPIO 34)
                     .adc_channel = 6,

                     // 5 wide rolling average
                     .span = 5,
                     .buffer = (uint16_t *)&supply_voltage_buffer};

// Init the load voltage adc and check to see if it was successful
if (adc_init(&v_load) != ESP_OK)
{
    printf("Load voltage ADC init error!\n"); // If set up fails print the error
    vTaskDelete(NULL);
}

// Init the supply voltage adc and check to see if it was successful
if (adc_init(&v_supply) != ESP_OK)
{
    printf("Supply voltage ADC init error!\n"); // If set up fails print the
error
    vTaskDelete(NULL);
}

//PID set up
float target_voltage = V0;
float target_duty;

// Create the PID struct
PIDController voltage_controller = {
    // Controller gains
    .Kp = KP,
    .Ki = KI,
    .Kd = KD,

    // Differentiator low pass filter corner frequency
    .tau = 0.0f,

    // Min and max output duty cycles
    .limMin = 0.0f,  // 0% duty cycle
    .limMax = 0.98f, // 95% duty cycle

    // Integrator wind-up min and max
    .limMinInt = -5.0f,
    .limMaxInt = 5.0f,

    // Sample time period
    .T = TS / 100.0f,
};

```

```
108     // initialise the PID controller
109     PID_init(&voltage_controller);
110 
111     // Task variables
112     TickType_t xLastWakeTime; // Hold the time stamp of the last wake time
113 
114     // Enter the PID control loop
115     while (true)
116     {
117 
118         xLastWakeTime = xTaskGetTickCount(); // Store the current tick count
119 
120         // Read the supply voltage ADC
121         supply_adc_raw = adc_read(&v_supply);
122         supply_adc_raw = rolling_average(&v_supply, supply_adc_raw);
123         supply_adc_voltage = adc_conversion(supply_adc_raw);
124         supply_voltage = (supply_adc_voltage * (SUPPLY_R1 + SUPPLY_R2)) / SUPPLY_R2;
125 
126         // Read the load voltage ADC
127         load_adc_raw = adc_read(&v_load); // Take an adc reading
128         load_adc_voltage = adc_conversion(load_adc_raw); // Convert this value to
    the voltage at the adc
129         load_voltage = (load_adc_voltage * (LOAD_R1 + LOAD_R2)) / LOAD_R2; //
    Convert to the voltage at the buck converter load
130 
131         // Calculate the measured duty cycle and target duty cycle
132         measurment_duty = load_voltage / supply_voltage; // Calculate the duty
    theoretical duty cycle of the output
133         target_duty = target_voltage / supply_voltage; // Calculate the target duty
    cycle for the controller
134 
135         // Check for a new target voltage
136         if (uxQueueMessagesWaiting(target_voltage_queue))
137         { // If there is a new target voltage read it
138             if (xQueueReceive(target_voltage_queue, &target_voltage, (TickType_t)1))
139             {
140                 target_duty = target_voltage / supply_voltage;
141             }
142         }
143 
144         // Update the duty cycle with the PID controller
145         PID_update(&voltage_controller, target_duty, measurment_duty);
146 
147         // Output the new duty cycle
148         PWM_set_duty(&pwm_channel, voltage_controller.out);
149 
150         // Delay task until the provided time period is reached
151         vTaskDelayUntil(&xLastWakeTime, TS);
152     }
153 }
```

code/buck_coverter.c

main.c

```
#include "buck_converter.h"

void app_main()
{

    init_buck();

    // Declare task handles
    TaskHandle_t VO_controller = NULL;

    QueueHandle_t target_voltage_queue;

    // initialise the queues
    target_voltage_queue = xQueueCreate(5, sizeof(float));

    // Create the output voltage control task
    xTaskCreatePinnedToCore(control_loop,                   // Voltage control loop function
                            "Vout_controller",              // Task name
                            2048,                           // Task stack size
                            (void *)&target_voltage_queue, // Function parameters
                            2,                              // Priority of the task (app_main has priority 1)
                            &VO_controller,                 // Task handle
                            tskNO_AFFINITY);                // Core the task has been pinned to (No core selected)

    float input_voltage;
    char input_buffer[20] = {0};
    uint8_t buf_index = 0;

    while (true)
    {
        char input = fgetc(stdin);

        if (input != 0xFF)
        {
            input_buffer[buf_index++] = input; // read in the character, and increment the index
            fputc(input, stdout);              //echo the character back

            if ((input_buffer[buf_index - 1] == '\n'))
            {

                input_voltage = strtof(input_buffer, NULL);

                xQueueSend(target_voltage_queue, (void *)&input_voltage, (TickType_t)1);

                printf("\nInput Target Voltage: %f\n", strtof(input_buffer, NULL));
                buf_index = 0;
            }
        }

```

```
50          vTaskDelay(1);
51
52          // printf("task stack unused: %d\n\n",
        uxTaskGetStackHighWaterMark(VO_controller));
53          // vTaskDelay(100);
54      }
55  }
```

code/main.c