ABSTRACT

JUNEJA, AVIK, Embedding Power: A Real Time Approach in Incorporating SMPS within Low-Cost Embedded Systems. (Under the direction of Dr. Alexander G. Dean.)

A switch-mode power supply (SMPS) converts power efficiently between different voltage levels, making power and energy optimizations such as dynamic voltage scaling (DVS) feasible. Currently there is little overlap between SMPS and embedded system design communities. SMPS controllers are implemented in hardware or are separate processors, and are expensive when added to low-cost embedded applications. It is possible to integrate SMPS control into the application's microcontroller. Thus, 'embedding the control of power source,' in an embedded application will not only reduce costs but also enable various optimizations.

This work examines the real-time computational requirements for software which performs closed-loop control of switched-mode power supplies. By explaining the interplay between switch-mode power converter circuits and digital control, a formal, quantitative understanding of the real-time requirements of SMPS control is provided.

A buck converter controlled by a 32 MHz 16-bit microcontroller (Renesas' RL78G14) is evaluated for its transient response, which varies due to several anomalies in the system, such as, control software blocking times, capacitor technology, software control parameters, and other circuit parameters. These methods apply to a wide range of software task schedulers, from simple interrupt-based foreground/background systems to sophisticated preemptive real-time kernels.

Furthermore, a real-time embedded application running real-time operating system (RTOS) with various peripherals is evaluated and compared with various power conversion and management schemes. The methods are demonstrated on a position-logging embedded system, featuring a GPS receiver, Wi-Fi interface, LCD and micro SD card. The position logging system is designed using two different micro-controller units (MCUs) running different RTOS on them. One system consists of three buck converters controlled by the application's 48 MHz ARM Cortex-M0+ microcontroller, running a preemptive real-time kernel (RTX). The proposed methods reduce power by up to 48% compared with a single-rail SMPS system. Another system uses a 32 MHz 16-bit microcontroller (Renesas'

RL78G14) with a run-to-completion (RTC) scheduler for the same application. This system resulted in up to 88.6% power savings, compared with a system using single rail linear regulator.

An alternate platform, from the Renesas microcontroller, was designed for CubeSat's Electric Power System (EPS) is also partially evaluated, when burdened with four loads in parallel. This system uses a DSP (digital signal processor), operating at 150 KHz with floating point hardware, to control of four output voltage domains, offering opportunity to evaluate a broad range of control approaches.

Embedding Power: A Real Time Approach in Incorporating Switching Power Converters within Low-Cost Embedded Systems

by
Avik Juneja

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Engineering

Raleigh, North Carolina

2015

APPROVED BY:

_____	_____
Alexander Dean Chair of Advisory Committee	Subhashish Bhattacharya
_____	_____
Gregory Byrd	H. Troy Nagle

DEDICATION

Dedicated to

My Parents

Babita Juneja & Pawan Juneja

My Grandparents

Dharmendra Nath Juneja & (Late) Uma Juneja

My Brother

Manan Juneja

BIOGRAPHY

Avik Juneja was born to Babita Juneja and Pawan Juneja in New Delhi, India. He obtained his primary education in Silver Spring, MD (Takoma Park Middle School), and schools in New Delhi, India, including Sanskriti School. He obtained his Bachelor's degree in Electronics and Communications Engineering from Delhi College of Engineering, Delhi University, India in 2009. Thereafter, he enrolled in the PhD program at North Carolina State University in the Electrical and Computer Engineering Department during Fall 2009.

He has been working with Dr. Alexander Dean and Dr. Subhashish Bhattacharya on several research projects involving low cost embedded systems and embedded firmware design. His primary research focuses on power management in cost-sensitive and performance limited embedded systems using integration of SMPS control with real-time application firmware for improved energy efficiency. His interest also lies in micro-architecture design and power management techniques at micro-architecture level.

He has completed several internships at corporations such as Texas Instruments (Bangalore, India), Eaton Corporation (Raleigh, US) and an 8 month co-op at Intel Corporation (Hillsboro, OR).

Besides research, Avik also enjoys teaching fundamentals of Embedded Systems' Design, Computer System Design and Micro Architecture.

ACKNOWLEDGMENTS

The accomplishment of this thesis is attributed to many significant persons in my life. My first and foremost acknowledgment extends to Dr. Alexander Dean, whose outstanding mentorship, constant faith, and persistent encouragement and guidance has enabled me to attain a level of excellence in research, to his satisfaction. His immense knowledge and expertise in the field of embedded systems and firmware design has helped me over the years in not only overcoming difficulties, but allowing me to discover research solutions in an efficient and a holistic manner. His enthusiasm in the classroom and the ability to impart knowledge with such clarity motivated me to enroll as a teacher under his supervision for one of his courses (Embedded Systems' Design). The feedback and experience gained as a teacher has helped me become a better researcher. I am also grateful to my co-advisor, Dr. Subhashish Bhattacharya for his valuable advice and continuous support throughout my graduate studies.

I sincerely extend my gratitude to Dr. Gregory Byrd and Dr. H. Troy Nagle for providing useful insights into my research, and serving as members of my advisory committee. I would also like to thank Dr. Eric Rotenberg for research inputs, and for introducing me to the field of micro-architecture. His unparalleled teaching and presentation techniques, and constant strive for perfection has motivated me to achieve a whole another level of excellence in my work.

Most importantly, I owe my sincerest and deepest gratitude to my parents, Mrs. Babita and Mr. Pawan Juneja, my grandfather, Mr. Dharmendra Nath Juneja, and my brother Manan Juneja. I am indebted for their sacrifices in educating me, preparing me for my future and for their invaluable support throughout my life. They have showered me with their unconditional love and blessings. Their guidance and principles of hard work, dedication and honesty, throughout the walks of life have inspired me. Without their support, this degree could not have been accomplished.

I would also like to thank my extended family for their support, blessings and constant guidance: my maternal grandparents, Mrs. Pushpa and Late. Mr. B.N.P. Doultani, for their

blessings and love; my uncle, Mr. Rajendra Kashyap, and my cousin, Dr. Mayank Kashyap, for being my mentors since my childhood and guiding me throughout my education and career; Mrs. Anju Kashyap (aunt), Tulika Kashyap (cousin), Mrs. Komal and Mr. Anil Doultani (aunt and uncle), and Mr. Deepak Doultani (uncle) for their love and blessings; my family in Raleigh, Mr. Ashish and Mrs. Rina Nijhawan for their support and care.

A special thanks to my fiancé, Vrinda Thareja for her incessant support and care during the preparation of this report.

I am grateful to my best friends, Prashant Kumar, Sanup Haridas, Nikhil Mahajan and K. Karthik, who are like my family. I thank them for their unconditional support and understanding through various phases and events in my life.

I've had wonderful friends, peers and colleagues who have made my experience very educative, eventful and fulfilling during my doctorate. In this regard, I would like to thank Swapandeep Garcha, Gaurav Manchanda, Saurabh Gupta, Rahul Ramasubramanian, Rohit Taneja, Devesh Tiwari, Sandeep Navada, Rajeshwar Vanka, Sahil Sabharwal, Abhishek Bhattacharya, Raj Kumar, Bharat Malhotra, Rangeen Basu Roy Chowdhury, Arun Kadavelugu, and Sumit Dutta.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

This work targets the control of switch mode power converters to be embedded within a low-cost Embedded System. The novel technique leads to efficient power conversion, enables dynamic voltage and frequency scaling (DVFS), and at the same time increases system's flexibility and scalability in terms of supporting various voltage domains for optimal operating modes.

A major portion of this study examines the co-design of digitally-controlled switched mode power supplies (SMPS) using real-time system methods. Initially, a domain of low-end embedded systems is targeted, where cost pressures limit the available hardware and computational throughput. The fundamental questions to understand and answer are: 1) How do load transients translate into computational requirements; 2) How can the circuit design and control system be adjusted, in order to ease computational requirements (e.g. CPU utilization, response time, etc.) and vice versa?

The major contributions made in this research are enlisted below and elaborated in following chapters.

1. Quantifying the above mentioned fundamental questions.
    a. Relationship between the control loop update rate, $\mathbf{1/f_c}$ (in software) vs voltage deviation (in turn, $\mathbf{V_{margin}}$), and
    b. Control loop update rate as a function of circuit parameters.
2. Burst mode control for increased efficiency at lower loads along with reduced MCU utilization.
3. The software control of SMPS proved to improve the energy efficiency and hence, battery life of the targeted application.
4. Cost analysis performed on the proposed power management methodology proved to reduce cost while increasing flexibility, efficiency and scalability of the system.

1.1 Power Conversion in Real-Time Embedded Systems

Dynamic voltage scaling is a common method for saving power and energy in high-performance computer systems. Power dissipated due to dynamic switching of any digital logic circuit is directly proportional to the square of the operating voltage, *V*, and the switching frequency, f_{clock} , of the components as per the equation (1.1) [1] [2]. Thus, it is imperative to conserve energy by scaling voltage and frequency, whenever promising, as per the load requirements. The use of switching converters provides the flexibility of scaling these voltage levels, whereas, the frequency changes are enabled using programmable logic. However, the cost of energy-efficient (i.e. switching) voltage converters limits the use of DVS in low-cost embedded systems. Due to this reason, most of the low-cost systems use linear regulators with fewer voltage domains.

$$P_{dynamic} \alpha \ V_{CC}^2 f_{clock}; \quad P_{dynamic} \sim C_P V_{CC}^2 f_{clock} \quad where, V_{CC} \alpha \ f_{clock} \tag{1.1}$$

$$P_{static} = \left(\frac{W}{L}\right) I_{S0} e^{\left(\frac{-V_{off}+V_{th}}{nVt}\right)} Vcc \ ; \ P_{static} \sim S_P V_{CC}^2 \quad ; S_P = Conductivity \tag{1.2}$$

$$P_{total} = P_{dynamic} + P_{static} \tag{1.3}$$

Linear regulators, although cheap, result in high power losses due to their dissipating nature. Due to the reduced efficiencies, fewer voltage domains are present in a system. Hence, many on-board devices and peripherals are made to operate at sub-optimal power levels.

One way to reduce costs is to integrate the SMPS control software (control system and protection) into the embedded system's microcontroller unit (MCU). To do this successfully, the SMPS must be able to tolerate the timing interference from higher-priority application processing and interrupt lock-out times. This not only allows efficient and low cost power conversion, but also enables multiple voltage domains to be incorporated in a system, further allowing on-board peripherals to operate at optimal power points, and hence increasing overall system efficiency.

This study contributes methods to analyze and model SMPS control functionality on an MCU. The SMPS controller requirements are presented with real-time scheduling formalisms, enabling the composition of real-time systems with onboard control of one or

more SMPSs. The presented methodology is demonstrated and validated through various experiments.

Prior research has analyzed and quantified the voltage response to a load transient for best control effort, and has assumed the controller is implemented in dedicated digital logic or on a DSP completely dedicated to that task. Our work is different in that we use real-time system design and analysis methods in order to determine how to share the processor between SMPS control (which is inherently time-critical) and other processing. No one has characterized the real time implication and processing requirements of software controlled power supplies, SMPS in particular. The effects of blocking of the control function, jitter in sampling, relative occurrence of load transient with respect to a switch period of the SMPS, have not been previously studied. These factors can result in various anomalies in the transient response causing large deviation in voltages and eventually leading to unwanted behavior in the load.

Embedded SMPS control in the application MCU requires not only the load characterization but also software compliancy to ensure the control software operates as expected, while allowing other real-time scheduled tasks to perform operations within stipulated *deadlines* in low-cost systems with limited hardware capabilities. The real-time analysis carried out in this study provides an understanding of the control software and its hardware implications. It will be shown in Section 3.4.2 how various control software control parameters affect the hardware response and voltage regulation. Effects due to parameters like control loop frequency, f_c, worst case execution time (WCET) and blocking time, t_b, allow us to perform schedulability analysis. This analysis includes MCU utilization bound tests and a more precise analysis of tasks in a system, using Worst Case Response time (WCRT) calculations. The schedulability analysis presented in Section 3.4.1 connects the dots between the control parameters, timing parameters, software overhead, and task interference, thus, paving way for future work in analyzing task schedulability, response time requirements, MCU utilization and overall software performance. This will allow the control software to be embedded using suitable schedulers such µCOS-III and perhaps other pre-emptive or run-to-completion (RTC) scheduling based real-time operating systems (RTOS).

This study enables the SMPS designers as well the embedded system design engineers to take advantage of a simple switching power converter and tailor the system, either for efficient power management, or to boost performance of the system by extracting CPU utilization for application tasks by loosening the SMPS control effort. All this is done keeping in mind the system maintains correctness and appropriate power levels based on real time applications. The tradeoffs are straightforward, and this approach leads to a very power efficient system with flexibility in terms of performance and power consumption.

Improving energy efficiency allows a designer to extend a system's operational life, reduce battery size and weight, and use more sophisticated (and computationally intense) control methods. For some applications, it may even be possible to eliminate the battery through the use of energy harvesting, ultra-capacitor and wireless power transmission technologies. These technologies are advancing rapidly, further increasing the potential market for and impact of extremely energy-efficient embedded systems [3] [4]. The complementary trends of reduced computational energy costs and increased energy availability increase the impact of the proposed work.

1.2 Switching Converters

Switching converters use storage elements (inductors and capacitors) to efficiently raise a voltage (for a boost converter) or lower it (for a buck converter). Efficiency for these converters is typically above 80%, even reaching into the upper 90s. Linear regulators, on the other hand, can only lower a voltage, without storage, instead, by passing it through a transistor operating as a variable resistor. This wastes power proportional to the voltage drop times load current.

The basic operation of a buck converter (which lowers a voltage) is explained using Figure 1.1**.** When the transistor Q1 is ON and Q2 is OFF, the input voltage, V_{in}, is connected to the inductor, L. This leads to storage of energy in the inductor. In the next switching cycle Q2 turns ON and Q1 turns OFF, input source is disconnected from the circuit and the inductor supplies the required energy to the load. Due to the switching action, the output voltage is a fraction of the input voltage. The fraction is determined by the ON time of Q1, in other

words, its duty cycle, D. A low pass filter is required to pass the DC component and reject any AC components that might occur at switching frequency, f_{sw}. The switching frequency depends on the switching speed of the transistors, inductance and the bandwidth of the converter, $\boldsymbol{f_c}$ [5] [6].

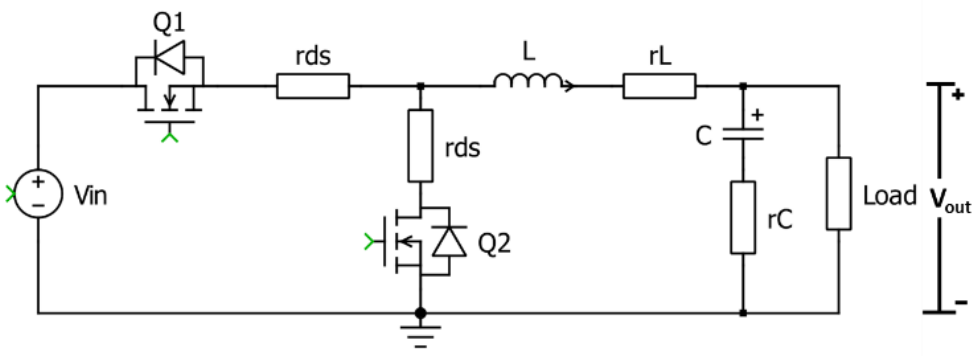


Figure 1.1: Synchronous Buck Converter

Similarly, the principle behind the operation of a boost converter (which raises a voltage) is that the energy stored in the inductor is reinforced by the input voltage source, resulting in the boost operation, as depicted in Figure 1.2. Initially, when Q1 is ON and Q2 is OFF, the energy is supplied to the inductor as it resists the change in the input. When Q1 and Q2 switch states, the inductor again resists the change in polarity but this time it comes in series with the input. This leads to higher energy supplied to the load and storage in the capacitor. When the inductor current falls below a value, switching again occurs, the inductor is charged and the capacitor supplies the energy to the load.

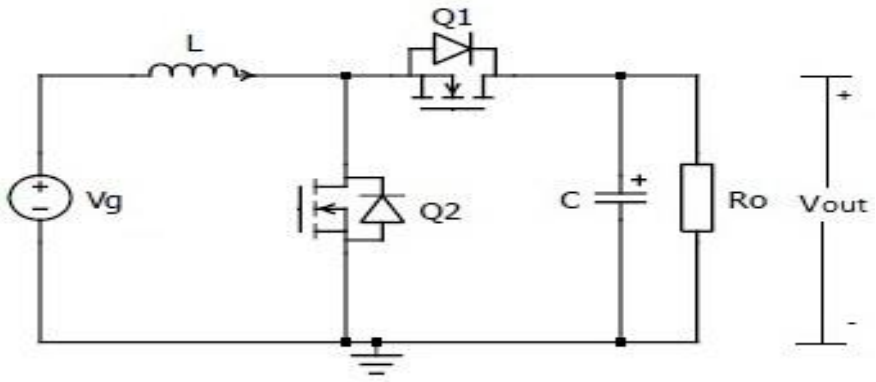


Figure 1.2: Synchronous Boost Converter

The **control input** to an SMPS converter is a pulse width modulated (PWM) signal of frequency f_{sw} which drives the transistors Q1 and Q2 (or a diode D1 in a non-synchronous converter). The duty cycle *D* of the PWM signal determines the DC voltage conversion ratio of the circuit under ideal, steady-state conditions. For a buck converter operating in continuous conduction mode, the output voltage is simply:

$$V_o = DV_g; \tag{1.4}$$

For the boost converter it is:

$$V_o = \frac{1}{1-D}V_g; \tag{1.5}$$

These basic equations do not include transient behavior and parasitics. We must model these because they are critical to our system analysis.

It should be noted that in simpler converters, a diode is used in place of Q2. The use of two transistors eliminates the forward voltage drop due to the diode, and allows a faster switching frequency for conversion. This type of buck/boost converter is known as synchronous buck/boost converter, due to the synchronous switching nature of the two transistors.

The switching behavior of the converter superimposes a small saw-tooth ripple on the output voltage, as characterized in equation (1.6). The ripple period is equal to the switching period ($T_{sw}=1/f_{sw}$) of the PWM signal. The maximum ripple amplitude depends on various factors, including the converter's inductance, switching frequency, and output capacitance.

$$\Delta V_{ripple,max} = \frac{(V_{In} - V_{out})D}{f_{sw}L} * \left(\frac{1}{8f_{sw}C_{out}} + r_C\right) \quad (1.6)$$

The term r_C above is the parasitic resistance of the output capacitor. Actual implementations of power converters use real, imperfect components which feature parasitic circuit characteristics. The most critical in this domain are the equivalent series resistances (ESRs) of the transistor switches (r_{DS}), output capacitor (r_C) and inductor (r_L), and the capacitances between the switch (transistor or diode) terminals.

1.3 SMPS Response to Changes in Load

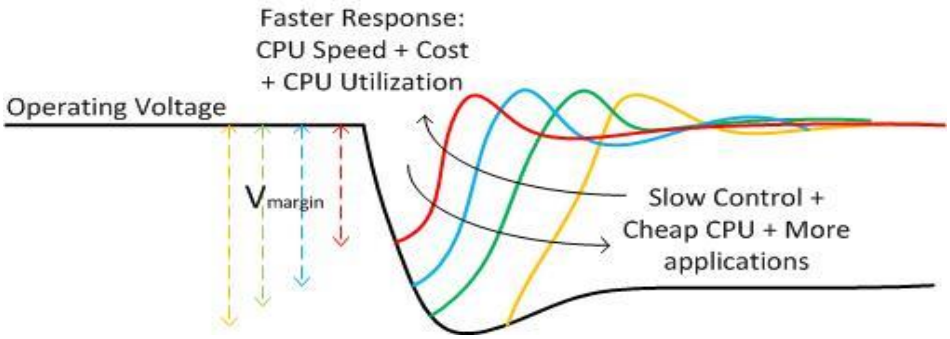


Figure 1.3: SMPS output voltage falls in response to increase in load current. Reducing the voltage drop requires faster processing by the SMPS controller

Embedded systems typically include multiple peripheral devices in addition to the microcontroller. The current drawn by these devices varies, changing the amount of current drawn from the power supply. The current drawn may increase (a **loading transient**) or decrease (an **unloading transient**). Transients for embedded system devices may reach tens of milliamps, mA (high-brightness LEDs, SD memory card access, Bluetooth radio) or even hundreds of mA (e.g. Wi-Fi radio).

The voltage conversion ratio V_{out}/V_{in} of an SMPS depends primarily on the duty cycle *D* of its control input when the output load is steady. However, changes in the load current

introduce transient conditions which affect the output voltage. We need to ensure that the voltage will not drop below a given minimum voltage or else the system may malfunction.

Figure 1.3 shows various possible responses by an SMPS to a loading transient. The output voltage drops from the desired value (set-point) and may recover partially or completely. The **open-loop response** (bottom trace, in black) shows V_{out} drops to a minimum value and then settles at a final voltage below the desired set-point. When the load current decreases, an unloading occurs, leading the output voltage to respond in a similar way but by rising.

Most of peak error and all of the final error can be eliminated by adding a control system (compensator) to adjust the SMPS duty cycle *D* based on the error. The color traces show the **closed-loop response** of the system when controlled by various control systems. Accelerating the controller's response to a transient cuts the size of the voltage transient, as shown by the progression of the yellow, green, blue and red traces. However, improving the responsiveness requires increasing the control loop frequency, which raises CPU processing requirements. On the other hand, if a moderate voltage variation is acceptable, we may be able to reduce the CPU processing required.

1.4 SMPS Overview

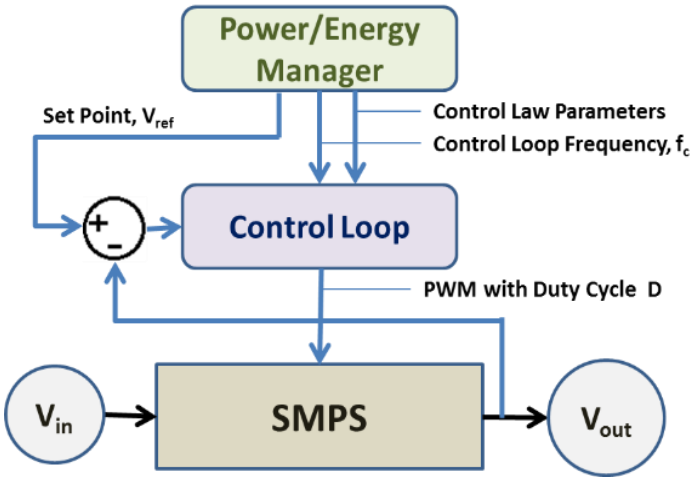


Figure 1.4: Overview of system voltage conversion approach

Consider a system shown in Figure 1.4. There are multiple layers: the converter at the bottom converts input V_{in} to output V_{out} according to the duty cycle D of a pulse-width-modulated control signal. D is adjusted by a digital control loop above so V_{out} matches the desired output voltage V_{ref}. The upper layer defines the 'set-point' as needed for DVS, DVFS, and other forms of power and energy management.

We target the buck converter (shown in Figure 1.1) due to its ubiquity, simplicity, low cost and extensive analysis [5] [7] [8] [9].

1.5 Organization

This thesis is organized as follows. Chapter 2 discusses the existing technology and studies in the field of converter designs, converter characteristics and software implementations. The deficiencies of related work along with the takeaways are also highlighted in the same chapter. Contributions of this study along with concepts and implementation details are discussed in Chapter 3. Experiments and analysis carried out in order to understand the contributions are presented in Chapter 4. A prototype system was developed to prove the proposed techniques in power conversion and power management. The integrated system and results are presented in Chapter 5. Some potential future work and optimizations feasible, which can exploit the current implementation, are discussed briefly in Chapter 6. Finally, Chapter 7 concludes this study.

Chapter 2

Related Work

2.1 Existing Technology

In the power supply design community, digital control of switching converters has been a very active field [6] [10] [11] [12] [13] [14]. The transient response has been investigated to identify critical relationships between circuit parameters [8] [15]. It is especially important for voltage regulator modules which must handle the large transients of power-hungry high-performance processors [5] [9].

The power design communities have solely relied upon circuit parameters and control theory in developing power supplies to handle load transients. Due to the independent nature of their designs, dedicated hardware is used for voltage regulation. This increases the area on-board, and the cost of developing a high performance regulation technique. Although a good solution for high-end systems exists, this technique is impractical for low cost real-time systems.

There have been many investigations into the software issues with controlling power converters using MCUs and DSPs [16] [17]. In fact, many MCU and DSP vendors provide working reference designs for such applications. But again, these are software running on dedicated MCUs and DSPs solely responsible for power conversion. The constraints on their performance are far more relaxed than that of our system in which the power conversion software is embedded within the application processor and shares the resources. Having dedicated processors for this purpose increases cost as well as the complexity of interaction between power conversion and real-time application.

The real-time design community has investigated the implications of jitter on digital controllers for other applications and has suggested various solutions [18] [19] [20]. This information is of particular interest to us in determining the exact behavior of application

software anomalies and how they may affect the SMPS control software, if embedded in the application MCU.

Designers of high-performance processors (including those for personal computers and cellphones) have long relied on buck converters for dynamic voltage scaling. There are many papers which concentrate on power management at the microprocessor level [21] [22] [23] [24] [25]. These discuss various techniques to avoid voltage fluctuation by use of sense or prediction techniques, such as application interactions, microprocessor's architectural changes, and dedicated hybrid multi-phase converters. Due to their high performance nature, they are equipped with dedicated controllers for these converters, and the cost of power conversion is relatively small as compared to that of the entire system.

One of the initial endeavors in controlling an inverter by a microcontroller for uninterruptible power supply (UPS) applications had been proposed in [26]. Khan and Manning used Intel's 8088 microprocessor technology for this purpose, and Khanniche [18] employed the 8097 microcontroller. In both cases, the control algorithm was written in assembly and the microprocessors were dedicated in controlling only one voltage domain, without any RTOS support to include other domains or even applications. Sorescu [19] made similar attempts, but with no real time analysis of the SMPS transient response, and only one voltage domain being regulated. Rahman *et al* [27] talk about multimode digital SMPS controller IC. They have come up with a dedicated hardware for digital control with no room for other applications to run or flexibility in terms of voltage scaling in real time. [28] [29] go a step ahead in incorporating the SMPS control using real time scheduling with other applications in the system. They discuss the scheduling technique to avoid Electromagnetic Interference (EMI) caused by the switching modules in SMPS and come up with a Make and Take (MAT) scheduling model. This technique is only responsible for controlling the timing of SMPS operation so as to avoid EMI with other sensitive operations like ADC conversion. MAT does not perform any analysis of the control algorithm and neither does it incorporate multiple voltage domains. There are other papers which concentrate on power management at the microprocessor level, as well. Joseph *et al* [24] presented circuit level sensing techniques with application characterization to sense and eliminate voltage emergencies.

Their work confined to simulations and assumed availability of accurate circuit level sensors. An improved and faster on-chip 3-level DC-DC converter was introduced in [30]. In this paper, the properties of a buck converter and a switched capacitor (SC) type converter were combined to form a hybrid regulator for tight control with faster response, but with reduced efficiency.

Most of the low-cost embedded systems employ linear regulators for power conversion. This reduces cost and area, but also results in inefficiencies and limits optimizations in such systems.

All of the above mentioned works on power regulators can be categorized and applied towards specific applications, and are useful for our analysis to some extent. But none of these provide a holistic solution which is cost and power efficient. Our aim is to bring together the experience and expertise of the power design community and merge it with that of real-time design community in coming up with a system which is not only power efficient, but also flexible and scalable at a reduced cost. This will empower a real time system developer to configure the system to their need for maximum performance or efficiency, and at the same time extend a new dimension for power and performance optimization.

2.1.1 Work on characterizing transients

Previous work by some authors [9] [5] [8] are of particular interest, as they provide an insight and a fairly clearer picture of the response of a switching converter in case of load transients. They have quantified these responses and come up with best case voltage deviations when reinforced with optimal feedback control.

2.1.1.1 Optimizing the Load Transient Response

Redl "analyzes the system, determines the best compensation, provides design guidelines for buck converters powering Pentium II processors, and presents results of simulations and experiments." [8]. The work introduces the transient characteristics and breaks it in different phases as depicted in Figure 2.1. The quantification of the different phases is not presented; rather the bounds are presented for the response.

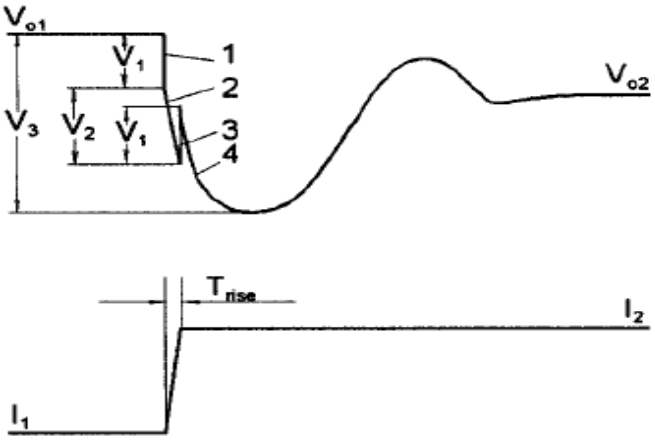


Figure 2.1: Output voltage (top) and load current (bottom) [8]

The presented best case is found to depend on the control effort, which is assumed to be optimal for their case (saturation of duty cycle). The worst case is similar to be the open loop deviation if the response is sluggish. The quantification of the open loop deviation is similar to a LC circuit, neglecting all losses due to effective series resistances (ESR) and loss components. Since the loss components only increase damping, thus decrease peak deviation, it is safe to assume the worst case to be similar to a deviation in ideal circuit.

2.1.1.2 Digital VRM Controller & Load Line Regulation

Similar to Redl, Peterchev *et al* [9] [5] have presented a very complex and dedicated voltage regulation module (VRM) implemented as an independent IC for applications with high performance processors. Their work also characterizes the transient response of a buck converter, using similar techniques, along with detailed analysis on factors affecting the response. The work analyzes system response due to parameters such as Critical Bandwidth, Critical Capacitance, and Critical Inductance. The system response due to these circuit parameters is of interest to our study and has helped us come up with a holistic approach in SMPS design, which will be discussed in following chapter with appropriate references. [5] analyzes similar transient behavior but presents another method of an improved control of

such regulators using estimated load-current feed-forward, which is not analyzed in our work, due to its complexity.

Goals: The primary goal of the work above is optimizing the transients using known circuit reduction techniques, control methodologies and circuit changes (increasing output capacitor, load current/capacitor current feed-forward, V^2 architecture, zero-impedance converter), and then employing a specific type of current mode control. This is all preformed using dedicated control logic and hardware.

Deficiencies: Since the goal of presented in the paper is different from ours, their work lacks any real time characterization of the transients, which is required for real-time control application. The presented work assumes an optimal control to determine best case deviation, which might not be the case in real-time applications. Similarly, the worst case, the lower bound on the controller response approaches open loop condition, and there is no intermediate analysis. The application is very specific (high performance microprocessors), their analysis is limited and makes assumptions (optimal or extremely sluggish control), which is of limited help in our work. Their work assumes dedicated control logic/IC for the processor's power supply and different control techniques which might be better, but complex and expensive (both in terms of cost and development).

Learnings: From these works, the knowledge of the transient characteristics, worst and best case controller response are utilized in determining bounds of the system. In our study, we define the behavior of the system not only based of circuit parameters and digital control, but also from the point of view of software implementation and real-time requirements. The tradeoffs due to relaxed and suboptimal control are determined. This gives a better understanding while designing and controlling an SMPS with known configurations and expected behavior.

Chapter 3

Contributions

3.1 Introduction

There are multiple reasons to use an SMPS in an embedded system -- the two biggest are efficient power conversion for supply rails, and efficiently driving LEDs with a constant current source. DVFS is one way to efficiently and adaptively power the supply rails for digital logic (such as processors) which support clock scaling. We discuss DVFS here as a motivating example but it is not the only application or method.

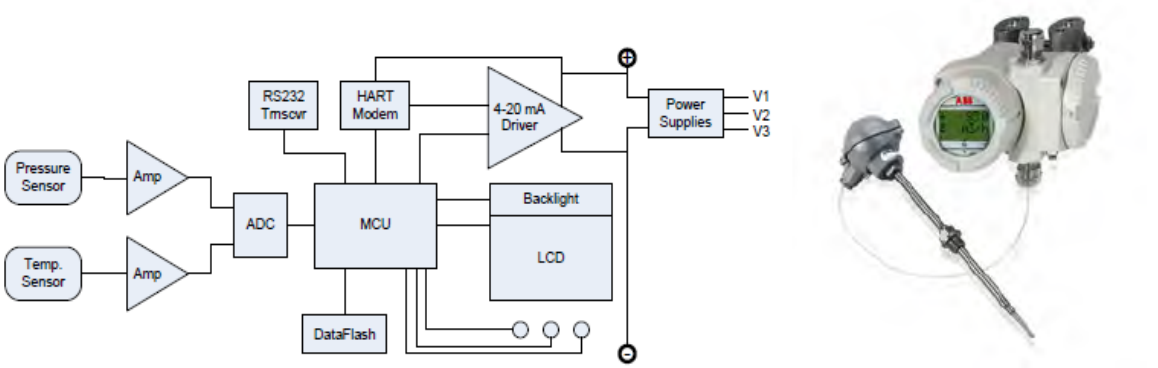
Due to the overall higher cost of the system, it is possible to relegate the task of power management and voltage regulation to dedicated hardware(s) (VRMs). The application processor simply communicates with the VRM and assumes perfect 'on-demand' regulation. The VRM, being a standalone device, performs regulation using devoted hardware/software, without loading the application processor.

In low-cost and real-time embedded systems, incorporation of such VRMs increases the cost of the system significantly. Thus, the only method of voltage conversion is by deploying linear regulators. These regulators are inherently very inefficient as they dissipate the residual power in order to perform voltage conversion. The reduced efficiencies of such converters limit their usage in the system, leading to lesser number of voltage domains available.

Any real-time system consists of various peripherals responsible for several tasks. The operating voltage ranges of these peripherals also vary. Due to limited voltage domains, many of the devices are made to operate at sub-optimal voltages (higher or lower), leading to further inefficiency. This is explained using an example below.

Consider an example of a power-limited embedded application – the remote pressure and temperature transmitter shown in Figure 3.1 (a) and (b), powered by a current loop. This

application cannot afford to use a microcontroller more expensive than $4[1]. This example is **power-limited**, but the ideas presented typically also apply to **energy-limited** systems. Other power-limited sources include solar, wireless, thermoelectric and piezoelectric. Energy-limited sources include batteries and ultra-capacitors.



(a) Example hardware block diagram

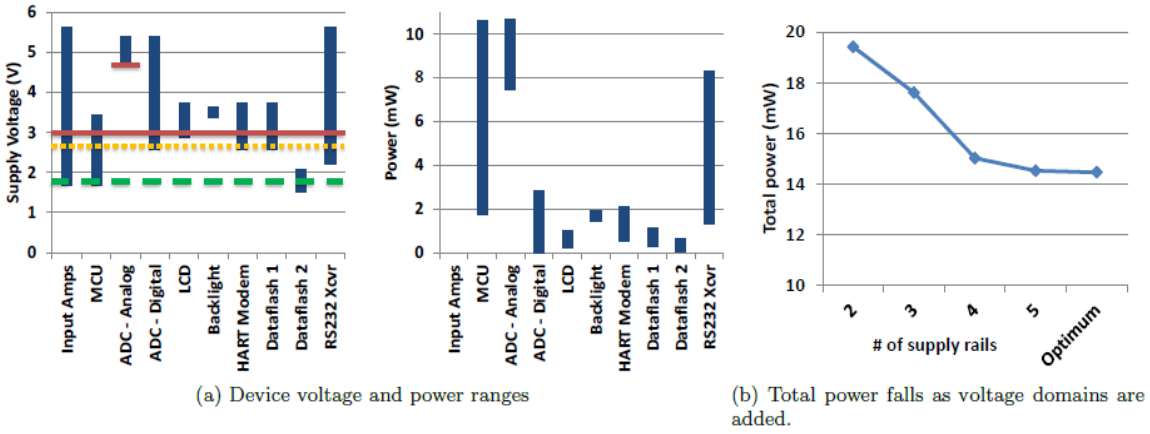
(b) Commercial device (courtesy ABB)

Figure 3.1: Multivariable pressure and temperature sensor

Industrial process control requires accurate (e.g. within 0.04%) and timely (e.g. 22 Hz update rate) monitoring of process variables such as temperature, pressure, flow rate, etc. The industry standard communication system is a 4-20 mA current loop. A master provides a supply voltage (e.g. 9 to 40 V) and the transmitter gauge controls the amount of current it draws to indicate analog process variable values ranging from 4 mA (0%) to 20 mA (100%). Additional process variables are transmitted using the HART protocol and its FSK modulation. By designing the transmitter so its operating current is always under 4 mA, one can eliminate the need for a separate power supply. This is industry practice for many devices, as it saves installation and maintenance time and costs.

[1] This example is based on actual industrial product, so the prices and design trade-offs reflect real constraints faced by practicing engineers.

Figure 3.2(a) shows the voltage range (left) and corresponding power consumption (right) for these components at those voltages as derived from their characterizations in data sheets. The non-overlapping voltage ranges result from reasons such as needing to support existing communication standards, and use legacy parts (which are typically less expensive than new parts).



(a) Device voltage and power ranges

(b) Total power falls as voltage domains are added.

Figure 3.2: Power characteristics for example system

Consider how to reduce the power used by this system. One approach is to use two supply voltages (red solid lines in Figure 3.2(a): 4.75 V and 3 V. This results in a power consumption of 19.4 mW. Adding a third supply (2.7 V, yellow dotted line) reduces power to 17.6 mW. Adding a fourth supply (1.8 V, green dashed line) allows us to switch to a lower-voltage Dataflash part, further reducing power to 15.0 mW. Operating each component at its lowest voltage yields the lowest power possible: 14.4 mW. However, this solution is **costly**. It requires six voltage domains with a regulator for each, and multiple signal level translators. Reducing the number of voltage domains will cut the number of regulators and level translators required, at the cost of increased energy consumption. Figure 3.2(b) shows the minimum power required given two, three, four, five or six (optimum) voltage domains.

Which number of voltage domains is best depends on the costs and overheads of adding more regulators and translators as well as their power conversion efficiency.

In this work we aim at employing switching converters for voltage regulation, but instead of having dedicated hardware for their control, the control is embedded in the application MCU. This not only eradicates the need of special control hardware, but allows interaction between the control software and application software, leading to an efficient power management based on load requirements.

The embedding of power control software into the application software is not straightforward, as it requires in-depth knowledge of system's response to several hardware and software dynamics. The characterization of the SMPS's transient response is of foremost importance to have a reliable voltage regulation. The transient characteristics are not solely dependent on the load transients, but also on control methodology, software interactions (jitters, ADC sampling, blocking by other tasks), circuit parameters, and operating points. This work presents the effect on SMPS transient and response based on these parameters through mathematical and software analysis, and experiments done on hardware.

In this chapter, the contributions of the work are highlighted through theoretical and conceptual explanations which form the basis of our Experiment and Analysis, presented in the following Chapter, which are performed to validate those contributions.

A mathematical model is derived for the buck converter through standard modelling techniques. Using this model, a feedback compensator will be designed for reliable closed loop control of the SMPS, with desired characteristics. In Section 3.3, the SMPS transient response will be reviewed in detail, based on prior techniques relative for real-time requirements, giving Worst, Best, and Exact Closed Loop response parameters. Real-Time analysis of the SMPS based on various software parameters is performed in Section 3.4.

The converter efficiency is analyzed and new methodologies are proposed in Section 3.7 in order to extract maximum efficiency at light, mid-range and heavy load conditions. Burst mode control employed at light load conditions also result in lower MCU utilization is explained in Section 3.7.

Energy and power improvements are quantified by employing a set of fully charged ultra-capacitors to power the systems. The energy and power consumptions are measured using a timed approach while monitoring the voltage on the ultra-caps as the systems consume fixed amount energy. This methodology is explained in Section 3.6.

In order to analyze the software complexity and computational requirements, readily available tools and standard approach are used as explained in Section 3.8.

The cost incurred in developing the proposed power management technique is compared with conventional methodologies by using TI's Webench tool. Further details of the cost analysis and the tool are provided in Section 3.9.

3.2 System Definition and Overview

Mathematical modeling, simulations and actual hardware response of the open loop converter have helped quantify and unveil some key parameters of the system and their effects on the transient response, which resulted in efficient, fast and reliable regulation. Synchronous and non-synchronous buck, and synchronous boost converters were first modeled mathematically and then under a simulation environment, while including their loss components. Most of the analysis presented is for the Buck converter, but similar analysis also holds true for the Boost. Effects of each converter component were studied for their effect on converter's transient response under open loop conditions. Upon completion of the converter modeling, an overall digital closed loop system was defined and modeled. This led us to examine the digital control system in depth, which adjusts the buck converter to improve output accuracy and transient response. Various control techniques were compared to determine the best suited control design method.

3.2.1 Modelling Techniques

Two widely used approaches are employed to model the buck converter, which is of main interest in this study. One such approach is the circuit averaging technique, as explained in [7]. The other is a mainstay of modern control theory and is known as state space averaging [11] [7] [10]. These techniques result in the same steady state equations and dynamic small

signal AC model of the concerned buck converter. The *synchronous buck converter* has been analyzed here. A similar approach was taken while modeling the *non-synchronous buck* and *synchronous boost* converters, and in incorporating it in the SMPS system. While modeling the converters, parasitic loss components were included in order to approximate the real hardware as closely as possible. The schematic of the synchronous Buck converter and its resulting small-signal ac equivalent circuit are shown in Figure 3.3 and Figure 3.4, respectively. The parameters are explained in Table 3-1.

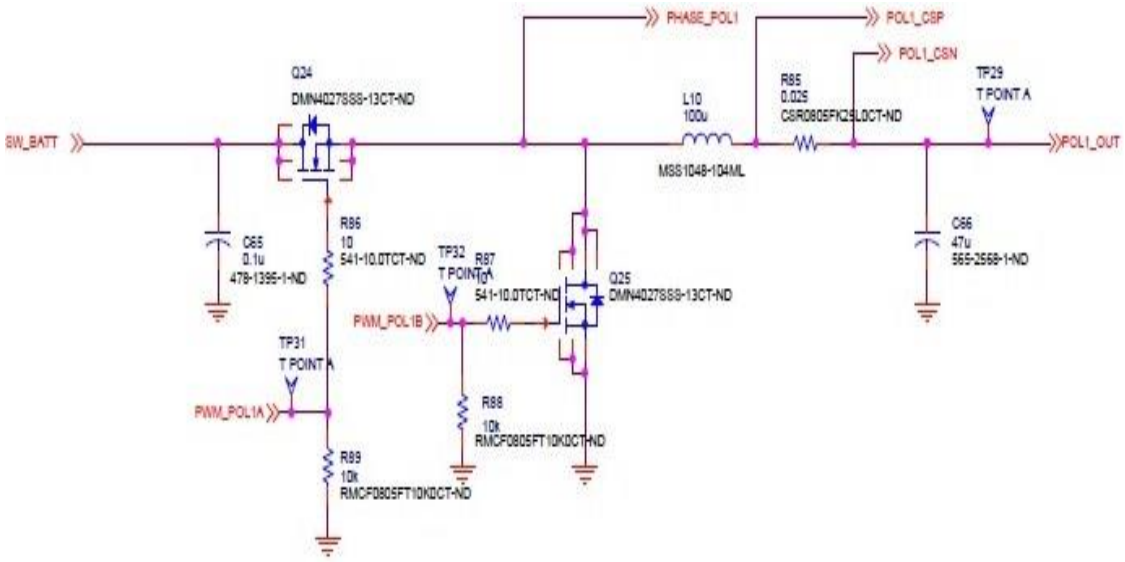


Figure 3.3: Schematic of Synchronous Buck Converter used in EPS[2]

[2]Electrical Power System (EPS) – Application in CubeSAT mission.

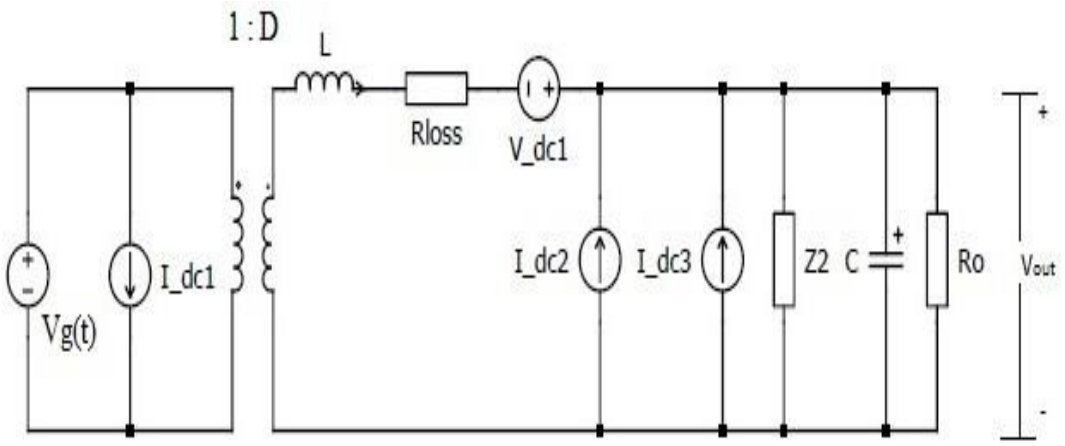


Figure 3.4: Equivalent Circuit of Synchronous Buck with Resistive Load

Table 3-1: Synchronous Buck Converter Parameter Definitions

Vg	Input Voltage		vg(t)	Input voltage perturbation
L	Inductance		d(t)	Duty cycle perturbation
C	Output Capacitor		R_{loss}	r_L+r_{DS} → Loss Components
Ro	Load Resistance		V_dc1	V_gd(t)
r_{DS}	MOSFET ON resistance		I_dc1	Id(t)
r_L	Inductor ESR		I_dc2	Vg r_C C d(t)/L
r_C	Capacitor ESR		I_dc3	D r_C C v_g(t)/L
D	Duty Cycle		Z2	L/(r_CC)
Vo	Output Voltage		I	Inductor Current

The following approximations were made during the modeling process.

$$\frac{R_{loss} r_C C}{L} \ll 1 \quad ; \quad \frac{R_{loss}}{R_o} \ll 1 \tag{3.1}$$

3.2.1.1 Characteristic Equation

The characteristic equation of the buck converter is derived first, using state space analysis. All the open loop transfer functions have a common characteristic equation, given by *P*. This

is responsible for determining the natural frequency and damping factor of the system, and therefore open-loop performance parameters such as rise time, settling time, over/undershoots steady state error etc. These parameters are then used to design a compensator which will improve their values in a closed loop condition.

A constant current load provides an ideal step transient and is typically used to evaluate the transient response of power supplies. The characteristic equation of the buck converter for a constant current load is approximated as:

$$P = 1 + (R_{loss}C + r_C C)s + LCs^2 \tag{3.2}$$

Whereas, for a Resistive load, R_0 is given as

$$P = 1 + \left(R_{loss}C + r_C C + \frac{L}{R_o}\right)s + LCs^2 \tag{3.3}$$

3.2.1.2 Transfer Functions

The derived small signal transfer functions show the system's response to changes in various parameters of the converter, such as output current (i_0), duty cycle (d), input voltage (v_g) etc. The parameters of interest are the output voltage (v_0) and inductor current (i_L). These can be represented in closed loop form to include the compensator transfer function Ds (Dz in z-domain). If Dz is used, the above transfer functions have to be converted to z-domain for discrete time representation.

i. Response to Changes in Output Load

The **output impedance, Z_{out},** of the converter determines the transient response of the output voltage, $V_{O,}$ to changes in the load current, I_o.

$$Z_{out} = \frac{r_C LCs^2 + (L + r_C R_{loss}C)s + R_{loss}}{LCs^2 + (R_{loss}C + r_C C)s + 1}; \tag{3.4}$$

ii. Response to Changes in Input Voltage

The **input-to-output** transfer function determines the transient response of the output voltage to changes in the input voltage.

$$G_{vg} = \frac{D\ (1 + r_C Cs)}{1 + (R_{loss}C + r_C C)s + LCs^2}; \tag{3.5}$$

These input transients may occur due to battery discharge or other loads drawing additional power from the same power source.

iii. Response to Changes in Control Signal

The **control-to-output** transfer function describes the transient response of the output voltage to a change in the control signal input.

$$G_{vd} = \frac{V_g\ (1 + r_C Cs)}{1 + (R_{loss}C + r_C C)s + LCs^2}; \tag{3.6}$$

The step responses of input-to-output control and control-to-output transfer functions are similar due to the same characteristic equation (3.2). This transfer function also helps in determining the frequency response of the system and the system bandwidth, $\boldsymbol{f_c}$. This is then used to find an appropriate switching frequency. The relation of the convertor bandwidth, $\boldsymbol{f_c}$ and the switching frequency, f_{sw} is given in [6] [5] is:

$$f_c < \alpha\, f_{sw}; \tag{3.7}$$

α was chosen to be greater than 5 to minimize noise in the output. But this can have an impact on MCU utilization (increasing it if too high), and on the switching ripple (as given in equation (1.6))

3.2.2 Digital Controller Design

There are many reasons for designing a digital controller instead of an analog one. Digital control allows for the implementation of more functional control schemes. They are less susceptible to noise and parameter variations. The boom in semiconductor industry has led to the availability of microprocessors at low costs. Digital controllers can be easily ported to new technology, which is fast evolving. Instead of having a dedicated hardware for digital control, implementation on an off-the-shelf microprocessor increases flexibility of changing the control algorithm, and further reduces cost.

Various design methods exists when it comes to designing a digital compensator. From here onwards, we will be using the terms ‘compensator’ and ‘controller’ interchangeably. They both represent the controller for closed loop operation of the SMPS converter. Martin [15]

has presented different techniques, widely used for digital compensator design. Duan and Jin [13] have evaluated some of these techniques. Some are indirect methods, in which a compensator is designed in the '***s***', or frequency, domain and then converted back to the '**z**' or discrete domain using approximations and transformations. The direct methods convert the continuous-time plant with ZOH and samplers using discrete time approximations. Here, 'plant' refers to the entity being controlled. In this case, it is a buck converter. The continuous-time plant is represented by its transfer functions. Once converted to z- plane, the compensator is designed using root locus techniques.

Compensator design knowledge was derived from various sources [12] [13] [14] [15] [17]. Since the buck converter is controlled using a PWM control signal, the second transfer function, G_{vd} is used for designing the compensator. The converter's stability has to be analyzed due to changes in duty cycle. Since the input-to-output transfer function, $G_{vg,}$ the output impedance, Z_{out}, and the control-to-output transfer functions, G_{vd}, all have the same characteristic equation; the compensator design would take care of perturbations that may be caused by all three parameters.

An overall closed loop system is depicted in Figure 3.5.

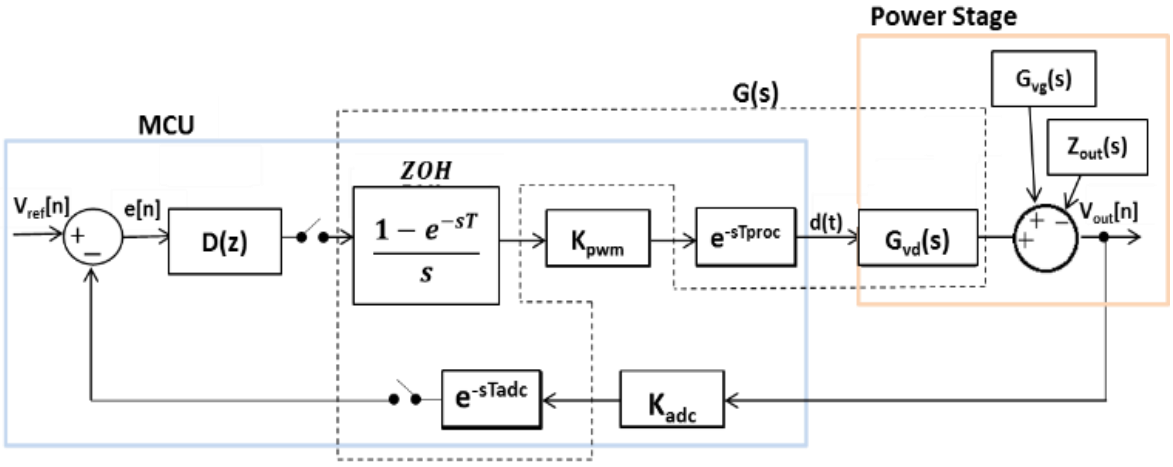


Figure 3.5: Block diagram of the closed loop system

The $G_{vd}(s)$ represents the plant's continuous-time transfer function, with G_{vg} and Z_{out} as perturbations, and $V_o(s)$ as output, a continuous time output voltage. The input to the plant is a PWM signal passed through a ZOH. The PWM signal is represented in z-plane as d[z], and d[n] in discrete time domain. Delay blocks are represented using $e^{-sTproc}$ and e^{-sTadc} in continuous time domain. T_{proc} corresponds to the delay in control processing and generation of updated PWM control signal. T_{adc} is the delay in sampling the output voltage by the Analog to Digital Converter (ADC) and providing it to the compensator. The gain blocks K_{proc} and K_{adc} are merely the scaling factors corresponding to the PWM generator and the ADC gain. They are useful in representing the system, and are compensated within the controller. Thus, we will not take these into account while determining the compensator transfer functions. Their values are determined as explained in [17] as,

$$K_{pwm} = \frac{1}{2^{n_{pwm}} - 1} \; and \; K_{adc} = \frac{1 - 2^{-n_{adc}}}{V_{max_{adc}}} \tag{3.8}$$

The switch in the feedback loop depicts the sampling action of the ADC. D(z) represents the compensator in the z-domain and in this case it is an MCU. This would also help in determining the effects of control loop frequency on the voltage regulation and the V_{margin}. The input to the system is represented by a discrete time reference voltage, $V_{ref}[n]$. This is stored within the compensator. Thus, the voltage is sampled at the output of the system and is compared with the reference voltage to generate a discrete time error signal, e[n] or e(z), in the z-plane. This error signal then goes through the compensating action of the controller and an updated control signal d(z) is generated as an input to the plant. The compensator transfer function is of the form,

$$D(z) = \frac{a_2 + a_1 z^{-1} + a_0 z^{-2}}{1 + b_1 z^{-1} + b_0 z^{-2}} = \frac{d(z)}{e(z)}; \tag{3.9}$$

This transfer function is then implemented in the controller using direct discrete time control law [25], as a discrete time difference equation,

$$d[n] = a_2 e[n] + a_1 e[n-1] + a_0 e[n-2] - b_1 d[n-1] - b_0 d[n-2]; \tag{3.10}$$

The compensator was designed using earlier mentioned techniques and a tool available as part of MATLAB toolbox, known as ***sisotool***. This tool allows us to analyze the root locus, frequency response and various other step, ramp, pulse etc. for a closed loop system. This tool also provides automated design technique in deriving the discrete time control equation of the form given in equation (3.9).

3.3 SMPS Transient Response

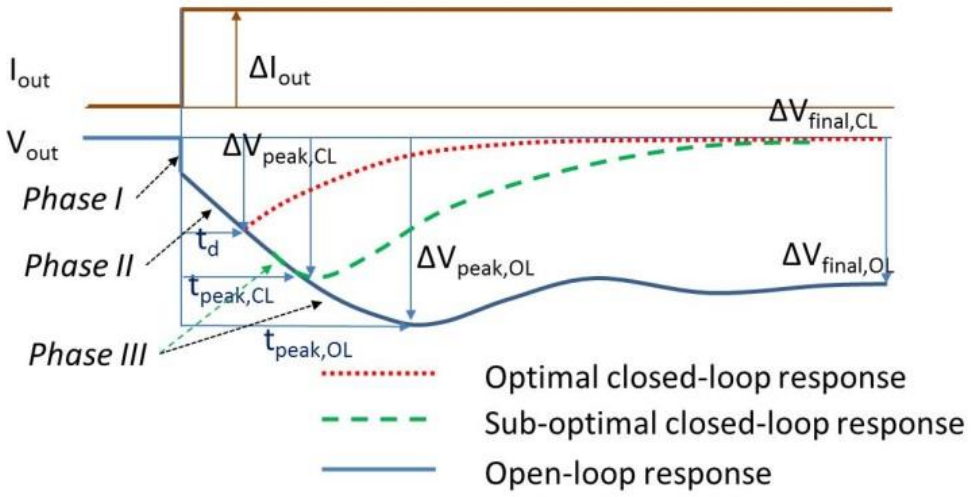


Figure 3.6: Detailed loading transient response for buck converter

First, the response of a buck converter to the step loading transient ΔI_{out} occurring at time $t = T_{step}$, is examined, as shown in Figure 3.6. Loading transients are of primary focus in this study, because unloading transients lead only to a minor increase in power and energy consumption and do not threaten proper circuit operation.

Understanding this response is the key to determining: 1) How far the voltage will fall initially (ΔV_{peak}); 2) When that peak voltage occurs (T_{peak}); and 3) Where the voltage will

finally stabilize (ΔV_{final}). These factors need to be considered in conjunction with the requirements of the devices powered by this converter.

In this analysis the system's response is examined without including switching ripple when possible since they are generally independent. Switching ripple affects both, voltages and times. The bounds on a given voltage can be determined by adding or subtracting the maximum ripple voltage of equation (1.6). The actual ripple voltage as a function of time depends on the switching phase in the PWM signal and RLC values. Ripple also affects time by introducing temporal sampling effects, shifting times by up to one switching period T_{sw} in either direction.

Two methods are presented for evaluating the response. The first method finds **bounds on possible performance**, and is useful for quickly creating a buck converter which is potentially capable of meeting our goals. The second method determines **actual performance** of the system (both open and closed loop) and shows how to design the compensator.

First we consider two extremes: the open loop response (the worst case) and then optimal closed-loop response (the best case). These two cases bound times and voltages of interest for the feasible responses.

A buck converter's transient response has been examined in detail and is summarized here [8] [9]. The output voltage drops in three phases, as shown in Figure 3.6[3]. Both open and closed loop buck converters have the same behavior in Phases I and II, because the controller does not affect the output until Phase III. In **Phase I**, the voltage drops proportionally with ΔI_{out} and the capacitor's ESR r_C.

$$\Delta V_{out\_1} = -r_C \Delta I_{out} \tag{3.11}$$

Phase II follows Phase I and lasts until the response delay $\boldsymbol{t_d}$ has passed. In this phase, the voltage drops further due to the load and circuit characteristics. This delay is non-zero because the buck converter can only start to respond to a loading transient when transistor Q1 switches on; until that time the transistor is off and cannot provide more current to the output.

[3] We are ignoring the minor initial voltage drop and recovery from the effective series inductance of the output capacitor.

This occurs at time T_{PWM+}. Similarly, the response to an unloading transient doesn't occur until Q1 turns off, at time T_{PWM-}. Thus we define $\boldsymbol{t_d}$ as follows:

$$t_{d,load} = T_{PWM+} - T_{step} \tag{3.12}$$

$$t_{d,unload} = T_{PWM-} - T_{step} \tag{3.13}$$

The bounds on the value of $\boldsymbol{t_d}$ for an optimal closed loop control will be examined further in Section 3.4.2.

During Phase II, the slope of ΔV_{out} depends on the rate at which current is drawn from the output capacitor.

$$\Delta V_{out\_2} = -t_d \frac{\Delta I_{out}}{C_{out}} \tag{3.14}$$

The response of the circuit in **Phase III** depends on whether the system is open-loop or closed-loop.

3.3.1 Worst Case Performance

Without feedback, in Phase III the buck converter essentially acts as an RLC filter with the addition of parasitic elements. The peak output voltage deviation occurs at the earliest, after ¼ of a period of the circuit's natural frequency. Adding in the ripple voltage may shift this time by up to one switching period:

$$T_{peak,OL} = \frac{\pi\sqrt{LC_{out}}}{2} \pm T_{sw} \tag{3.15}$$

This time can be treated as the worst case (smallest) scenario, which will increase with inclusion of circuit parasitic values and resistances. At that time ΔV_{out} approaches the limit:

$$\Delta V_{peak,OL} = -\Delta I_{out}\left(\sqrt{\frac{L}{C_{out}}}\right) \tag{3.16}$$

The final settling voltage depends on the increase in load current and the ESR of the switches and inductor:

$$\Delta V_{final,OL} = -\Delta I_{out}(r_{DS} + r_L) \tag{3.17}$$

The step response of the a buck converter was simulated in MATLAB using Zout transfer function (Figure 3.7) and compared with a circuit simulation in PLECS (Figure 3.8). Note that these are characteristic responses, when there is no input voltage given to the system (V_{in} = 0). The results of actual hardware implementation and simulation of hardware will be presented in the next chapter.

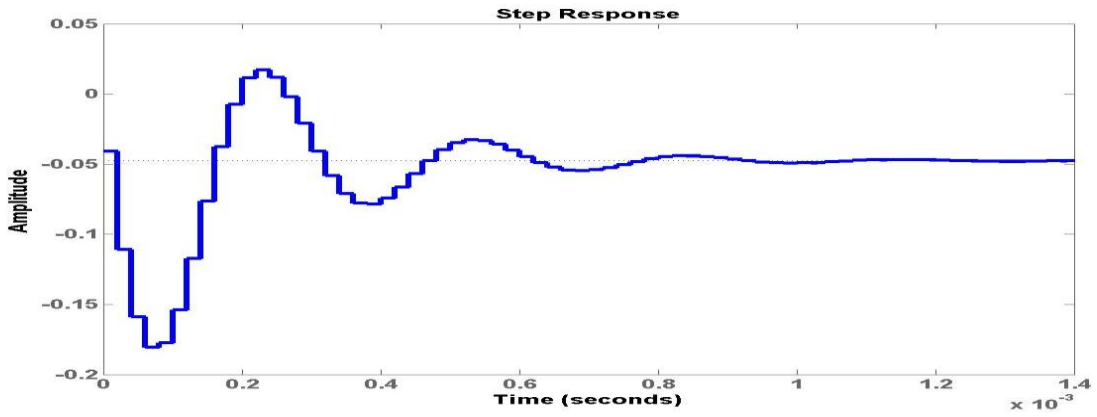


Figure 3.7: Step response of Buck in MATLAB

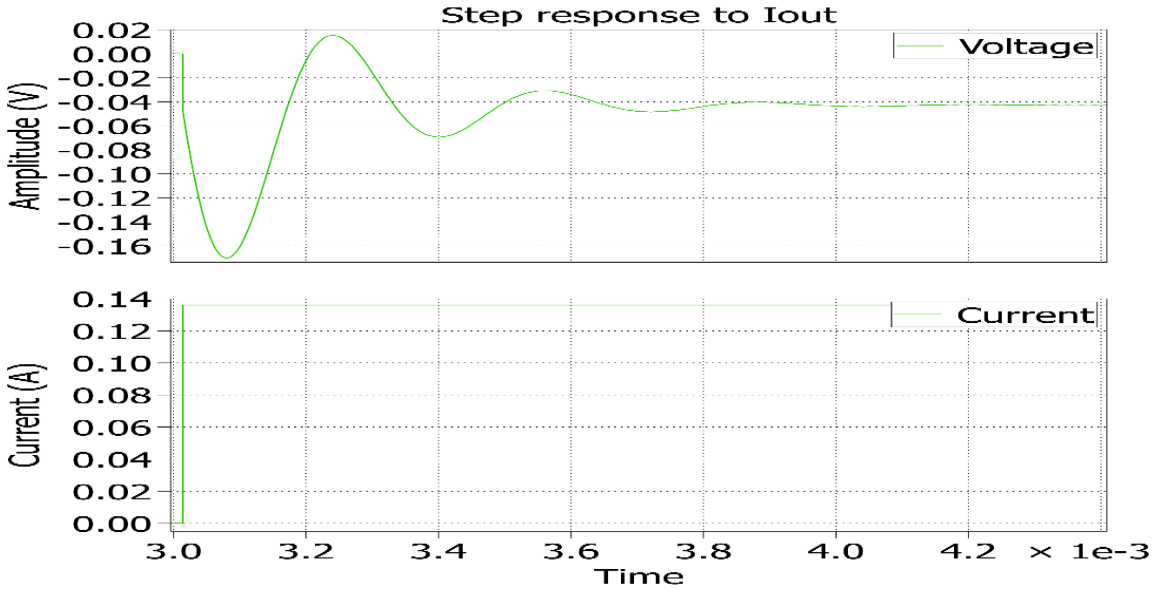


Figure 3.8: Step Response of Buck simulated in PLECS

The open loop circuit for a non-synchronous buck converter implemented in PLECS is shown below:

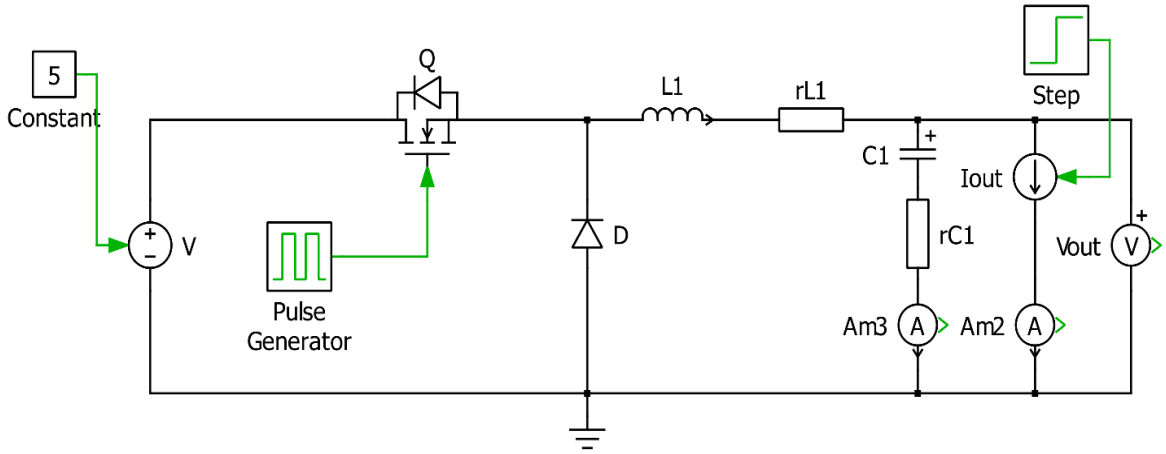


Figure 3.9: Open Loop Buck Circuit simulation in PLECS

3.3.2 Best Case: Optimal Closed-Loop Response

The Phase III response of the buck converter with optimal feedback control has been studied [8] [9]. For optimality, the controller response is assumed to be initially saturated. The minimum possible output voltage deviation as a function of time consists of the Phase 1 (eqn. (3.11)), Phase II (eqn. (3.14)) and Phase III responses:

$$\Delta V_{out}(t) = -r_C \Delta I_{out} - t\frac{\Delta I_{out}}{C_{out}} + (t - t_d) m r_C + (t - t_d)^2 \frac{m}{2C_{out}} \qquad (3.18)$$

Here, m is the slope V_L/L of the inductor current. For loading transients V_L is approximated to $(V_{in}\text{-}V_{out})$; for unloading transients it is approximated to $-V_{out}$.

Now, $\boldsymbol{V_L}$ is the voltage across the inductor at any point and is defined for a large signal as:

i. $V_L = V_{in} - V_0 - i_L(r_{DS} + r_L)$ → Loading of Buck Converter

ii. $V_L = -V_0 - i_L (r_{DS} + r_L)$ → Unloading of Buck Converter

iii. $V_L = V_{in} - i_L (r_{DS} + r_L)$ → Loading of Boost Converter

iv. $V_L = V_{in} - V_0 - i_L(r_{DS} + r_L)$ → Unloading of Boost Converter

where, i_L = inductor current; r_L = ESR of inductance

In most cases, it is assumed that $(r_{DS} + r_L)$ is very small and $i_L(r_{DS} + r_L) << |V_{in}-V_0|;|V_{in}|;|V_0|$
Therefore,

i. $\mathbf{V_L = V_g - V_0}$ → **Loading of Buck Converter**

ii. $\mathbf{V_L = -V_0}$ → **Unloading of Buck Converter**

iii. $\mathbf{V_L = V_g}$ → **Loading of Boost Converter**

iv. $\mathbf{V_L = V_g - V_0}$ → **Unloading of Boost Converter**

The size of the inductor relative to the *critical inductance value* (equation (3.19)) determines whether the output voltage will rise immediately or after an additional delay [5].

$$L_{crit} = \frac{r_C C_{Out}(V_{in} - V_{out})}{\Delta I_{out}} \tag{3.19}$$

A **smaller inductance** ($L<L_{crit}$) results in recovery of voltage immediately after t_d thus eliminating $\Delta V_{out}(III)$.

$$T_{peak,CL} = t_d \tag{3.20}$$

Equation (3.18) then reduces to:

$$\Delta V_{out} = -r_C \Delta I_{out} - t\frac{\Delta I_{out}}{C_{out}} \tag{3.21}$$

At that time, the output voltage reaches:

$$\Delta V_{peak,CL} = -\Delta I_{Out}\left(\frac{\Delta I_{out}}{C_{out}} t_d + r_C\right) \tag{3.22}$$

However, with a **larger inductance (**$L>L_{crit}$) the peak output voltage deviation occurs later:

$$T_{peak,CL} = \max\left\{t_d, t_d + \frac{\Delta I_{out} L}{V_L} - r_C C_{out}\right\} \tag{3.23}$$

It is straightforward to apply this time to equation (3.18) to determine the peak voltage for the closed loop case with a large inductor.

3.3.3 Exact Closed Loop response

The derived functions are inserted in the overall system model (as shown in Figure 3.5) in order to design the digital feedback control compensator function. The compensator transfer function is implemented in the controller as a discrete time difference equation using direct discrete time control law [12]:

$$d[n] = d[n-1] + 0.2033e[n] - 0.175e[n-1]; \tag{3.24}$$

This is computationally simple, requiring only two multiplies, one addition and one subtraction. These can be performed in fixed point math to avoid floating point operation overhead.

3.3.3.1 Response Evaluation

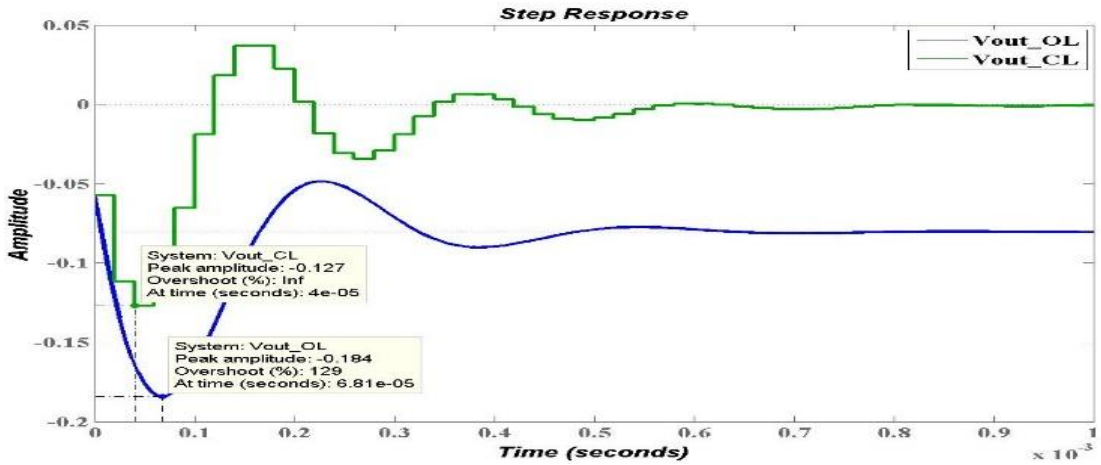


Figure 3.10: Step response of open-loop (lower trace) and closed-loop (upper trace) SMPS.

The system's response parameters can now be determined using MATLAB functions (*step, stepinfo* and *damp*). Figure 3.10 shows the response of the prototype buck converter's output to a step increase in load. Closed loop control dramatically improves the response, reducing

the peak error from -0.184 to -0.127 and eliminating steady-state error. The PLECS [31] simulation and actual hardware implementation will be discussed in the following chapter consisting of precise operating voltages, initial and final current states and exact values of voltage deviations, ΔV_{peak}, and peak time to deviation, T_{peak}.

We have also derived the continuous time equation for the buck converter's output voltage response to a step load transient as a function of the component values and time. The equation is complex and out of scope of this report (which focuses on tight bounds), and can be made available on request.

3.4 Real Time Analysis

An analysis of the real-time requirements on the SMPS controller software driven by the hardware characteristics are presented in this section. These can be applied either to the bounds or the exact response described in the previous section.

3.4.1 Schedulability Analysis

In Section 1.1 we explained why and how it is important to determine the real-time impact of embedding the SMPS control software in a real-time application MCU. One dimension already explored in this chapter is the impact of the software parameters like control loop frequency, $\boldsymbol{f_c}$, interrupt lockout and task blocking time, $\boldsymbol{t_b}$, on the SMPS hardware response and voltage regulation. We explore the second dimension in this section, which is the impact of same parameters and the WCET of the control task, on the schedulability of a system. Thus, as explained earlier, it is important to determine if all tasks in this low-cost system with limited hardware capabilities and shared resources will be able to meet all deadlines for correctness, performance and efficiency of the overall system. Here, crucial parameters are introduced and studied w.r.t. the schedulability analysis, such as WCET, MCU utilization, and WCRT. This allowed us to include control software into a real-time operating system (RTOS), be it pre-emptive (like µCOS-III, RTX) or non-pre-emptive (RTC). This further led to a prototype embedded system integration and experimentation results which assert the claims made in this work.

3.4.1.1 WCET, C_i

Worst case execution time is one of the fundamental parameters in helping us determine the bottlenecks in the software implementation of a particular task. It allows further real-time analysis in calculation of MCU utilization, blocking times for higher or lower priority tasks, and WCRT for all tasks.

An average execution time might not be same as the WCET. In order to obtain an accurate number, multiple runs and profiling is required to obtain a true worst case time. The execution time presented in Section 4.1.2.4 is the exact time of execution of the control loop and can be treated as WCET, C_i. This is because it does not have any software dependencies or conditional statements in the code. The control task is a sequential code with only one Basic Block.

3.4.1.2 Task Deadline/Period, T_i

In this system, the task period or deadline, T_i, is the same as control loop update period, $\boldsymbol{1/f_c}$. If the control loop is perfectly synced with the SMPS PWM, it would be same as the switching period, T_{sw} = 20µs. Sections 3.4.2.2 and 4.2.4 provide the effect of delaying the control loop update rate on the hardware response. T_i, is also crucial for schedulability analysis, as it sets the deadline for the tasks to complete execution, determines the utilization of MCU, WCRT of the task, and hence, the schedulability of the task set. Thus, an appropriate $\boldsymbol{f_c}$ can be determined to allow the tasks to be schedulable and at the same time maintain a certain hardware regulation.

3.4.1.3 MCU Utilization, U

MCU Utilization determines the percentage of time MCU spends in performing useful work. The MCU utilization of each task gives the percentage of MCU time it spends in performing operations of that particular task. This is an important parameter as it helps determine the active time of the MCU. With special power modes, it is possible to reduce energy consumption by placing the MCU into low power sleep/halt modes when it doesn't have any tasks at hand.

The utilization of a single task is defined as the ratio of its WCET, C_i and Period, T_i. The overall utilization of the MCU for the entire task set (m tasks) becomes the sum of individual utilizations and is given as:

$$U_{MCU} = \sum_{i=1}^{m} \frac{C_i}{T_i} \tag{3.25}$$

The utilization parameter for the task set also allows the schedulability of the task set to be determined, to some extent. For a rate monotonic scheduler (RM), the task set is definitely schedulable if $U < U_{max}$, where $\mathbf{U_{max} = m(2^{1/m} - 1)}$. It may or may not be schedulable if $U_{max} < U < 1$. For the latter case, a more detailed analysis involving WCRT is required, which will be explained in Section 3.4.1.4. For a scheduler based on earliest deadline first (EDF), the requirement for schedulability is simply $U<1$. In general for any scheduler, a set of tasks is not schedulable if $U>1$. This means there are bound to be tasks that will miss their deadlines, without completed execution.

In our study, the utilization can also vary due to the control loop update rate, $1/\boldsymbol{f_c}$. This is the period of occurrence of the control algorithm. Thus, if the control loop is delayed, the utilization will also decrease. A simple relation is given as:

$$U = \frac{T_{exec}}{T_{ctl}} = WCET \times f_C \tag{3.26}$$

This is further analyzed in Section 4.2.6.

3.4.1.4 WCRT

Worst case response time provides a more meaningful insight into a set of tasks' schedulability and individual task performance. Based on the type of scheduler (preemptive or non-preemptive), it determines the worst case time for a task to complete execution after it becomes ready for execution. This analysis includes any possible interference faced by a task by either higher priority tasks (preemption) or even lower priority tasks (RTC).

The WCRT for any task can be determined from the following two iterative equations.

$$R_i' = C_i + \sum_{j \in hp(i)}^{m} \left\lceil \frac{R_i}{T_i} \right\rceil C_j \tag{3.27}$$

$$R_i' = C_i + B_i + \sum_{j \in hp(i)}^{m} \left\lceil \frac{R_i}{T_i} \right\rceil C_j \tag{3.28}$$

Equation (3.27) refers to the preemption case and equation (3.28) to the non-preemption scheduler. Notice the presence of the term B_i in (3.28). This term takes into account the time spent in the blocked state because of a lower priority task. Thus, for worst case scenarios, B_i is the longest execution time amongst the lower priority tasks, and can be defined as:

$$B_i = max_{j \in lp(i)}\{\lceil C_j \rceil\} \tag{3.29}$$

The iterative analysis is performed until $R_i = R_i'$. This settling value gives the WCRT of that task. In case of preemption, WCRT of the lowest priority is calculated and if it is less than its deadline, Ti, then the entire task set is considered to be schedulable else, it isn't.

Determining the schedulability of the control task and the interference due to other lower priority tasks in the system using above analysis, the exact WCRT, and the blocking/interference time of the control task can be determined and applied to the analysis in Section 3.4.2.3 for t_b in order to estimate a more precise the hardware response.

It is worth noting that in a system with limited resources such as one ADC module shared amongst other tasks, may further lead to task interference. Figuring out the nature of blocking time due to such interference could also be incorporated in the above formula for evaluating WCRT.

3.4.2 Evaluating Timing Requirements

3.4.2.1 Sampling time within Switching Cycle

The timing relationship between the transient and the PWM cycle has a large impact on the delay until the system responds. The controller must perform an ADC conversion (which takes t_{sample}) and then perform the control law calculations and update the PWM duty cycle (which takes $t_{control}$).

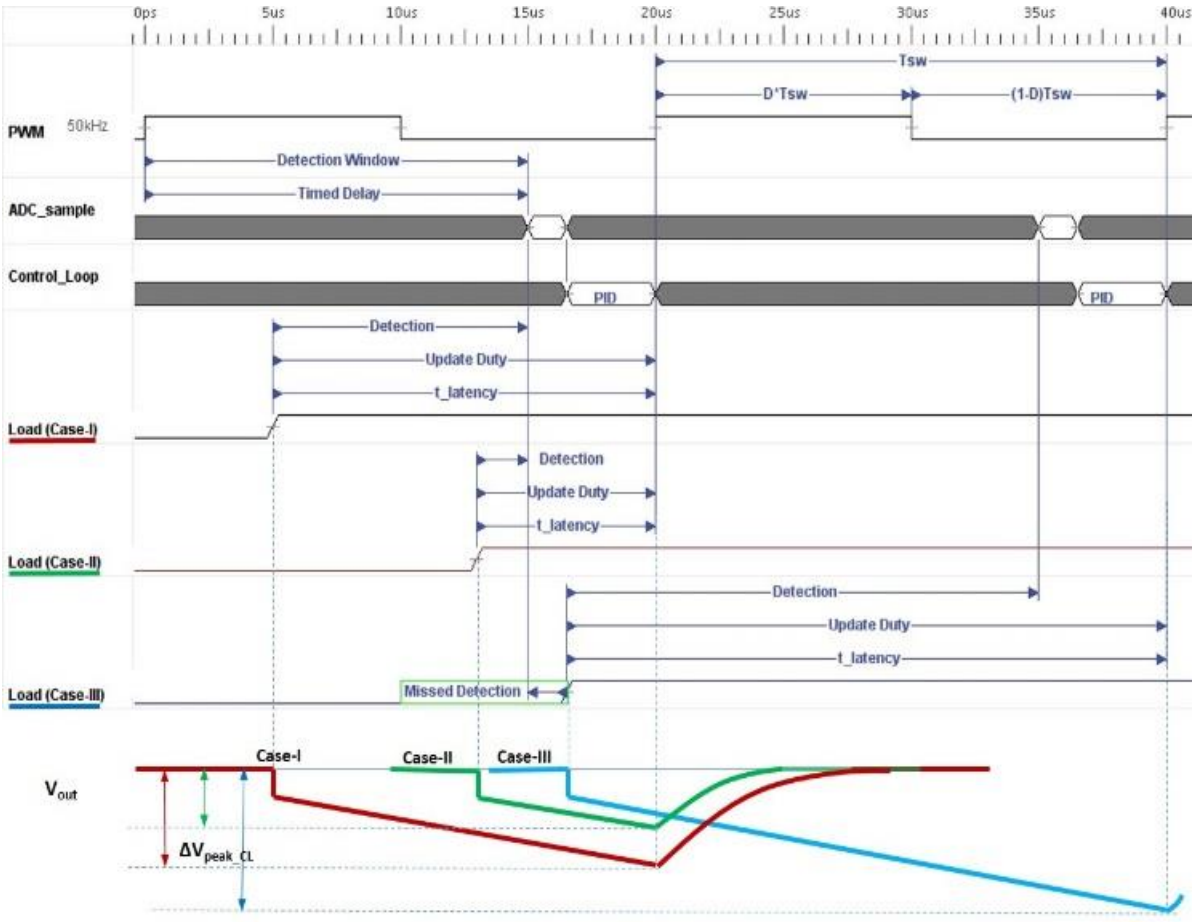


Figure 3.11: Timing sequence and voltage response with JIT sampling and different time of transient

It is a common practice to sample the output voltage on every positive edge of the PWM. But this can have serious impact on the control and can lead to delay in response to a load transient.

The duty cycle count should be available 'just-in-time' (JIT), before every positive edge of the PWM cycle, because the timer register is reloaded then as depicted in Case-I and II of Figure 3.11. Thus, if the voltage is sampled at every positive edge, the change due to the control will not be reflected until the next positive edge. Further, if a transient occurs, it will

not be detected until one switching period later, incrementally delaying the response by two switching periods.

Therefore, the sampling and the control loop processing should be delayed such that they occur 'just-in-time' before the positive edge, ensuring that the sampling and update will take place in the same cycle. The amount of delay depends on the ADC sampling time t_{sample} and the time required for control calculations (ADC ISR and PID) $t_{control}$. The signal indicating the sampling and control calculation is **ADC_sample** and **Control_loop** in Figure 3.11.

The impact of this delay is also depicted in succeeding sections. It is possible to estimate the probability, ***P()***, of detection and response due to a transient with-in one switching period for three sampling scenarios: i). Sample at positive edge of PWM; ii) Sample at negative edge of PWM; and iii) Sample and update just-in-time before the next positive edge of PWM.

$$P(i) \cong 0 \tag{3.30}$$

$$P(ii) \cong \frac{DT_{sw}}{T_{sw}} = D \tag{3.31}$$

$$P(iii) = \frac{T_{sw} - t_{sample} - t_{control}}{T_{sw}} \tag{3.32}$$

Equation (3.30) indicates that it is impossible to detect and compensate a voltage deviation in the same switching period. This is because if the ADC sampling is synced with the positive edge of the PWM, the transient will be detected at one edge and will be corrected ONLY in the following edge (when the duty cycle is updated in hardware next time around at positive edge).

Equation (3.31), on the other hand, intimates the window of opportunity when the transient occurrence might be detected and compensated in the same switching period. This is only possible when the transient occurs just before the ADC sampling begins (before negative edge), and after the current positive edge. Thus, the probability in that case becomes dependent on the duty cycle, as the window of opportunity is nothing but the positive width of the switching period (D*Ts).

P(iii) is a more efficient and probable way of detecting and correcting the transient in the same period. This is because if the ADC sampling is delayed such that the control loop is able to generate an updated duty cycle just in time for the next duty update in hardware, then the window of detection becomes larger and equal to $T_{sw} - t_{sample} - t_{control}$. As shown in Figure 3.11, Case-I and II fall in this category, whereas, Case-III is the case when the transient occurs during or after the sampling begins, delaying the PID to update the duty cycle by one switching period. The probability for the same is 1-P(iii).

Thus, looking at the above equations, it is evident that that the probability of responding to a load transient in less than a switching period is the highest for just-in-time sampling if the sample and control times are a small fraction of the switching period. $(T_{sw} - t_{sample} - t_{control})/T_{sw} > D$.

JIT: Just-in-time sampling allows us to determine the bounds on $\boldsymbol{t_d}$, discussed earlier. For a best case scenario, if a load transient occurs immediately before ADC sampling, the voltage can be detected and compensated immediately in the same switching period (similar to Load Case –II, but with shorter *t_latency* in Figure 3.11). $\boldsymbol{t_d}$ in this case becomes $t_{sample}+t_{control}$. Similarly, if the step occurs just after the sampling completes for the current cycles, the time for the system to respond would be limited to T_{sw}. So, for an optimal control with JIT sampling techniques,

$$\min(t_d) = t_{sample} + t_{control} \tag{3.33}$$

$$\max(t_d) = T_{sw} \tag{3.34}$$

It is evident from equations (3.18), (3.22), (3.33) and (3.34), that the ΔVpeak for open or closed loop becomes heavily dependent on $\mathbf{t}_d$ and in turn t_{sample} and $t_{control}$.

3.4.2.2 Control Loop Frequency

Next we examine the lower limits on the control loop frequency. Traditional SMPS controllers set the control loop frequency to be the same as the SMPS switching frequency in order to minimize the output transients. We remove this assumption in order to reduce the computational load and real-time requirements. The control loop runs at a frequency of $\boldsymbol{f_c}$, while the SMPS hardware switches at a frequency of $\boldsymbol{f_{sw.}} > \boldsymbol{f_c}$.

i. Switching Cycles until Peak Voltage

Traditional SMPS controllers set the control loop frequency to be the same as the SMPS switching frequency in order to minimize the output transients. This assumption has been ignored in order to reduce the computational load and real-time requirements. The control loop runs at a frequency of $\boldsymbol{f_c}$, while the SMPS hardware switches at a frequency of $\boldsymbol{f_{sw}}$.

Since a new duty cycle can only be updated every switching period, we redefine the time to maximum voltage deviation, T_{peak}, in terms of an integral multiple of switching period. Let the time for maximum peak voltage deviation for an open loop and closed loop condition be, $T_{peak,OL}$, and $T_{peak,CL}$, respectively. Following a transient, the maximum number of switching cycles it takes for the voltage to drop by ΔV_{peak} can be given as:

$$\alpha = \left\lceil \frac{T_{peak}}{T_{sw}} \right\rceil ;\ \alpha_{OL} = \left\lceil \frac{T_{peak,OL}}{T_{sw}} \right\rceil ;\ \alpha_{CL} = \left\lceil \frac{T_{peak,CL}}{T_{sw}} \right\rceil \tag{3.35}$$

It is worth noting that this value includes the voltage change during $\boldsymbol{t_d}$. Thus, for a best case scenario (JIT sampling), an optimal controller will recover the voltage within ONE switching cycle. In general, $T_{peak,CL}$ must be bounded by the following limits:

$$\boldsymbol{t}_d \leq T_{peak,CL} < T_{peak,OP} \tag{3.36}$$

For an optimal case, $\min\{T_{peak,CL}\} = \boldsymbol{t_d}$, which implies the duty cycle must be updated within the same PWM period of the transient. Thus, $\alpha_{CL} = 1 \rightarrow T_{peak,CL} = T_{sw} \rightarrow \boldsymbol{f_c = f_{sw}}$.

In case of other sampling scenarios (at every positive PWM edge), the lower bound on $T_{peak,CL}$ increases by at least two switching cycles, as it would take one cycle each for detection and response to transient. In this study, only JIT sampling scenarios are considered.

For an open loop and sub-optimal control, it will take maximum of α_{OL} and α_{CL} PWM cycles, respectively, for their respective voltages to reach their peak deviations.

Sub-optimality in the closed loop control may arise due to two factors: (a) An inherently slow compensator leading to small changes in the duty cycle; and (b) a slow control loop update rate implemented in conjunction with an optimal control algorithm (or duty cycle saturation). This would relax computational requirements on the microcontroller and allow

the voltage to drop beyond t_d, before the recovering. For case (b), $f_c = 1/T_{peak,CL}$. This implies, if an optimal control algorithm is allowed to run at a lower control frequency, $f_c < f_{sw}$, the voltage will be restored in the same cycle as the control loop runs and in the worst case, the maximum time of peak voltage deviation, $T_{peak,CL}$, would equal to the control loop update rate, $1/f_c$.

ii. Minimum Control Loop Frequency

What is the minimum control loop frequency f_c which will reduce ΔV_{peak} from the open-loop case? **JIT** ADC sampling is assumed. It is known that f_c cannot exceed f_{sw}. Now, in order determine the lower bound, the following cases are analyzed.

Case 1: Large α_{CL}, If $\alpha_{CL} >= \alpha_{OL}$ then the control loop will not affect the peak voltage ΔV_{peak}. Hence, to tolerate this larger voltage transient we will need to operate at a higher voltage, reducing power and energy savings accordingly.

Cases 2 and 3: Small α_{CL}, However, if $\alpha_{CL} < \alpha_{OL}$ then the control loop will affect the peak voltage ΔV_{peak}. Recall that L affects the delay between when the duty cycle D changes and when V_{out} starts to rise. For the fastest response, the inductor must be less than a critical value L_{crit} [5], as given by equation (3.19).

Note that we can also modify this equation to determine the critical values of capacitance, output current transient, or inductor voltage, based on which parameters can be changed.

The voltage drops by $\Delta V_{peak,OL}$ in α_{OL} cycles, and we wish to respond before that time. Therefore the duty cycle must be updated at least every α_{OL}-1 cycles. This follows the latest detection by control loop for JIT ADC sampling to be α_{OL} -1 cycles. Hence, α_{CL} is an integer from 1 to α_{OL} - 1:

$$\alpha_{CL} \epsilon Z[1\ , \alpha_{OL} - 1] \tag{3.37}$$

The range of useful control frequencies can now be defined as:

$$f_C\ = \frac{f_{sw}}{\alpha_{CL}} \tag{3.38}$$

$$f_C \epsilon \left[\frac{f_{sw}}{\alpha_{OL} - 1}\ , f_{sw}\right] \tag{3.39}$$

Case 2: Small α_{CL}, small inductor. If the inductor's value L is less than L_{crit}, then the output voltage will begin to ramp up immediately after $T_{peak} = T_{sw} * \boldsymbol{\alpha_{CL}}$, assuming optimal (saturated D) controller response.

Case 3: Small α_{CL}, large inductor If L is greater than or equal to L_{crit}, then the ramp up in output voltage will be delayed at most until $T_{peak,CL} = T_{\mathrm{sw}} * \alpha_{\mathrm{CL}} + \frac{\Delta I_{out} L}{V_L} - r_C C_{out}$ even with optimal controller response.

Note that L_{crit} can be modified by changing other parameters such as capacitance, inductor voltage or switching frequency.

3.4.2.3 Impact of Timing Interference

Some tasks in embedded systems may suffer long blocking times from higher-priority tasks or ISRs, or critical sections of code where interrupts are disabled. If the control loop code is not able to run soon enough (i.e. to complete before the next PWM cycle (T_{sw})), how does this affect the output voltage?

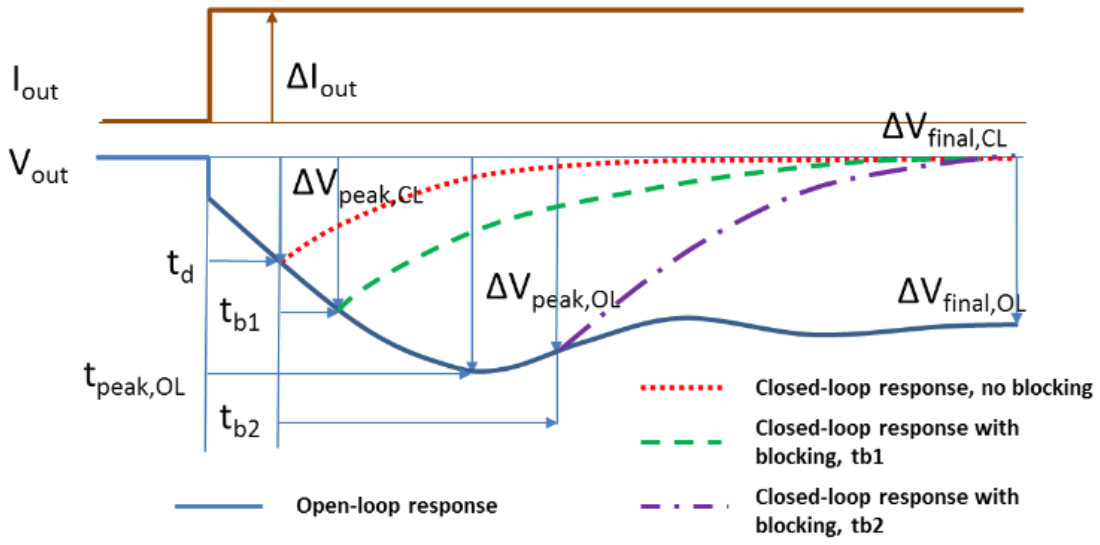


Figure 3.12: Transient response is affected by delay (t_{b1}, t_{b2}) in executing control loop.

Figure 3.12 shows the impact of delaying the control loop. The system follows open-loop response until the control loop executes. For example, if the control loop only completes at time t_{b1}, then the peak voltage transient is larger than the optimal closed loop response, but better than the open-loop response. However, a delay until t_{b2} is large enough that the deadline of the open loop peak has already passed and the control system can no longer reduce that peak.

To address this problem we begin by reducing the system's vulnerability to blocking. We design the system using the MCU's hardware peripherals to isolate operations from software scheduling and task blocking as much as possible. Hardware timers trigger the ADC sampling (at $\boldsymbol{f_c}$) and the PWM signal generation (at $\boldsymbol{f_{sw}}$). In addition, we rely on ADC data update features or use direct memory access to transfer ADC results, eliminating the need for ISRs to perform these operations.

The remaining processing is the control loop calculation, which is typically executed in an ISR to minimize blocking. However, it is still vulnerable to blocking when higher-priority ISRs execute, or when ISRs are disabled for data atomicity or other reasons.

The impact of blocking time t_{block} on the peak output voltage deviation is analyzed further in Section 4.2.3. Because the PWM output duty cycle is sampled at a rate of $\boldsymbol{f_{sw}}$, the possible time delays $\boldsymbol{t_b}$ are integer multiples of one switching period ($1/\boldsymbol{f_{sw}}$). Please note, this again considers **JIT** sampling to be implemented immediately after the blocking is removed, which allows the voltage to be compensated in the same switching period. Thus,

$$\boldsymbol{t_b} = T_{sw}\left\lceil \frac{t_{block}}{T_{sw}} \right\rceil \tag{3.40}$$

It is now straightforward to determine the maximum voltage impact from blocking for time duration $\boldsymbol{t_b}$ in equation (3.18) by replacing t with $\boldsymbol{t_b}$, with a maximum voltage deviation of $\Delta V_{peak,OL}$ if $\boldsymbol{t_b} > T_{peak,OL}$.

3.4.3 Optimizations to Tolerate Blocking

We next examine how to change the SMPS design in order to better fit the limited computational capabilities of low-end embedded microcontrollers.

There are various possible optimizations to increase the delay between the transient and the peak voltage, and therefore tolerate this blocking.

3.4.3.1 Capacitor Changes

The "brute-force" approach is to **increase the size of the output capacitor** [8]. Increasing C_{out} by a factor of x will reduce the slope of the output voltage by 1/x. It will delay the open-loop voltage peak by a factor of √x. Finally, it will reduce the peak voltage by a factor of 1/√x. However, this increases cost, volume and board area.

Using a capacitor with a lower ESR will reduce the voltage drop, therefore delaying the time until the voltage reaches a given level. For example, changing from tantalum to ceramic will reduce r_C, and therefore the initial voltage drop and voltage ripple (equation (1.6)). However, it will also reduce the critical inductance, making the system more likely to have a slower response even with optimal closed-loop control. Further analysis is performed in Section 4.2.2.

3.4.3.2 Inductor Changes

Changing the inductance slows both the open-loop and closed-loop responses.

Increasing the inductance increases the delay until the open-loop peak voltage. When the control loop finally updates the duty cycle, the larger inductor (greater than the critical inductance) will delay the ramp-up of the output voltage. In both cases the magnitude of that peak voltage increases.

Reducing the inductance raises the amount of ripple voltage, unless the switching period is also decreased proportionally (which will increase power losses). The maximum inductor current will also increase, potentially raising inductor costs. This may also lead the converter to operate in discontinuous conduction mode (DCM), further complicating control and converter design.

3.5 Aggressive Voltage Scaling

Raising the operating voltage will allow the system to tolerate a larger voltage ΔV_{peak}. However, this would also lead to a quadratic increase in the system power dissipation, and energy consumption. On the other hand, decreasing the 'set-point' will require a tighter control to avoid the voltage from dropping below the lower threshold. A simple experiment depicting the response time and transient on changing the set-point and a conceptual analysis is presented in Section 4.2.5.

3.5.1 Concept of Voltage Margin

In a voltage regulator, the function of a closed loop control is to maintain a constant output voltage irrespective of the transients that might occur. Ideally, the closed loop control should be fast enough to respond to any transients without any delay. This is not so for the real world scenario. Due to presence of circuit components, delays in signal propagation, error detection, computational delays and delay in plant's response causes the control loop to respond after a time $\boldsymbol{t_d}$. The delay leads to some deviation in the output voltage for some time, before it can be restored. This unavoidable deviation is an allowable margin for any closed loop operation. In this study, we call this the Voltage Margin or $\mathbf{V_{margin}}$, it is also synonymous to earlier introduced $\mathbf{\Delta V_{peak}}$. The value of this margin depends on how tightly the system is maintained within bounds before the restoration begins. Since the signal propagation, error detection and plant's response time are fixed for a system, only the computational limits for the compensator can determine the required V_{margin}. The concept of $\mathbf{V_{margin}}$ has been illustrated in Figure 1.3**.**

3.5.2 Scaling Operating Voltage

In a real time scheduling technique, since the SMPS control function runs periodically as a task, the task frequency, Worst Case Execution Time (WCET) and phase jitter determine the allowable voltage margin. Every electronic device operates reliably within a voltage range. Suppose the upper limit is given by V_{max} and the lower limit is given by V_{min}. Now, the desired operating voltage, V_O, is obtained by setting the V_{ref}, using the relation:

$$V_{ref} = \min\{ (V_{min} + V_{margin}), V_{max}\} \; for \; Loading \tag{3.41}$$

$$V_{ref} = \mathrm{V_{min}} \; for \; Unloading; => \; V_{margin} < V_{max} - V_{min} \tag{3.42}$$

(V_{min} + V_{margin}) is for the loading condition, when a voltage drop occurs due to increase in load and (V_{max}-V_{margin}) is for the unloading condition, if the load decreases. Please note that the V_{margin} is always positive in our calculations, as it represents a magnitude, irrespective of loading/unloading conditions. For our convenience, let us analyze loading conditions only, assuming the initial operating voltage, V_O (or V_{ref}) is determined for no load condition. Here, V_{ref} is the desired output voltage and V_O being the actual output voltage, assuming the gains K_{proc} and K_{adc} are compensated in the software, thus for calculation purposes they are assumed to be 1. The increase in load drops the voltage by $V_{margin.}$ Thus, it is best to fix V_{ref} as given in equation (3.41).

If a given load can operate within a large voltage range, i.e. V_{max} - V_{min} is large, the controlling task can run at a much slower frequency, allowing a larger voltage margin and hence, larger operating voltage, V_O. This in turn leads to higher power consumption. This also results in lower CPU utilization and allows other voltage domains to be added to the system, or even other applications to run on the same processor. In case there is no other task to run, the operating voltage of the load can be reduced aggressively while the task runs at higher frequency allowing smaller voltage margin. This would result in lower power consumption at the load. This is the principle behind aggressive voltage scaling (AVS). On the other hand, if the load's operating voltage has a narrower range its controlling task has to run at a higher frequency, with smaller voltage margin.

In order to demonstrate the effects of voltage scaling on a system which in turn is dependent on the control loop update rate, some experiments were carried out using a servo motor as a load to EPS's FDPOL and a constant current load on the RL78/G14 based converter, further explained in Section 4.3.2 and 4.2.4, respectively.

3.6 Power and Energy Analysis

A simple approach was employed in order to measure the power improvement of the proposed system over that of a conventional linear regulator based system. Three ultra-caps, totaling 3F capacity (1F each) were employed to power the system. The total energy available is constant for all test cases and is given by the simple equation (3.43)

$$E_{total} = \frac{1}{2} C \, (V_{init}^2 - V_{final}^2) \tag{3.43}$$

The ultra-caps are charged for a fixed amount of time to voltage $\mathbf{V_{init}}$ and then allowed to discharge by powering the system. The amount of time, $\boldsymbol{t_{discharge}}$, taken by the system to drop the voltage of the ultra-caps to $\mathbf{V_{final}}$ determines the rate of energy consumption or the average power, $\boldsymbol{P_{avg}}$ consumed by the system, given as:

$$P_{avg} = \frac{E_{total}}{t_{discahrge}} = \frac{1}{2} C \frac{V_{init}^2 - V_{final}^2}{t_{discharge}} \tag{3.44}$$

A comparison drawn is with respect to the energy and power consumption between the proposed methodology of real-time software control of SMPS and the linear regulator based system. A real-time embedded application operating off a linear regulator serves as a baseline implementation, whereas software control based single and multi-rail SMPS implementations are the test cases. The detailed results are discussed in Section 5.4.

An example system running off a single buck converter with 2.7V output, which is powered using a 1F ultra-cap, is depicted below.

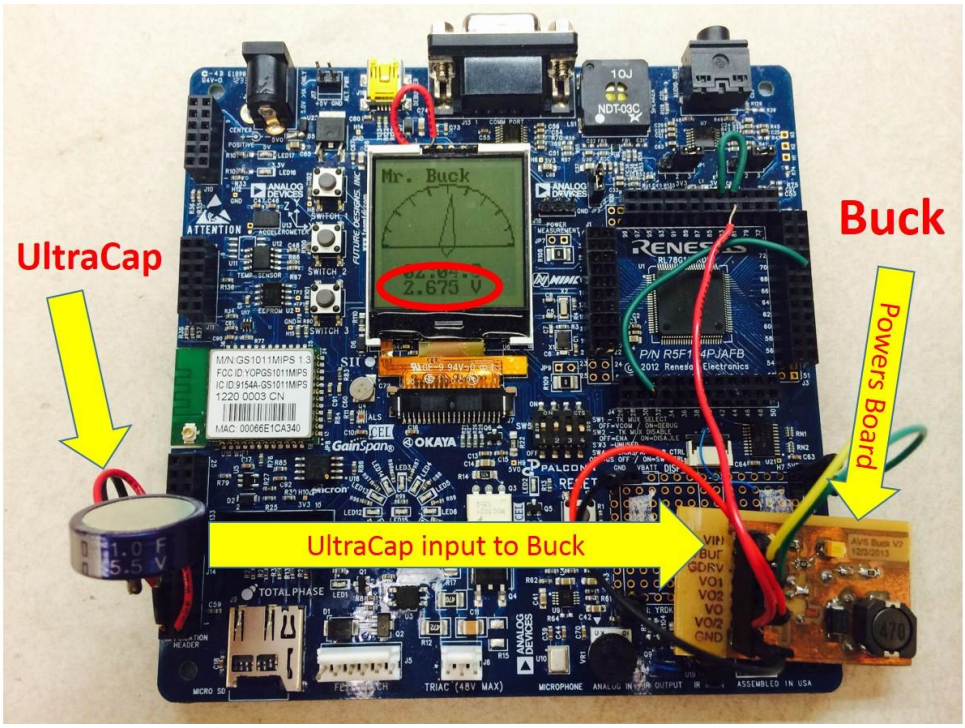


Figure 3.13: Standalone RDK powered by Buck with input from an ultra-cap

3.7 Converter Efficiency

The efficiency curve of the SMPS employed (Mr. Buck) is studied in this work for a wide range of loads. At static loads, the efficiency was measured and plotted as depicted in Figure 3.14. The operating duty cycle, D, of the converter is also increased with increasing load current in order to maintain the operating voltage.

It is observed that the efficiency varies by a great extent in different regions of the curve [7] [32]. The curve can be divided into three regions based on the shape and magnitude variation of the efficiency trends. A light load condition is assumed between 0 -25mA (blue region). A medium and heavy load ranges are highlighted with yellow and orange regions respectively.

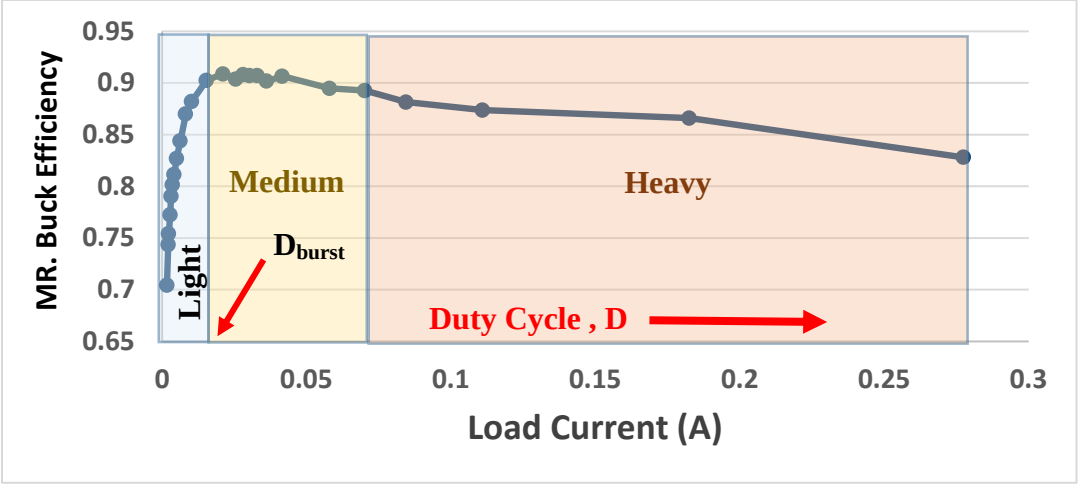


Figure 3.14: Non-synchronous Buck converter efficiency curve

The light load operation is dominated by the switching losses of the MOSFET, the circuit parasitics (like the ESRs of inductor and capacitors), and the forward voltage drop due to the diode. Whereas, in heavy load conditions, the switching losses are smaller as compared to the latter two losses, which are proportional to the square of the load current, I_{out} (α I^2_{out}) [7] [32]. Some of the parasitics for a synchronous buck are highlighted in Figure 3.15. For a non-synchronous buck, Q2 is replaced by a Diode, with a forward voltage drop, V_D ~0.7V.

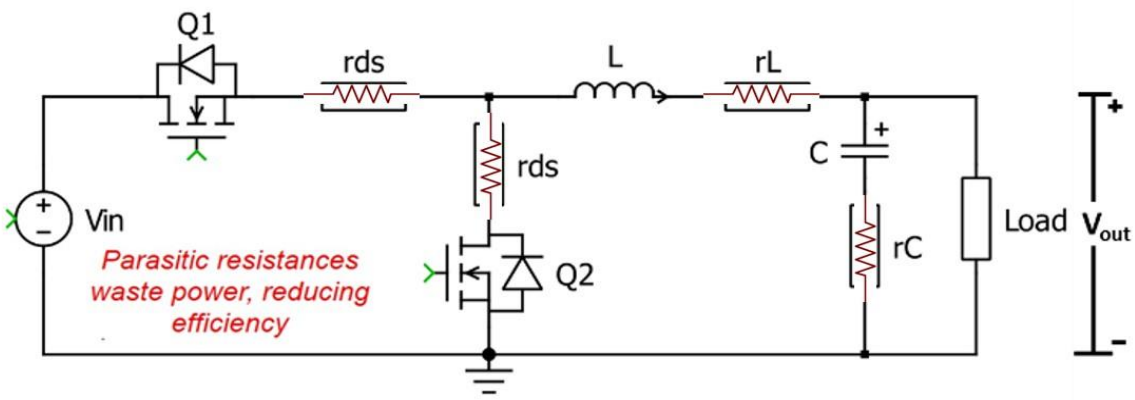


Figure 3.15: A synchronous buck converter with highlighted parasitic resistances

In this study, two methodologies are proposed which have proven to increase the efficiency in light and heavy load segments of the efficiency curve. First technique is deployed for light loads, and works on the principle of supplying energy in bursts so as to decrease the switching cycles. This is explained below in the following subsection in details. The second implementation is the use of synchronous buck converter, which results in higher efficiency in heavy load applications. The advantages and disadvantages of a synchronous buck are also discussed in Section 3.7.2. [33] [34].

3.7.1 Burst Mode Control

The principle behind the operation of burst mode control is to allow the control action to take place in bursts or periodic quanta of time, rather than every switching period. Due to continuous switching of the MOSFET, switching losses tend to dominate light load conditions. Since, the control loop in software is required to run at every switching period, this increases the overall utilization of the MCU, as well. In case of the burst mode, PWM signals are allowed to stop, thus halting the control action. This leads to interim drop in voltage. The voltage is continuously monitored and the control action comes back into play if the deviation exceeds certain percentage of operating voltage (voltage deviation). This triggers the PWM control action to resume. The burst mode operation is highlighted in Figure 3.16.

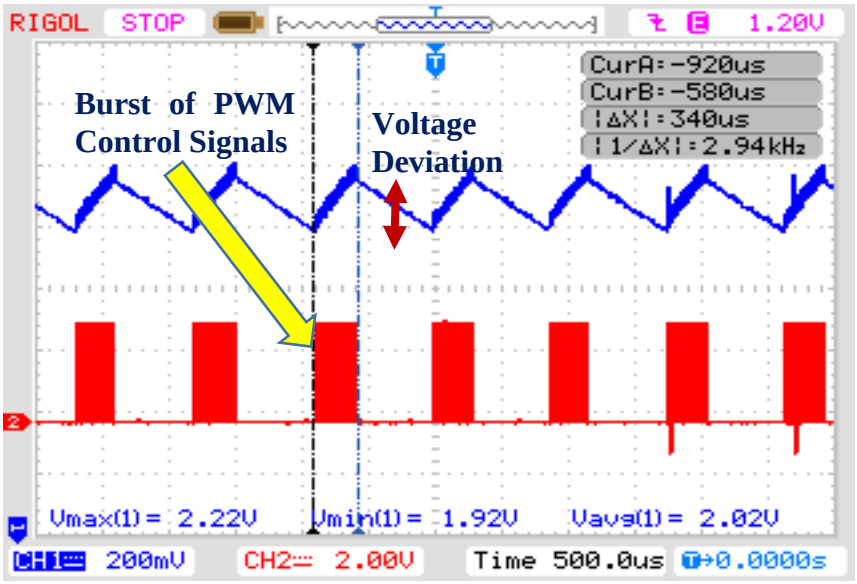


Figure 3.16: Burst Mode implementation of SMPS under light load conditions

The characteristic output voltage waveform (blue) is a resultant of the circuit parameters and operating point. The PWM burst (red) is enabled when the voltage in blue drops below a certain percentage of the set-point (operating voltage). The percentage margin is determined by the duty cycle corresponding to the upper bound of the light load range (as highlighted in Figure 3.14), and the maximum allowed *voltage deviation* for the load to operate without malfunctioning. The bounds on duty cycle in burst mode, D_{burst} are again determined by the segment of the efficiency curve, which need to be improved under light load conditions. For simplicity, D_{burst} is fixed at the boundary of light/medium load, as highlighted in Figure 3.14. A pseudo code implementing the burst mode can be written as:

```
if(V_deviation < V_set – V_buck)
        exit();
else
        new_duty = perform_control();
if (new_duty < D_burst)
        duty_cycle = 0;
else
        duty_cycle = new_duty;
```

As pointed out earlier, since, the PWM control action is not required at every switching period, the MCU does not incur computational load during the OFF time of the PWM signals. Hence, the MCU utilization is significantly reduced over a period of time.
The efficiency improvement will be discussed in Section 4.2.6

3.7.2 Synchronous Buck

As explained in Section 1.2 and 3.2, a synchronous buck converter has two MOSFETs switching, but are complementary to each other. In case of a non-synchronous buck converter, the lower MOSFET is replaced by a diode. There are various advantages and disadvantages between the two implementations. Some of which are discussed in by Nowakowaski [34]. Having a lower MOSFET (synchronous buck) helps eliminate the high voltage drop across the diode (non-synchronous), and increases the efficiency at high loads. The drop across the MOSFET is lower as compared to the diode in high load conditions. However, having two MOSFETS require complementary but synchronous PWM signals, which complicate the circuitry. Synchronous buck have to ensure that both transistors do not turn ON at the same time, resulting in catastrophic failure. In terms of cost, non-synchronous buck is cheaper than having a MOSFET with additional circuitry in case of synchronous buck. [4]Efficiency improvements in high load conditions by use of synchronous buck is highlighted in Section 4.2.6.2.

3.8 Control Software

There were various versions of the control software: 1) An MCU controlling a simple buck converter which powers an independent load, 2) An MCU controlling a buck converter which powers the same rail as the MCU itself, and 3) An MCU controlling various buck converters to power several loads including the MCU.
The difference lies in the sampling and the reference voltages used for ADC sampling.

[4] The efficiency measurements on synchronous buck were made by Zhi Qu, an MS student working under the guidance of Dr. Alex Dean in the ECE department at North Carolina State University

In the first case, where the MCU is not being powered by the controlled buck, but by a constant voltage source, the ADC uses the MCU voltage (V_{MCU}) as a constant reference, V_{ref} (=V_{MCU}), and the sampled ADC value, ADCR, represents the sampled voltage, V_{buck} (varying buck voltage). The formula used in this case to calculate the buck output/sampled voltage, V_{buck}, using an ADC with n-bit resolution and a potential divider to divider with ratio 1/r:

$$V_{buck} = \frac{ADCR}{2^n - 1} V_{MCU} * r = K * ADCR \tag{3.45}$$

Here, V_{buck} = K*ADCR, where K is a constant value which helps in reducing the computation of V_{buck}. The same is highlighted in Figure 3.17.

Whereas, in the second and third scenarios, since the rail voltage powering the MCU itself is to be controlled, the V_{MCU} cannot be treated as a constant voltage. In this case, if a constant known voltage can be sampled (V_{ref} = constant) and V_{MCU} is treated as a varying voltage, then V_{MCU} = V_{buck}, and the formula becomes:

$$V_{buck} = V_{MCU} = V_{ref} \frac{2^n - 1}{ADCR} = \frac{K'}{ADCR} \tag{3.1}$$

The implementation of a stand-a-lone MCU powered off the SMPS is shown in Figure 3.18.

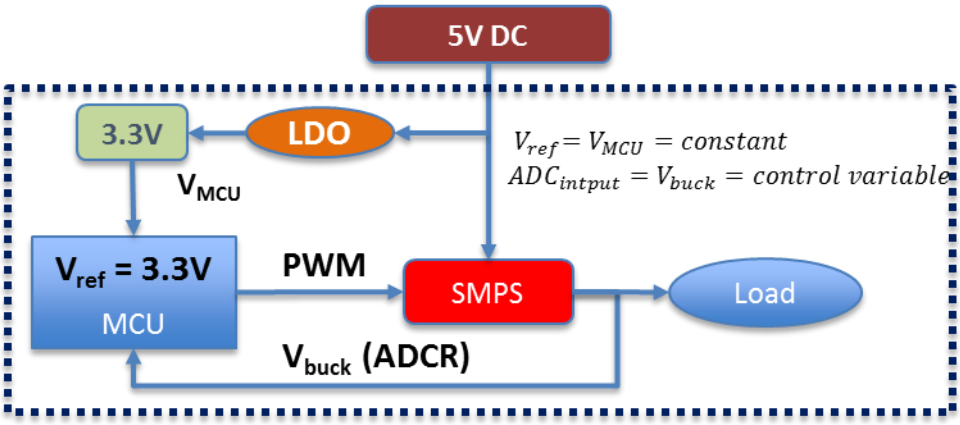


Figure 3.17: SMPS controlled by MCU with constant reference, V_{MCU}

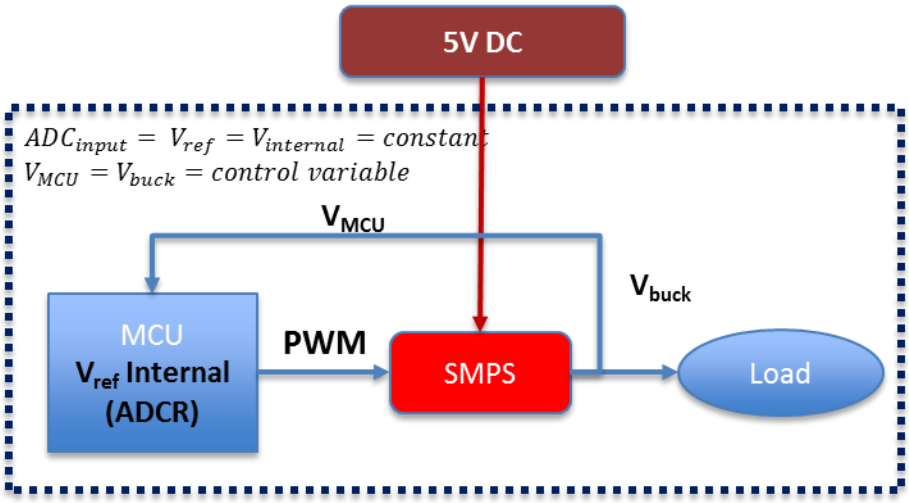


Figure 3.18: SMPS controlled by MCU with internal constant reference

Depending on the MCU, there is an internal constant voltage available and linked to an ADC channel. Also, it is easier on most MCUs to optimize multiplication than division, based on the architecture. Thus, it's faster to compute equation (3.45) than (3.1) on most MCU architectures.

3.8.1 Code Analysis

An evaluation version of a commercially available software tool, IDA Pro, was used to analyze the SMPS control software for the ARM Cortex M0+ MCU. "*IDA Pro combines an interactive, programmable, multi-processor disassembler coupled to a local and remote debugger and augmented by a complete plugin programming environment.*" [35]. It was employed as a dissembler to analyze the object code produced by the compiler targeting certain processors. In this study, IDA pro was used to create Control Flow Graphs (CFGs) of the SMPS control software. This helped in determine the longest execution path in the control software in terms of the assemble code. The instructions in the assembly code were then studied for their execution times. Thus, worst-case execution time (WCET) was calculated for the overall control code, along with execution times for other paths. The execution times were then verified in hardware using a scope to measure the execution times

of the SMPS control software. The CFG and further code analysis are presented in Section 0. Tool used for code analysis

3.9 Cost Analysis

To present a fair comparison of cost between i) linear regulators, ii) commercially available switching converters, and iii) the proposed switching converter design, Texas Instruments (TI) Webench was employed [36]. TI's Webench is a tool used to design, analyze, and customize power designs including power converters (linear or switching), based on commercially available components. Some of these converters can also be simulated on this tool for their characterization and analysis. A screenshot of the tool is provided below.

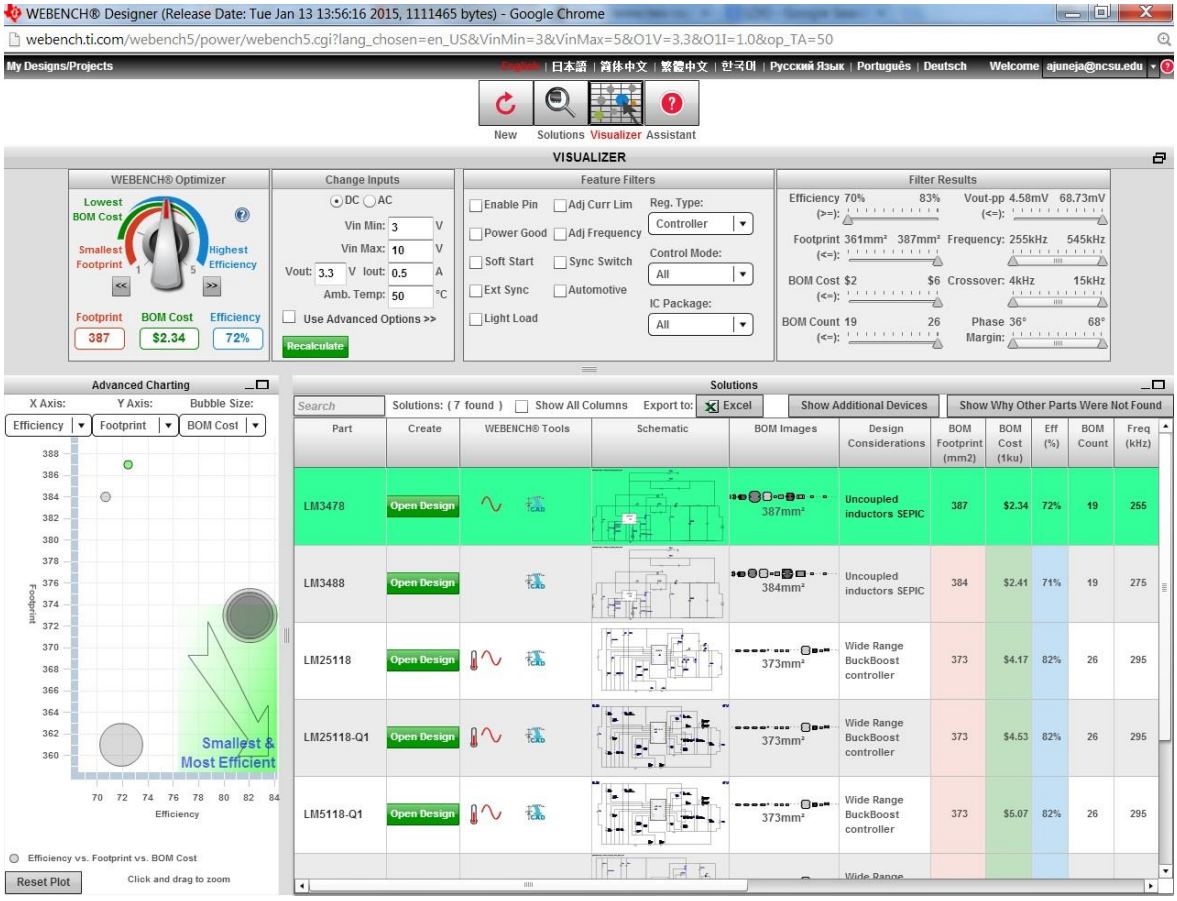


Figure 3.19: TI WEBENCH home screen - screenshot

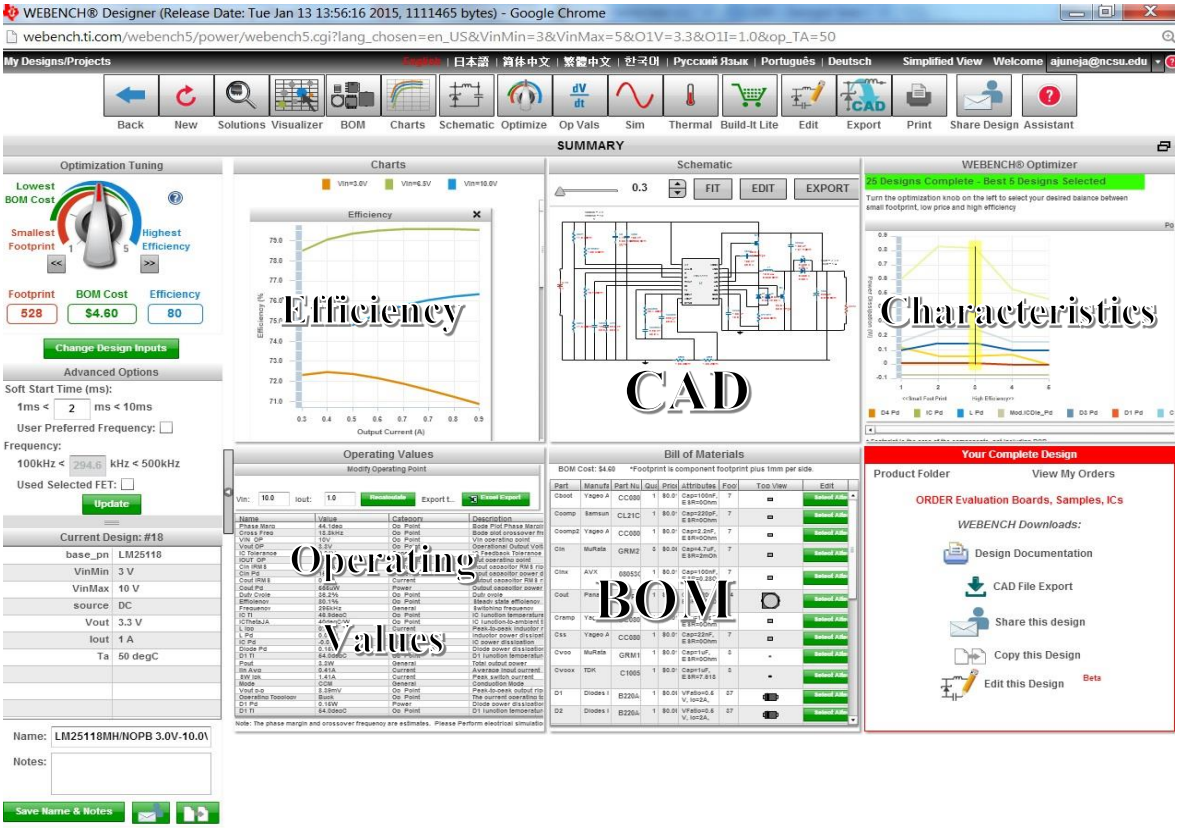


Figure 3.20: TI WEBENCH Converter Design and Analysis Tools Screen

It also provides detailed bill of materials (BOM) for each commercially available product.

In this work, the commercially available buck controller ICs and Linear Dropout regulator (LDO) complementary to our system's requirements were studied. This helped in drawing a fair conclusion in terms of cost of power conversion for the proposed design. The components required for the buck converter using the commercial controller IC, was treated as a common design point for the proposed SMPS design. In the proposed design, the cost of controller IC was eliminated by the use of application MCU, which is anyway present in all three cases. However, the linear regulator (LDO) replaces the buck converter, and the controller IC.

A detailed cost analysis is presented in Section 5.5.

Chapter 4

Experimental Setup

This chapter discusses the hardware and software involved in the study and analysis of the SMPS designs. In Section 4.1, the three platforms are discussed in detail and their control software implemented. Section 4.1.1.3 starts off with a brief introduction of the software sequence implemented on the Electric Power Supply Board for a CubeSat Application and Section 4.1.2.4 and Section 0 goes into detailed discussion on control sequence implemented on low cost MCUs, the Renesas RL78/G14 and ARM Cortex M0+ based Freescale's KL25Z, which led to detailed analysis of SMPS design and development cost. Experiments and hardware implementations are presented in Section 4.2, which provides the proof of concept for this study.

4.1 Development Platforms

Initial analysis and development of SMPS control was carried out on an indigenous Electric Power Supply (EPS) board. The EPS board is designed specifically for managing power on a CubeSat. Due to CubeSat's design and power requirements, a high-end Texas Instrument's, TI floating point DSP, F28335, was employed.

Due to the complexity of the EPS board, the lead time for development and debugging was significantly longer. For simpler embedded applications, where the voltage domains do not have a wide range of load requirements and the input voltage is within a smaller range, an off the shelf microcontroller can be used with limited computational capabilities. It was also observed that the difference equation implemented in the processor requires simple calculations, with limited memory requirements. Thus, two simpler processors: 1) Renesas' 16bit MCU, RL78 based G14 board, and 2) Freescale's ARM Cortex M0+ based FRDM board without floating point hardware was later employed for this study. Some approximations were be made while performing the calculations. Use of fixed point math

also improved the execution time to around 1.26µs on RL78/G14 and 2.32µs on KL25Z. Concept of digital controller design using look-up tables, implemented in hardware, has been proposed in [16]. Such designs can be used for their software counterparts for improved and faster control as well.

In the following sections, applications, capabilities, hardware design, and control software sequence implemented on the two platforms are discussed in detail.

4.1.1 EPS – Dedicated control

The bulk of the activity with the EPS has been getting the board up and running. We started with a first version of a PCB which had been designed but not tested. Mihir Shah (MS Graduate) and Shikhar Singh (MS Graduate) worked on debugging the power circuitry and writing the basic DSP firmware to operate the system in open-loop fashion. This foundation was then used for integration of the DSP firmware into the RTOS-based SYS/BIOS software platform, and applying and evaluating control methods using PID controller theory.

4.1.1.1 Motivation

The motivation of the architecture used for reliable implementation of the system is from Electrical Power System (EPS) used for the processing and distribution of power in the CubeSat. It takes in power from strings of Solar Panels, stores it in Li batteries and distributes it to various points of load (subsystems).

They should meet size and weight constraints, and should be scalable to various types of CubeSats (1U – 12U). Also, controller should be running the MPPT algorithm (also implemented by Mihir Shah and Shikhar Singh) to extract maximum power from the solar panels along with battery charging and monitoring algorithms, besides the supply voltage generation for the point of loads.

4.1.1.2 Architecture

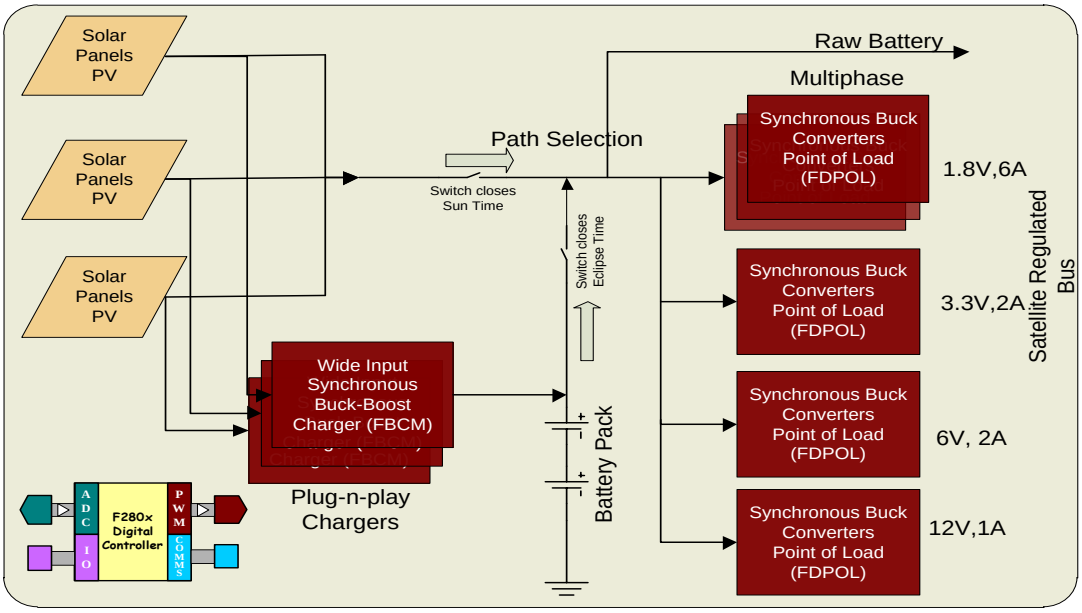


Figure 4.1: Power distribution architecture of Cubesat EPS

The architecture has multiple Flexible Battery Charging Modules (FBCMs) which are Wide Input Synchronous Buck Boost Converters. These converters can be connected in parallel to scale the system to higher ratings. They extract the power from the solar strings and deliver it to batteries connected in the system. The batteries power the Flexible Digital Point of Load (FDPOL) Converters which are Synchronous Buck or Synchronous Boost converters. These FDPOL converters generate the required voltage and current for the load circuits (payloads). All the converters in the system are reconfigurable on-the-fly to accommodate changes in the system and can keep the system at the optimal power point.

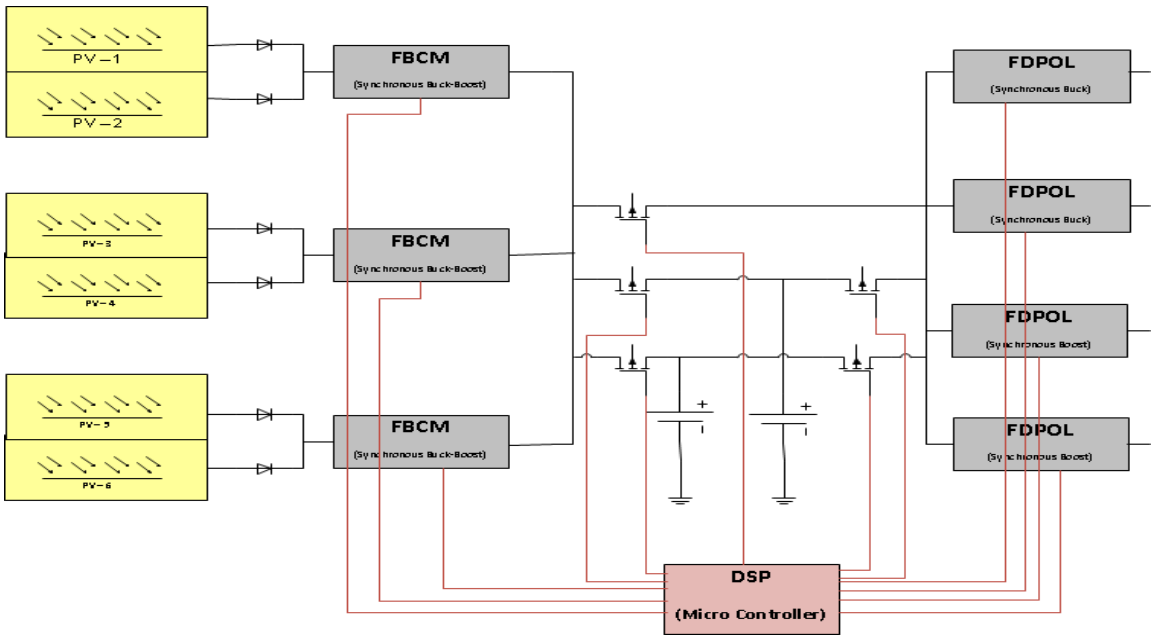


Figure 4.2: Control architecture of CubeSat EPS

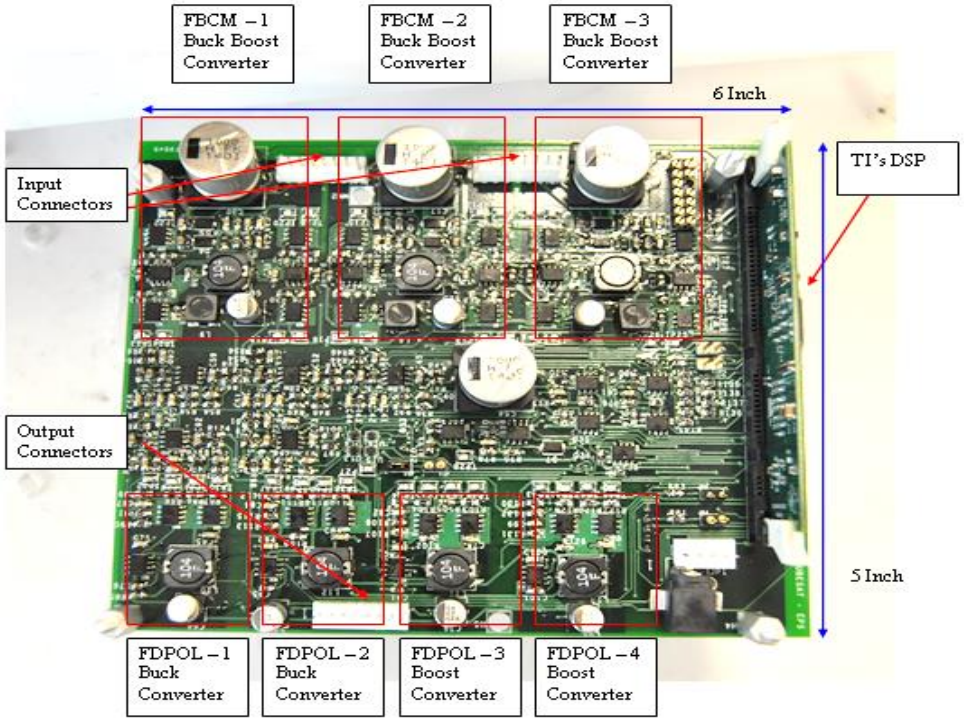


Figure 4.3: CubeSat EPS board with labeled subsystems.

Currently, the architecture implemented has three Flexible Battery Charging Modules (FBCMs) which can be interfaced with three different Solar Panel strings. FBCMs generate the required range of voltage and current to interface number of batteries. The Flexible Digital Point of Load (FDPOL) Converters regulates the power delivered to the points of load from the unregulated DC bus from the batteries.

The FBCMs, as shown in figure, perform the interface between the input power (solar panels) and the energy storage devices (Li-ion Batteries or Ultra Capacitors). Each FBCM is a synchronous Buck converter followed by a synchronous boost converter. At a given time, either buck or boost is used while the other acts as a closed switch to pass the current. A wide range of voltage (3V – 30V) can be generated from the modules. This enables batteries with different specs to be interfaced with the modules.

The FDPOL modules are synchronous buck converters and synchronous boost converters. They generate the voltage rails to connect the loads at a voltage higher/lower than the battery voltage. Currently the system has two buck and two boost FDPOL converters. These can be reconfigured to generate different voltage and also can be turned ON/OFF to save power.

The schematic of the Buck converter under study is given in Figure 4.4 and the parameters are tabulated in Table 4-1.

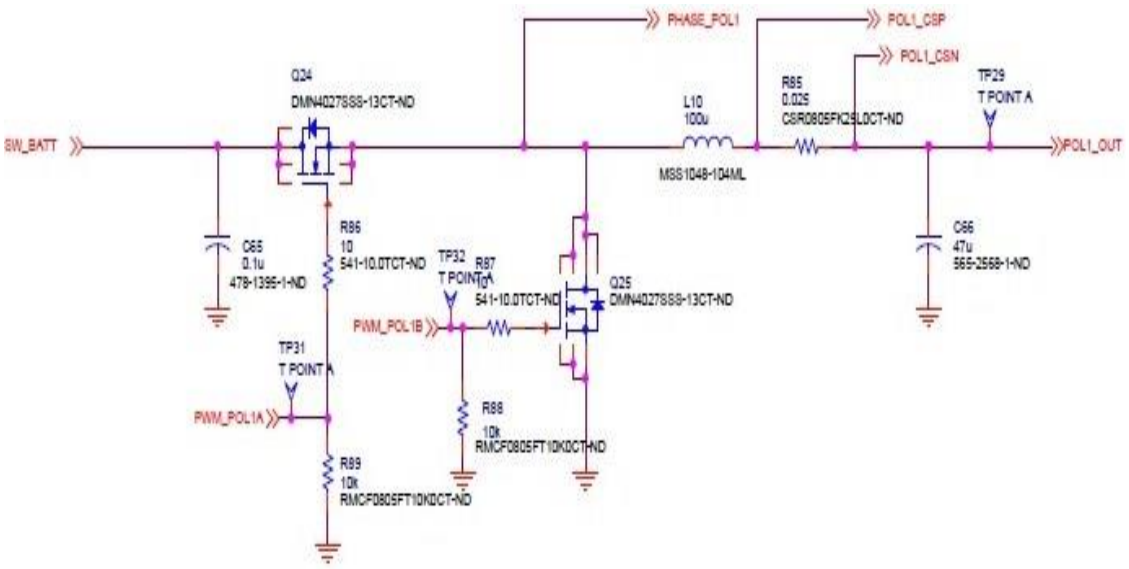


Figure 4.4: Buck Converter Schematic for EPS

Table 4-1: EPS Buck Converter Parameters

Vg	10V	Input Voltage
L	100µH	Inductance
C	47µF	Output Capacitor
Ro	10Ω	Load Resistance
r_{DS}	0.027Ω	MOSFET ON resistance
r_L	0.025Ω	Inductor ESR
r_C	0.6Ω	Capacitor ESR
D	--	Duty Cycle
Vo	--	Output Voltage
I	--	Inductor Current

The digital controller senses the voltage and current at each module, interfaces with the battery health monitoring system and controls the power modules. The controller also

executes MPPT algorithm to keep the solar panels to maximum power point voltage. The controller can have equivalent circuit models of the batteries which are used to determine the state of the charge of the batteries. Thus it makes the power system flexible for various energy storage devices and payloads.

The architecture has a path selection feature where the path of the power flow can be selected using the switches at various stages. The power flow can be enabled from the input to the batteries or to the point of load converters. This enables the charging of the batteries when there is a less demand of power at the payload or to provide more power along with battery when the demand is high.

The prototype has a pluggable micro controller interface with 100 pin DIMM connector which enables testing of the system with various classes of micro controllers. Currently, the implementation is being tested with TMS320F28335 for the software control of the devices.

4.1.1.3 Control Software for Cubesat EPS

Software development for the CubeSat EPS has followed two parallel tracks. The first track focused on bringing up the board and providing basic power converter interfacing (completed by Mihir Shah), while the second track involved creating a closed-loop power converter control system built upon an RTOS.

The first track used a simple single-threaded program enhanced with interrupts in order to provide basic converter monitoring and control. Features include analog to digital conversion of voltage and current inputs, timer control for PWM signal generation and serial communications to simplify debugging and related development tasks.

The second track is built upon a preemptive RTOS in order to support future integration of closed-loop control into real-time applications. TI provides SYS/BIOS, an advanced RTOS for its family of processors. Features include preemptive multitasking, synchronization, instrumentation, hardware abstraction, and memory management.

The EPS uses TI's TMS320F28335 DSP for the control of various voltage domains present on the board. This is a 150 MHz, 32-bit microcontroller with floating point hardware for faster and complex calculations. The integrated Development Environment (IDE) used for

software development and real time debugging is the Code Composer Studio 5.0. It is an Eclipse based environment, but specific for TI products.

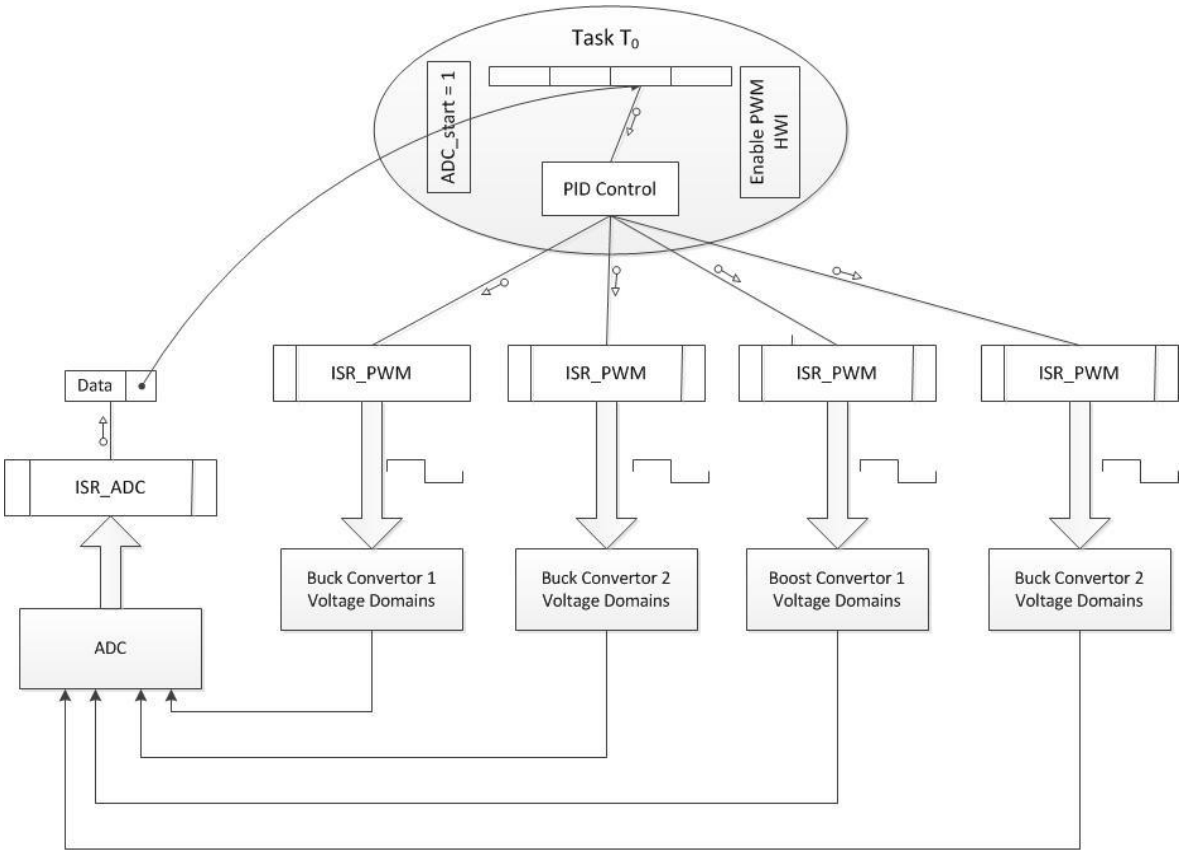


Figure 4.5: EPS Power converter control software overview.

The software architecture is depicted in Figure 4.5. There exists a single task, T_0, responsible for the control of voltage domains on the EPS. Within the task, there are various functions which control separate voltage domains. These domains can be made independent using individual tasks, instead of one controlling task. The tradeoffs of this segregation are yet to be explored.

There are two Hardware interrupts, HWI_{adc} and HWI_{pwm}, which are enabled once, each time, T_0 runs. The ISRs hooked to HWI_{adc} and HWI_{pwm} are responsible for collecting the ADC samples and updating the PWM duty cycles, respectively. The PWM's duty cycle controls output voltage of the respective voltage domain. The ISR execution time, T_{exec} is 2.5µs without any optimizations

The processor has only one ADC unit but is capable of performing sequential conversions using multiplexed input lines and storing their corresponding results in their respective result registers. Thus, for each Voltage domain, if separated by individual tasks, each task would need to re-initialize the conversion sequence before enabling the HWI for the ADC. For now, it is assumed that the control task accommodates all domains in one task, which enables the conversion of all required channels in sequence and storage of results in respective result registers.

At the beginning of execution, T_0 collects the ADC samples from various power domains (voltages from Buck, Boost and Buck-Boost, and currents from various parts of the board), by enabling HWI_{adc}. These samples are then used for control calculations and updating the PWM duty cycles of their respective domains, by enabling the $HWI_{PWM.}$ The HWIs are disabled within their respective ISRs to avoid further interruption, until the next occurrence of T_0.

The maximum operating frequency of this processor is 150MHz and the PWM frequency is kept at 150 kHz. The Task frequency of T_0 is currently 6 kHz (period of 170µs). This is when there is no other task present.

4.1.1.4 PLECS implementation

PLECS implementation of the closed loop EPS FDPOL Buck with a discrete time feedback controller is presented in Figure 4.6.

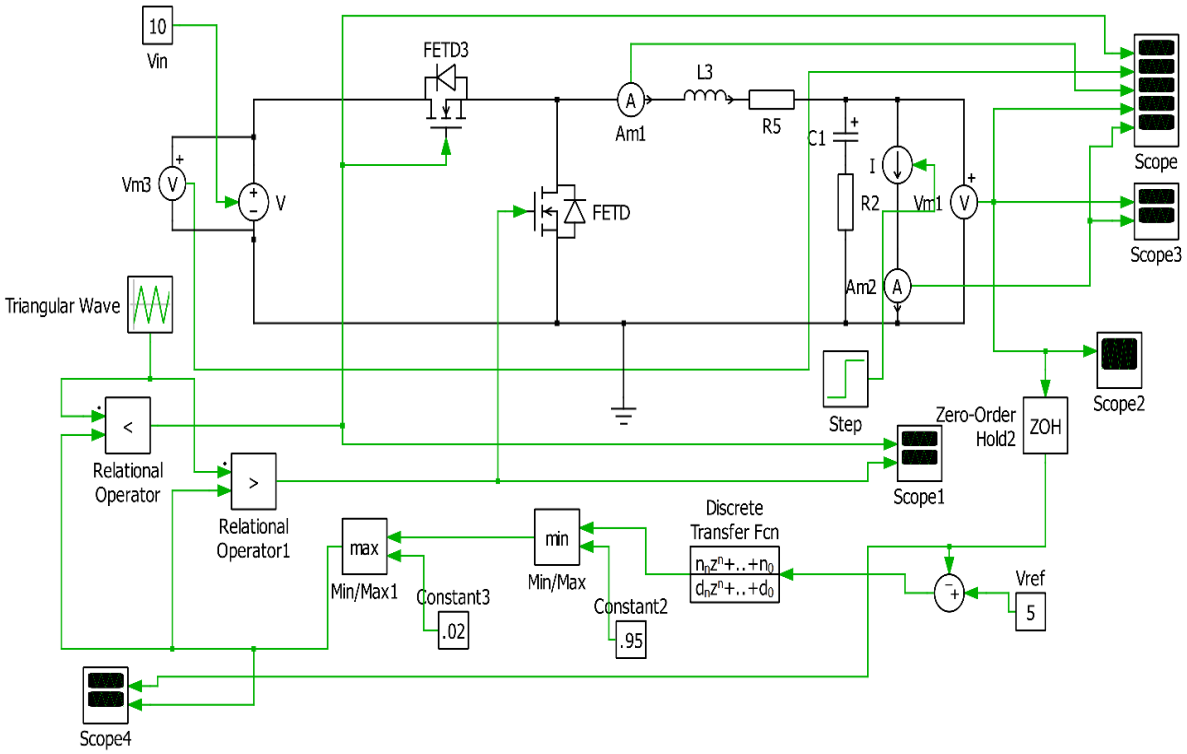


Figure 4.6: EPS Buck with discrete feedback voltage control in PLECS

The figure below compares the open and closed loop responses of the buck when loaded with a step current transient. The closed loop is the trace in dark green. The open loop response is given in light green, with a larger undershoot and steady state error.

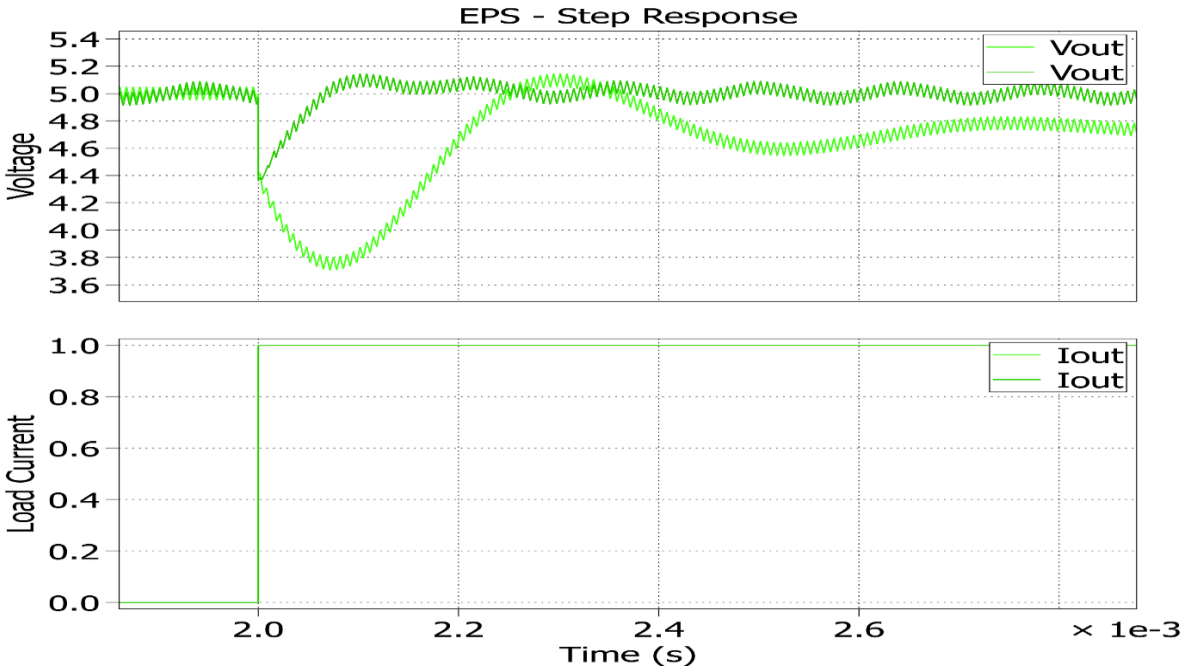


Figure 4.7: EPS Buck Step Response; Open Loop (Light); Closed Loop (Dark)

4.1.2 Low Cost Implementation

As mentioned earlier, the complexity of the EPS board leads to longer development and debug time. For simpler embedded applications, where the voltage domains do not have a wide range of load requirements and the input voltage is within a smaller range, an off the shelf microcontroller can be used with limited computational capabilities. It was also observed that the difference equation implemented in the processor requires simple calculations, with limited memory requirements. Thus, we demonstrate and validate our concepts using a microcontroller development board and a simple, low-cost non-synchronous buck converter. The target system is a microcontroller development board which is powered from a USB connection, a lithium ion cell or another 3.5V to 5V source.

4.1.2.1 Prototype Buck Converter

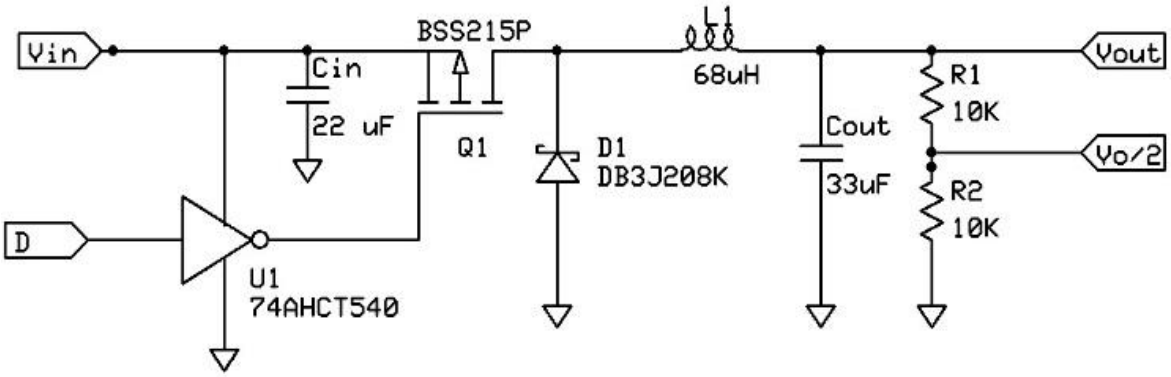


Figure 4.8: Schematic of prototype buck converter

Figure 4.9: Photos of prototype buck converter circuit boards.

The buck converter (shown in Figure 4.8 and Figure 4.9) provides a lower operating voltage to evaluate the impact of voltage scaling. The maximum output current required is roughly 300 mA.

Table 4-2: Buck Converter Components

ID	Description	Value	Part Number	Cost (@10k)
Q	**P-Channel MOSFET**		**BSS215P**	**$0.12**
D_i	**Schottky Diode**		**DB3J208K0L**	**$0.10**
L	**Inductor**	**68µH**	**SCRH105R-680**	**$0.38**
C_{Out}	**Output Capacitor 1**	**33µF**	**TPSB336K016R0350**	**$0.29**
C_{In}	**Input Capacitor**	**22 µF**	**CL05B103KP5NNNC**	**$0.002**
U1	**Octal Inverter**		**SN74AHCT540PWR**	**$0.18**

Component details are presented in Table 4-2. Two additions were made to enable the MCU to operate at lower voltages than the input or output. First, inverter U1 is added to buffer the transistor Q1 gate drive signal D to appropriate voltage level. Second, resistors R1 and R2 form a voltage divider to allow the MCU to monitor the output voltage safely.

Table 4-3: Buck Converter Parameters

Term	Value	Description	Term	Parasitic Value
L	**68µH**	**Inductor**	**ESR r_L**	**0.201Ω**
C_{out}	**33µF**	**Output Capacitor 1**	**ESR r_C**	**0.350Ω**
	22µF || 10µF	**Output Capacitor 2**	**ESR r_C**	**<0.005Ω**
		MOSFET	**ESR r_{DS}**	**0.120Ω**
V_f	**0.6V**	**Diode**	**ESR, r_f**	**$f(V_f)$**
R_{loss}	**$r_L + r_{DS}*d + r_f*(1-d)$**	**Effective parasitic resistance**		
C_{in}	**0.01 µF**	**Input Capacitor**		
V_{in}	**5V**	**Input Voltage**		
f_{sw}	**50 kHz**	**Switching frequency**	**T_{sw}**	**20µs**

4.1.2.2 Target Platform – RL78/G14

The target platform hardware is a development kit (RDKRL78G14) which is built around an MCU of the Renesas RL78G14 family.

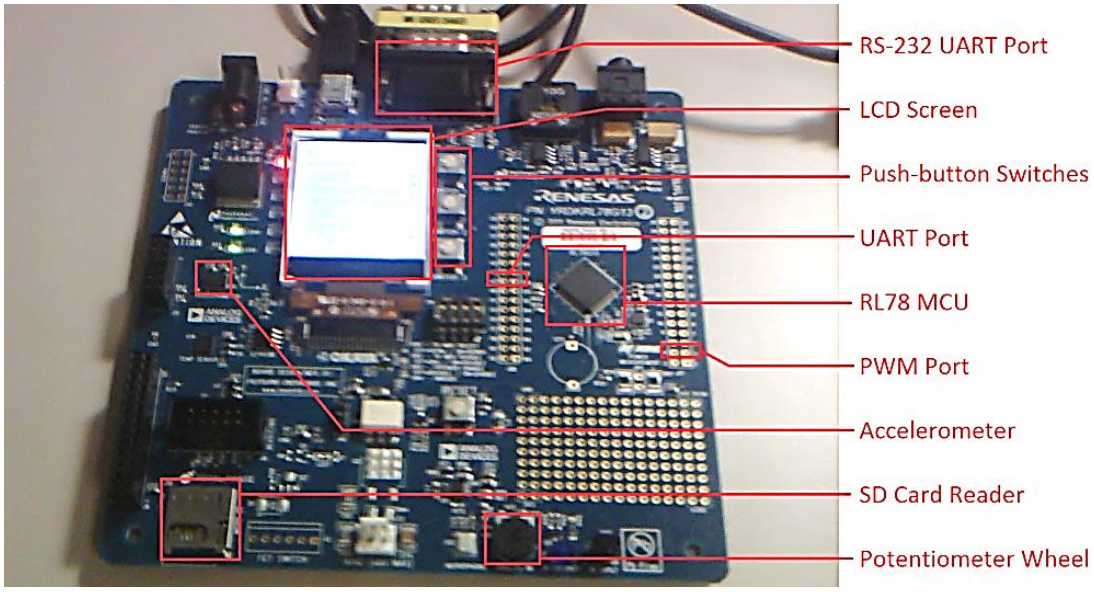


Figure 4.10: Renesas Development Kit (RDK) with RL78 MCU

4.1.2.3 Microcontroller – RL78/G14

The RL78 MCU family features a 16-bit CISC core with a three-stage pipeline. RL78G14 CPU cores run at up to 32 MHz and implement optional RL78 integer math instructions for higher performance, including 16x16 multiply, 16x16 multiply/accumulate (with a 32-bit accumulator), and both 16/16 and 32/32 bit divides.

The target system's microcontroller R5F105PJAFBfeatures 24 KB of SRAM and 256 KB of Flash ROM on-chip, and runs at supply voltages of 1.6 to 5.5 V. Energy consumption is quite low, ranging from 273 pJ/cycle at 4 MHz to 400 pJ/cycle at 32 MHz Power is similarly low, from 287 µW at 32 kHz to 12.8 mW at 32 MHz. For this application the processor runs at 32 MHz.

4.1.2.4 Software – RL78/G14

The majority of the code was written in C with a small amount of intrinsic operations to use the multiply-accumulate (MAC) instruction (***__mach()***). Peripheral configuration and interfacing code was generated with Applilet 3 for RL78/G14. The code was compiled with the IAR Embedded Workbench for RL78, version 1.30. Code optimization was set to maximum speed, and the source code was optimized with iterative tuning based on the object code.

i. Buck Converter Controller Implementation

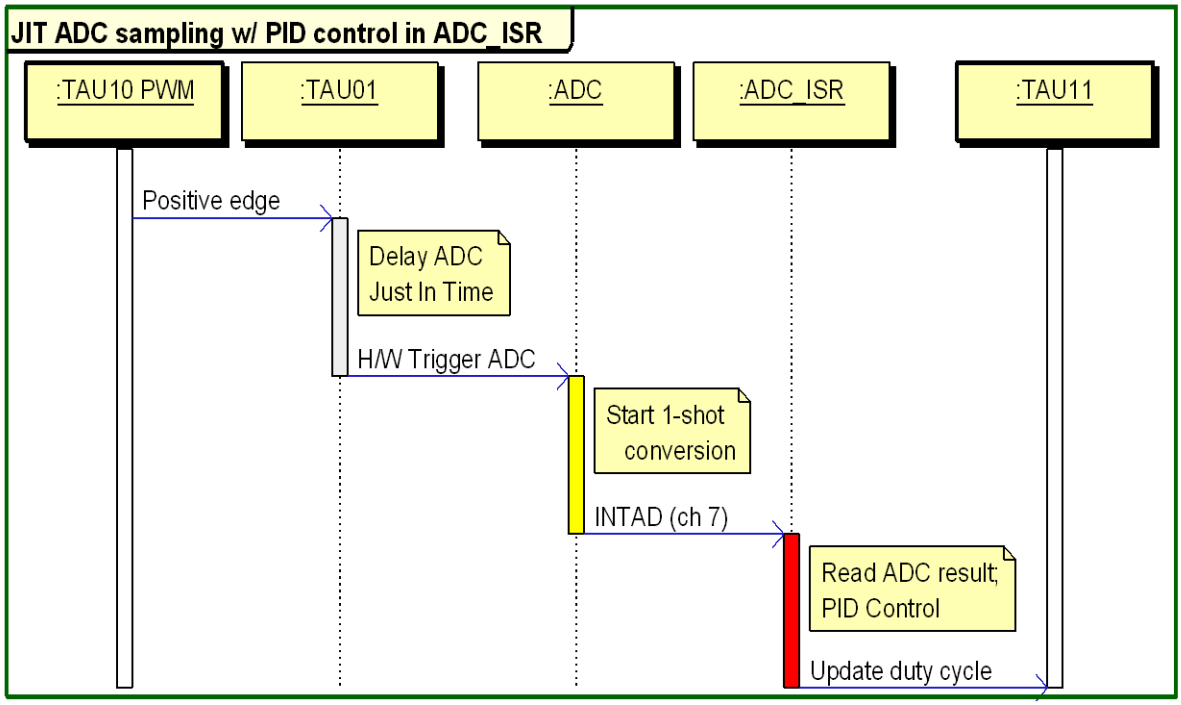


Figure 4.11: Sequence diagram of buck controller shows interactions between hardware (timers TAU10, TAU01, TAU11, and ADC) and software (ADC ISR).

Figure 4.11 shows the sequence of events which occur for the buck converter every $1/f_c$ = 20 µs. Hardware peripherals are leveraged so that the only software processing needed is implemented in the ADC interrupt service routine.

Timer TAU10 marks the start of the PWM cycle with an interrupt signal which triggers timer TAU01 to delay and then generate an interrupt to trigger a single A/D conversion. ADC sampling is in phase with the PWM switching cycle in order to minimize noise. This delay schedules the ADC conversion to occur as late as possible while ensuring that the duty cycle of the next (rather than a subsequent) PWM cycle will be updated, as discussed previously for **JIT** sampling. When the conversion completes, the ADC generates an interrupt (INTAD) which invokes the execution of the ADC ISR. ISR latency and overhead are reduced through register bank switching. The ISR reads the ADC result, performs the PI control law calculation, and updates the duty cycle in timer TAU11.

Note that the only software which executes in this sequence is the ADC ISR. The other interrupts are used as hardware triggering signals between peripherals. Because of this they incur no processing load, and also are invulnerable to preemption or delays from by other activities in the system.

ii. Controller Code

Here, some of the optimizations are discussed for simple control software for a buck converter with an independent load. This corresponds to equation (3.45), explained in Section 3.8. Control software for multiple voltage domain SMPS based system is given in Appendix A, which use equation (3.1) for the voltage rail powering the MCU itself and are optimized using similar techniques. Following is a snippet of the C code along with its generated assembly, which sits in the ADC ISR and is responsible for sampling the output voltage (sampled result in ADCR) and updating the duty cycle by storing a value in a timer register TDR11.

The assembly is also supplemented with the number of clock cycles it takes to execute each instruction.

```
 scaled error = scaled set-point – scaled Vout
t = set_value - ADCR;
   \   000007 AF....              MOVW      AX, N:set_value    ;; 1 cycle
   \   00000A 261E                 SUBW      AX, S:0xFFF1E      ;; 1 cycle
   \   00000C 16                   MOVW      HL, AX             ;; 1 cycle

//store prev error
ctl_parms.e[1] = ctl_parms.e[0];
   \   00000D AF....              MOVW      AX, N:ctl_parms+6  ;; 1 cycle
   \   000010 BF....              MOVW      N:ctl_parms+8, AX  ;; 1 cycle

// calculate current error and convert to 4_12 FXP format  e = t*2*3.3*4096/(1024*64)
ctl_parms.e[0] = (t>>2) + (t>>3) + (t>>5);
   \   000013 17                 MOVW      AX, HL             ;; 1 cycle
   \   000014 315F                 SARW      AX, 0x5            ;; 1 cycle
   \   000016 14                   MOVW      DE, AX             ;; 1 cycle
   \   000017 17                   MOVW      AX, HL             ;; 1 cycle
   \   000018 313F                 SARW      AX, 0x3            ;; 1 cycle
   \   00001A 12                   MOVW      BC, AX             ;; 1 cycle
   \   00001B 17                   MOVW      AX, HL             ;; 1 cycle
   \   00001C 312F                 SARW      AX, 0x2            ;; 1 cycle
   \   00001E 03                   ADDW      AX, BC             ;; 1 cycle
   \   00001F 05                   ADDW      AX, DE             ;; 1 cycle
   \   000020 BF....              MOVW      N:ctl_parms+6, AX  ;; 1 cycle

// Control law calculation
    // FLPrepresentation
    //  ctl_parms.d[0] =
                    ctl_parms.d[1] + ctl_parms.e[0]*K1_PI + ctl_parms.e[1]*K2_PI;
   // clear MACRL and move old duty cycle to MACRH → duty converted to 16_16 FXP
                                                                    //     16  LSBs  of
duty will be zero, no need to shift by 16 bits
    MACRL = 0;
```

```
\   000023 CBF00000            MOVW      0xFFFF0, #0x0      ;; 1 cycle
 MACRH = ctl_parms.d[0];
\   000027 AF....              MOVW      AX, N:ctl_parms    ;; 1 cycle
\   00002A BEF2                MOVW      0xFFFF2, AX        ;; 1 cycle

 // MAC new error and k1
 __mach(K1_PI_12_4, ctl_parms.e[0]);
\   00002C DB....              MOVW      BC, N:ctl_parms+6  ;; 1 cycle
\   00002F AF....              MOVW      AX, N:K1_PI_12_4   ;; 1 cycle
\   000032 CEFB06              MACH                         ;; 3 cycles

 // MAC old error and k2
 __mach(K2_PI_12_4, ctl_parms.e[1]);
\   000035 DB....              MOVW      BC, N:ctl_parms+8  ;; 1 cycle
\   000038 AF....              MOVW      AX, N:K2_PI_12_4   ;; 1 cycle
\   00003B CEFB06              MACH                         ;; 3 cycles

 // normalize MAC accumulator from 16.16 down to 16.0 for d (integer value for timer)
 ctl_parms.d[0] = MACRH;
\   00003E AEF2                MOVW      AX, 0xFFFF2        ;; 1 cycle

// make sure duty is within range
 if (ctl_parms.d[0] > D_MAX_FXP) {
\   000040 440010              CMPW      AX, #0x1000        ;; 1 cycle
\   000043 BF....              MOVW      N:ctl_parms, AX    ;; 1 cycle
\   000046 DC06                BC        ??r_adc_interrupt_0  ;; 4 cycles
\   000048                   ; ------------------------------------ Block: 42 cycles
\   000048                   ; * Stack frame (at entry) *
\   000048                   ; Param size: 0
\   000048                   ; Auto size: 0
 ctl_parms.d[0] = D_MAX_FXP;
\   000048 30FF0F              MOVW      AX, #0xFFF         ;; 1 cycle
\   00004B BF....              MOVW      N:ctl_parms, AX    ;; 1 cycle
```

```
\   00004E                  ; ------------------------------------ Block: 2 cycles
 } else if (ctl_parms.d[0] < D_MIN_FXP) {
 ctl_parms.d[0] = D_MIN_FXP;
 }

 // load new duty cycle to timer register to be read in next PWM positive edge
 TDR11 = ctl_parms.d[0];//D_local;
\                ??r_adc_interrupt_0:
\   00004E AF....              MOVW      AX, N:ctl_parms    ;; 1 cycle
\   000051 BE72                MOVW      0xFFF72, AX        ;; 1 cycle
```

The execution time of the ISR is evaluated experimentally using GPIO output bits and an oscilloscope with infinite persistence enabled (to show all timing cases encountered). This was verified by manually examining the object code to determine the worst-case execution path, using the cycle-count information generated by the compiler and then confirming their times matched. Each execution of the ISR requires 1.26 µs, or about 40 clock cycles, which can be treated as the Worst Case Execution Time (WCET). We see that with a 50 kHz loop update rate, the SMPS requires about 6.3% of the CPU's time. However, it is not necessary to run the control loop that frequently, given that the bandwidth of the buck converter is 8.57 kHz. A compensator with a lower update rate (e.g. 25 kHz) can be designed using the standard methods

iii. Software Optimizations

Apart from the algorithmic optimizations, such as the JIT sampling, there were several optimizations performed in the software implementation to reduce the overhead due control calculations. There was no loss of precision in this implementation and at the same time, the CPU utilization was kept at minimum. This not only reduces race conditions, but also allows the CPU to run other application tasks without incurring much interference from the SMPS control loop. Few of the optimizations performed are enumerated below:

Fixed Point Implementation

The control law implementation for the SMPS requires floating point calculations. Since RL78 does not have dedicated hardware, it relies on standard floating point libraries for floating point operations. This leads to excessive amount of overhead in code as many instructions are required for floating point calculation on an integer Arithmetic and Logic Units (ALUs). Thus, in order to avoid such overhead, fixed point math operations were performed. This not only allowed retaining most of the precision, but also led to faster code execution. The fixed point format chosen for each variable is enlisted below:

Table 4-4: Fixed Point Representation of Control Parameters – RL78

Variable	Comments	Format	Possible Values	Variable Range
e[0], e[1]	Error values	FXP_4_12	[-6.6 , +6.6]V	~[-7.0-2^{-13} , +7+2^{-13}]
d[0], d[1]	Duty cycle (Timer value)	FXP_16_0	[0 , 640]	[0 , (2^{16}-1)]
K1, K2	Control Parameters (scaled to give timer value)	FXP_12_4	[-2096.125, +2096.125]	[-2096.125, +2096.125]
MACR	K*e	FXP_16_16	[0 , 640*2^{16}]	[-(2^{31}-1) , +(2^{31}-1)]
MACRL	Lower 16 bits of MACR	--	0	[0 , 2^{16}-1]

__mach()

The control calculation also takes advantage of the MAC (multiply and accumulate) unit of RL78 MCU. This allows a 16x16 multiplication/ accumulation with 32 bit result precision to complete in 2 CPU cycles. This was earlier explained in Section 4.1.2.3. The control law requires the following calculations.

$$d[0] = d[1] + \mathrm{K}_1 e[0] + \mathrm{K}_2 e[1]; \quad (4.1)$$

Since the new duty cycle, d[0] can be calculated by a few multiply and accumulation instructions, the RL78 MAC hardware is exploited for the same, as depicted in the code above.

Scaling of K1 and K2

K_1 and K_2 are constants and the calculated duty cycle, d[0] needs to be scaled from [0,1] to the timer value [0, 640], where 640 represents 20µs. This additional scaling back and forth can be avoided if K_1 and K_2 are scaled up by a factor of 640. Thus the control law would produce a value which can be directly rounded to the timer value, thus implementing the duty cycle in hardware.

Scaling of Vref-Vout

The ADC employed produces a 10 bit value representative of the sampled voltage and stores it in 16 bit register (ADCR). The conventional way requires following calculation of error,

>> error = Vref – (ADCR>>6)V_{3v3}*2/1023 ;*

If fixed point is implemented, it will further require scaling of error to the fixed point format. All this can be condensed in the following formula for an error in FXP_4_12 format:

>> t = (Vref_scaled – ADCR>>6);

>> error_FXP_4_12 = (t<<4) + (t<<3) + (t<<1) + (t>>1);

A simpler approach can be employed by having another scaled version of Vref_scaled<<6, which is in same format as ADCR. The error calculation further reduces to

>> t = Vref_scaled' – ADCR;

>> error_FXP_4_12 = (t>>2) + (t>>3) + (t>>5);

4.1.2.5 Target Platform –ARM Cortex M0+

The target platform hardware is a development board (FRDM-Kinetis KL25Z) which is built around an ARM Cortex M0+ family of processors by Freescale semiconductors.

Figure 4.12: Kinetis FRDM-KL25Z (ARM Cortex M0+)

4.1.2.6 Microcontroller – ARM Cortex M0+

The ARM Cortex M0+ MCU family features a 32-bit core with a two-stage pipeline. KL25Z CPU cores run at up to 48 MHz and implement integer math instructions for higher performance. It supports Thumb 2 ISA with single cycle 32x32 integer multiply.

The target system's microcontroller MKL25Z128VLK4 features 16 KB of SRAM and 128 KB of Flash ROM on-chip, and runs at supply voltages of 1.8 to 3.6 V. The power vs frequency curve for the KL25Z MCU operating at 3.0V is plotted in Figure 4.13 using equations (4.3) and (4.4).

$$\mathrm{P_{MCU}} = \mathrm{P_{dynamic}} + P_{static} \tag{4.2}$$

$$\mathrm{P_{dynamic}} = \frac{0.246mW}{MHz}\ ; \qquad P_{static} = 5.5689mW \tag{4.3}$$

$$\mathrm{P_{MCU}} = 0.246 * \mathrm{freq_{MHz}} + 5.5689\ mW \tag{4.4}$$

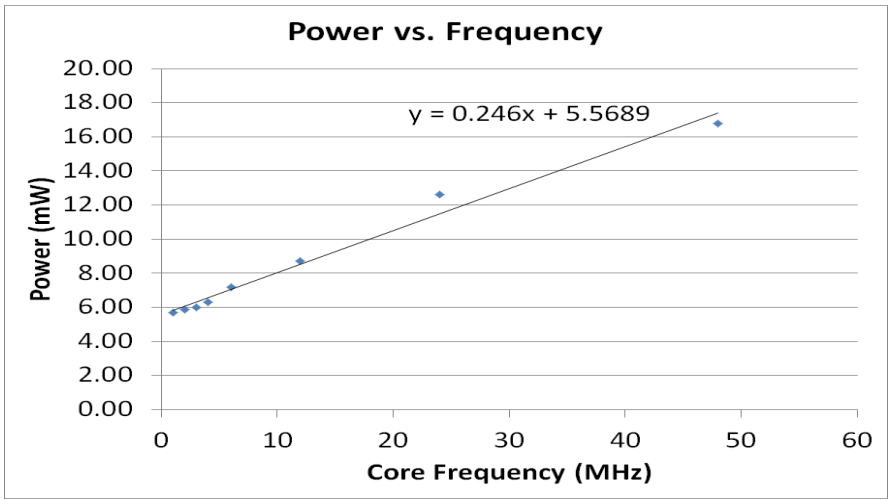


Figure 4.13: Power vs Frequency plot for KL25Z ARM Cortex M0+

There are various power modes for the MCU with appropriate power vs performance tradeoffs. These modes are enlisted below:

- **Run modes – CPU executes instructions**
  - Regular – runs at up to 48 MHz
  - Very Low Power – runs at up to 4 MHz
- **Wait modes (ARM "sleep") – CPU doesn't execute instructions**
  - Select sleep as low-power mode – SCR_SLEEPDEEP = 0
  - Enter by executing WFI instruction (wait for interrupt) or WFE (wait for event)
  - Peripherals operate
  - Exit when enabled interrupt occurs (limited set)
- **Stop modes (ARM "deep sleep") – CPU doesn't execute instructions**
  - Select deep sleep as low-power mode – SCR_SLEEPDEEP = 1
  - Enter by executing WFI instruction (wait for interrupt)
  - Peripherals don't operate
  - Exit when enabled interrupt from AWIC occurs

The current consumptions in these modes are given in Table 4-5.

Table 4-5: Typical currents in different modes of KL25Z MCU

	Current (µA) at V_{DD} = 3V			
Mode	**Normal**	**LL**	**VLL**	**VLP**
Run	5000			250
Wait	3700			135
Stop	345	1.9		4.4
Stop 3			1.4	
Stop 1			0.77	
Stop 0			0.38	

4.1.2.7 Software – ARM Cortex M0+

The SMPS control software implemented on Cortex M0+ works on a similar principle as explained in Section 3.8, but with a different set of optimizations as compared to the RL78/G14 counterpart.

The code was written in C and compiled with Keil MDK 4.0. Code optimization was set to maximum speed, and the source code was optimized with iterative tuning based on the object code.

i. Buck Converter Controller Implementation

Figure 4.14 shows the sequence of events which occur, every switching period, $1/f_c$ = 20 µs, for the control of a single buck converter. Hardware peripherals are leveraged so that the only software processing needed is implemented in the ADC ISR (interrupt service routine). Note that it is not necessary to run the control loop as frequently (f_c), given that the bandwidth of the buck converter is 8.57 kHz. A compensator with a lower update rate (e.g. 25 kHz) can be designed using standard methods.

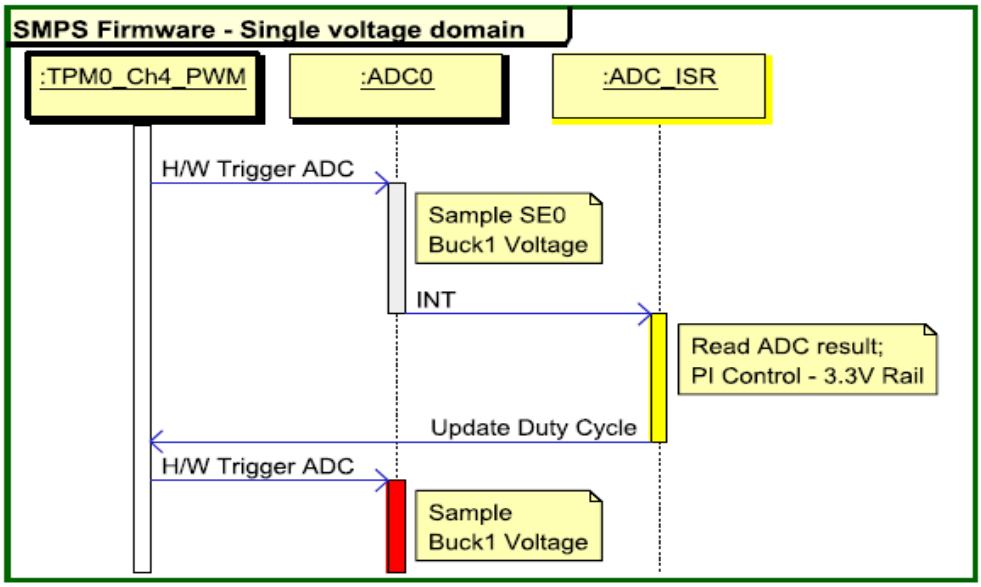


Figure 4.14: Buck control software sequence on FRDM KL25Z (ARM cortex M0+)

The PWM module, TPM0, has various channels (such as Ch 4, 3 & 2) for controlling the buck converters. TPM0 Ch4 generates a hardware trigger to start a single A/D conversion (for channel ADC0_SE0). ADC sampling is in phase with the PWM switching cycle in order to minimize noise. When the conversion completes, the ADC generates an interrupt (INT) which invokes ADC ISR. This ISR performs PI control law calculation, and updates the duty cycle in TPM0 Ch4.

The ADC ISR implements the control law for voltage regulation implemented in 32-bit Fixed Point Math. The voltage error and duty cycle have precisions of 20 integer bits and 12 fraction bits. The constants are maintained at 18 integer and 14 fraction bit precision.

ii. Controller Code

Similar to the RL78/G14 code, following is a snippet of the C code along with its generated assembly, which sits in the ADC ISR and is responsible for sampling the output voltage (sampled result in ADCR) and updating the duty cycle by storing a value in a timer register TDR11.

The object code for disassembly was analyzed by IDA PRO (introduced in Section 3.8.1) for longest execution path, and the instruction set architecture (ISA) manual [37] for ARM Cortex M0+ instruction set was referred for calculation of clock cycles it takes to execute each instruction. The total calculated WCET was confirmed by measuring the execution time on an oscilloscope with infinite precision on by use of a toggling GPIO.

```
int32_t set_value_3_3 = SET_3_3; //  3.3 V in 4_12 format
#define SET_POINT_3_3 3.3
#define SET_3_3 (int32_t)(65535*SET_POINT_3_3/(MCU_VOLT*2))
;;;88                    result= ADC0->R[0];
0000a0  4813            LDR     r0,|L1.240|
0000a2  6900            LDR     r0,[r0,#0x10]
0000a4  4918            LDR     r1,|L1.264|
ctl_parms_3_3.e[1] = ctl_parms_3_3.e[0];
;;;150                   ctl_parms_3_3.e[0] = set_value_3_3 - (int32_t)result;
0000a6  4b1a            LDR     r3,|L1.272|
0000a8  6008            STR     r0,[r1,#0]          ;147  ; result
0000aa  4918            LDR     r1,|L1.268|
;;;153                         ctl_parms_3_3.d[0] += K1_PI_28_4*ctl_parms_3_3.e[0];
0000ac  4c19            LDR     r4,|L1.276|
0000ae  68ca            LDR     r2,[r1,#0xc]        ;147  ; ctl_parms_3_3
0000b0  610a            STR     r2,[r1,#0x10]       ;150  ; ctl_parms_3_3
0000b2  681b            LDR     r3,[r3,#0]          ;150  ; set_value_3_3
0000b4  1a18            SUBS    r0,r3,r0            ;150
0000b6  60c8            STR     r0,[r1,#0xc]  ; ctl_parms_3_3
0000b8  6824            LDR     r4,[r4,#0]  ; K1_PI_28_4
0000ba  680b            LDR     r3,[r1,#0]  ; ctl_parms_3_3
0000bc  4360            MULS    r0,r4,r0
0000be  1818            ADDS    r0,r3,r0
;;;155                         ctl_parms_3_3.d[0] += K2_PI_28_4*ctl_parms_3_3.e[1];
0000c0  4b15            LDR     r3,|L1.280|
0000c2  681b            LDR     r3,[r3,#0]  ; K2_PI_28_4
```

```
0000c4  435a            MULS     r2,r3,r2
0000c6  1880            ADDS     r0,r0,r2
;;;159        if (ctl_parms_3_3.d[0] > D_MAX) {
0000c8  4a14            LDR      r2,|L1.284|
0000ca  6008            STR      r0,[r1,#0]  ; ctl_parms_3_3
0000cc  4290            CMP      r0,r2
0000ce  d902            BLS      |L1.214|
;;;160          ctl_parms_3_3.d[0] = D_MAX;
;;;161        }
;;;162                        else if (ctl_parms_3_3.d[0] < D_MIN) {
;;;163          ctl_parms_3_3.d[0] = D_MIN;
;;;164        }
;;;166                        TPM1->CONTROLS[0].CnV = ctl_parms_3_3.d[0]>>16;
0000d0  4610            MOV      r0,r2
0000d2  600a            STR      r2,[r1,#0]          ;160  ; ctl_parms_3_3
0000d4  e005            B        |L1.226|
            |L1.214|
0000d6  2213            MOVS     r2,#0x13            ;163
0000d8  0492            LSLS     r2,r2,#18           ;163
0000da  4290            CMP      r0,r2              ;163
0000dc  d800            BHI      |L1.224|
0000de  4610            MOV      r0,r2              ;163
            |L1.224|
0000e0  6008            STR      r0,[r1,#0]          ;163  ; ctl_parms_3_3
            |L1.226|
0000e2  490f            LDR      r1,|L1.288|
0000e4  0c00            LSRS     r0,r0,#16
0000e6  6108            STR      r0,[r1,#0x10]
```

Each execution of the ISR requires 2.32µs or 112 MCU cycles at 48 MHz, which can be treated as the Worst Case Execution Time (WCET). We see that with a 50 KHz loop update rate, the SMPS requires about 11.6% of the CPU's time. However, it is not necessary to run

the control loop that frequently, given that the bandwidth of the buck converter is 8.57 kHz. Compensator with a lower update rate (e.g. 25 kHz) can be designed using standard methods.

iii. Software Optimizations

Similar to the RL78/G14 controller code, there were several optimizations performed in the software implementation to reduce the overhead due control calculations on KL25Z. There was no loss of precision in this implementation and at the same time, the CPU utilization was kept at minimum. This not only reduces race conditions, but also allows the CPU to run other application tasks without incurring much interference from the SMPS control loop. Few of the optimizations performed are enumerated below:

Fixed Point Implementation

The control law implementation for the SMPS requires floating point calculations. Cortex M0+ does not have dedicated hardware; it relies on standard floating point libraries for floating point operations. This leads to excessive amount of overhead in code as many instructions are required for floating point calculation on an integer Arithmetic and Logic Units (ALUs). Thus, in order to avoid such overhead, fixed point math operations were performed. This not only allowed retaining most of the precision, but also led to faster code execution. The fixed point format chosen for each variable is enlisted below:

Table 4-6: Fixed Point Representation of Control Parameters - KL25Z

Variable	Comments	Format	Possible Values	Variable Range
e[0], e[1]	Error values	FXP_20_12	[-6.6 , +6.6]V	~[-2^{31}-1 , $+2^{-31}$-1]
d[0], d[1]	Duty cycle (calculated)	FXP_32_0	[0 , 959<<16]	[0 , (2^{32}-1)]
K1, K2	Control Parameters (scaled to give timer value)	FXP_28_4	[-((2^{28}-1)+1/16), +(2^{28}-1)+1/16]	~[-2^{31}-1 , $+2^{-31}$-1]
TPM1-Duty	Duty cycle (Timer value) d[0]>>16	FXP_16_0	[0 , 959]	[0 , +(2^{16}-1)]

32x32 Integer Multiplication (MULS)

The control calculation takes advantage of the 32x32 multiplication instruction of Cortex M0+. This allows a 32x32 multiplication with 32 bit result precision to complete with one hardware instruction. The control law requires the following calculations.

$$d[0] = d[1] + \mathrm{K}_1 e[0] + \mathrm{K}_2 e[1]; \tag{4.5}$$

Since the new duty cycle, d[0] can be calculated by a few multiply and accumulation instruction, the assembly for the same is depicted in the code above. The duty cycle in the hardware is a 16-bit value, whereas d[0] is a 32-bit value. Thus, a single instruction is required to shift out the 16 LSBs.

Scaling of K1 and K2

K_1 and K_2 are constants and the calculated duty cycle, d[0] needs to be scaled from [0,1] to the timer value [0, 959], where 960 represents 20µs. This additional scaling back and forth can be avoided if the float values of K_1 and K_2 are scaled up as presented in the following code:

```
#define SCALE   (float)((4096.0)*2*MCU_VOLT/65535.0)        // inverse of ADC scaling
#define PWM_PERIOD (959)
#define FL_TO_FX_28_4(a) ((FX_28_4) (a * (1<<4)))

int32_t K1_PI_28_4 = FL_TO_FX_28_4(K1_float * PWM_PERIOD * SCALE)
int32_t K2_PI_28_4 = -FL_TO_FX_28_4(K2_float * PWM_PERIOD * SCALE);
```

Thus the control law would produce a value which can be directly rounded to the timer value, thus implementing the duty cycle in hardware. SCALE is used to compensate for the scaling of ADC sampled value, as explained below.

Scaling of Vref OR set_value_3_3

The ADC employed produces a 16 bit value representative of the sampled voltage and stores it in 16 bit register (ADCR). The conventional way requires following calculation of error,

>> error = Vref – (ADCR)V_{3v3}*2/(2^{16}-1) ;*

If fixed point is implemented, it will further require scaling of error to the fixed point format. All this can be condensed in the following formula for an error in FXP_20_12 format. Since *Vref* is a constant, it can be scaled up once, rather than having to scale the calculated error every time. Thus, the dynamic scaling of ADCR is eliminated by using the following scaling of Vref and corresponding scaling of K1 and K2 for duty cycle calculations, which were explained earlier.

```
int32_t set_value_3_3 = SET_3_3;                    //  3.3 V in 4_12 format
#define SET_POINT_3_3 3.3
#define SET_3_3 (int32_t)(65535*SET_POINT_3_3/(MCU_VOLT*2))
```

4.2 Experiments and Analysis – Mr. Buck

In this section, the experiments and analysis are presented which validate the work and contributions discussed in the previous chapter. These experiments are divided into two sections, based on the platform employed and their relative loads. The experimental platform based on the RL78/G14 (RDK) controlled buck converter is presented in this section. The experiments carried out on the RDK are fundamental in nature and reinforce the theoretical analysis.

The experiments carried out on the EPS board on the other hand are conceptually broad in nature and give a high level view of some of the earlier presented concepts. These experiments serve as a proof of concept by showing a multi-domain system being regulated by a SMPS with a central control. The experiments performed using the EPS board are presented in the following section.

4.2.1 Load Characterization

The performance characteristics, controller response and reliability of any power supply are analyzed by loading it with a perfect step load. Since a step change in load current, with a very high slew rate, are challenging in practice, they must be treated as the worst case scenarios. Due to this reason, a constant current load was implemented, switching between two current levels intermittently to mimic a step change in load (loading and unloading conditions).

The principle of operation of the current load is depicted in Figure 4.15

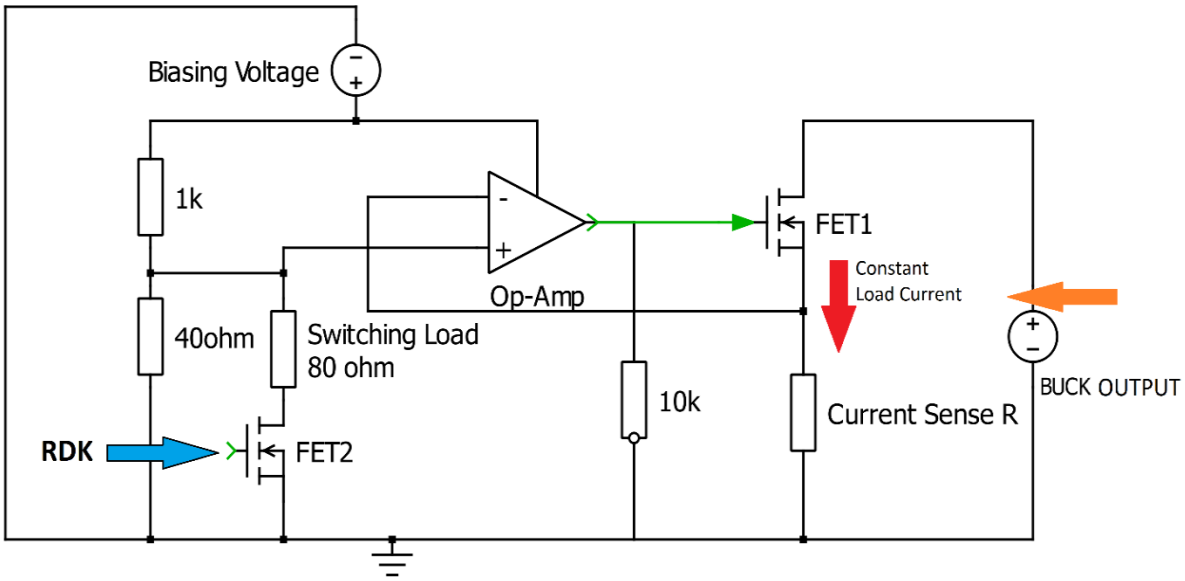


Figure 4.15: Constant Current Load Implementation

4.2.1.1 Constant Current Load Circuit

Here, a simple voltage follower circuit was implemented essentially using an op-amp, and a MOSFET (FET1), operating in the saturation region. The Op-Amp is provided with a negative feedback from the Source of FET1, which is in series with a 1ohm current sense resistor. The drain if the FET1 is fed by the positive terminal of the power supply (Buck

converter in this case). The output of the op-amp provides the biasing to the gate of FET1. The non-inverting input of the Op-Amp is connected to a potential divider circuit which switches between two voltage levels by use of another MOSFET, FET2. This switching effect allows two different gate voltages, V_{GS}, to be produced for MOSFET biasing. Based on the following equation, the drain-to-source current switched between two levels providing a step change in load current to the Buck converter. FET2 is controlled by the RL78/G14 (RDK) and is actually an on board JFET. This JFET is used as a switch using a GPIO from RDK

4.2.2 Response to Transients –Mr. Buck

The response of the experimental platform to transients is examined first. The current load depicted in Figure 4.15 accounts for an additional load of 136 mA by switching it with a separate FET2. The waveforms were captured using a digital oscilloscope and saving the data points for plotting. Typical results using a tantalum output capacitor are shown in Figure 4.16.

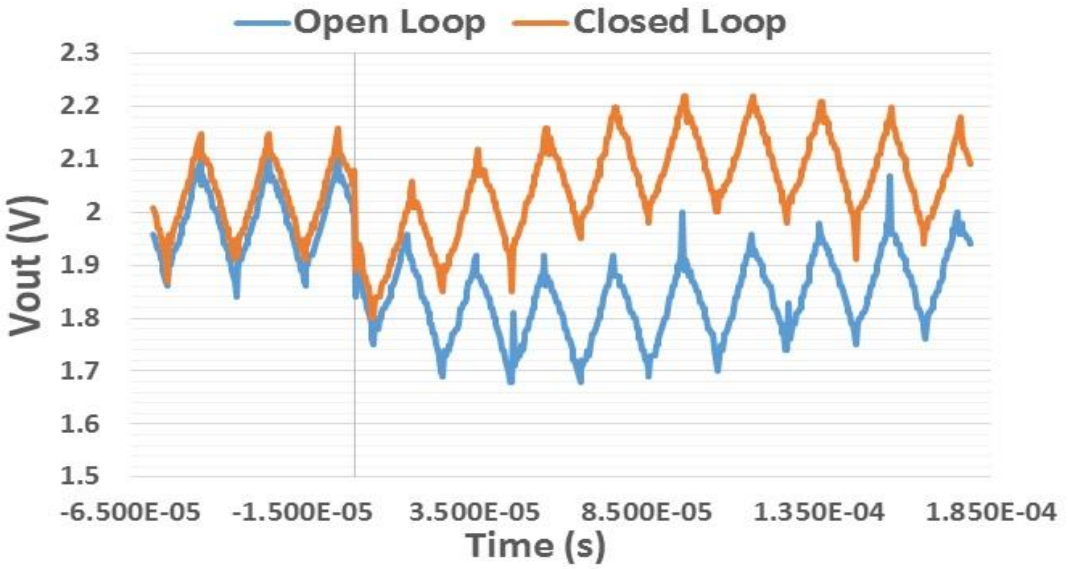


Figure 4.16: Response to 136 mA loading transient with tantalum capacitor.

Table 4-7 shows that the model and the experimentally measured values match closely.

Table 4-7: Experimental Validation of Transient Model

Parameter	Model	Experiment
ΔV_{out_1}	84mV	92mV
$t_{peak,OL}$	Max 74.4 µs	80 µs
$\Delta V_{peak,OL}$	Max 195 mV	184 mV
$t_{peak,CL}$, t_d	6 µs	6µs
$\Delta V_{peak,CL}$	108mV	112mV

We next examine the impact of replacing the tantalum capacitors with X5R ceramic capacitors, with results shown in Figure 4.17. The ripple voltage is reduced dramatically due to the much lower ESR of the capacitor.

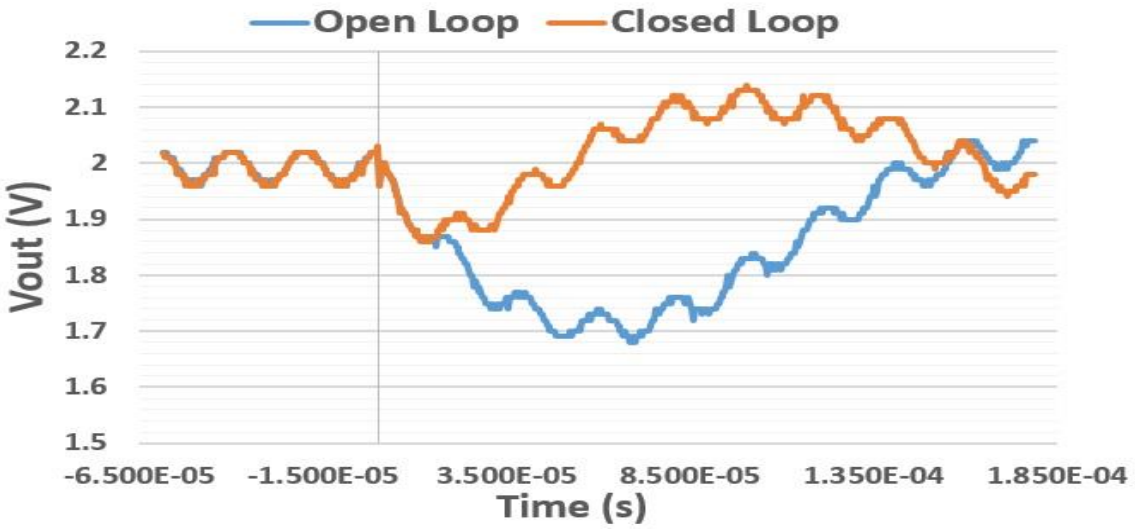


Figure 4.17: Response to 136 mA loading transient with X5R ceramic capacitor (left: open loop, right: closed loop).

The open loop transient responses of the two capacitor technologies were also modelled in a simulation environment, PLECS, which were consistent with the hardware implementations.

It is clear from equations (1.6) and (3.22) that ripple and overall voltage drop are proportional to the ESR, r_C, of the capacitor, which should be very small for a ceramic capacitor. But the voltage drop also depends on the L_{crit}, given in equation (3.19), which decreases significantly, leading to slower response. It can also be noticed in equation (3.4) that r_C is responsible for damping. Lower r_C reduces damping, leading to possible ringing, which is evident from the figure above.

Thus, the tradeoffs of replacing a tantalum capacitor with a ceramic for the same circuit are implementation dependent. A system which requires small ripple but can relax on the response time can employ a ceramic one, whereas, a system with tight response time requirements, but more tolerance for ripple can rely on a tantalum capacitor.

A more comparative depiction of the two capacitor technologies are also provided below. Figure 4.18 and Figure 4.19 depicts the open loop response of the converter during a step loading transient simulated in PLECS and implemented in hardware respectively. Figure 4.20 on the other hand compares the actual closed loop optimal response for the two capacitors as seen on hardware.

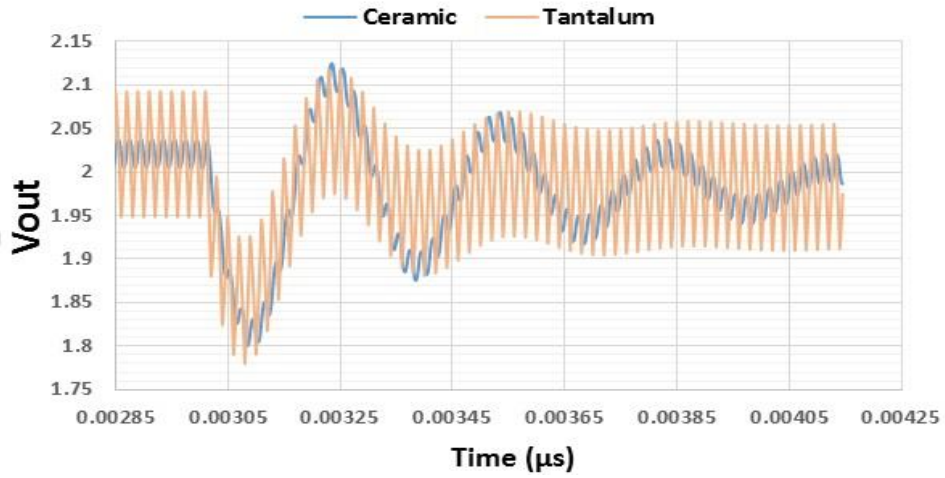


Figure 4.18: Open Loop Step Response in PLECS

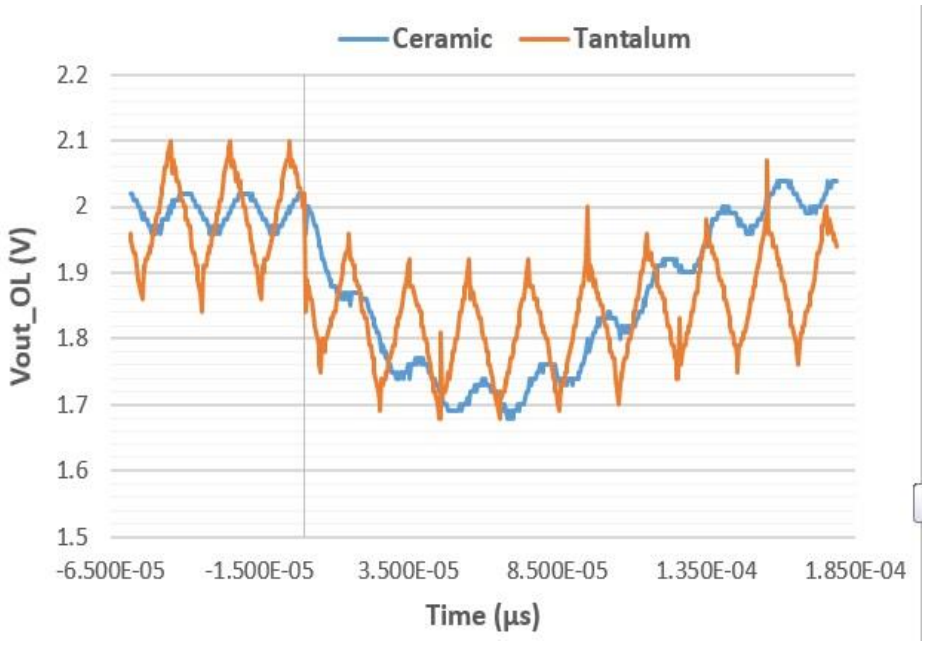


Figure 4.19: Open Loop Step response in Hardware

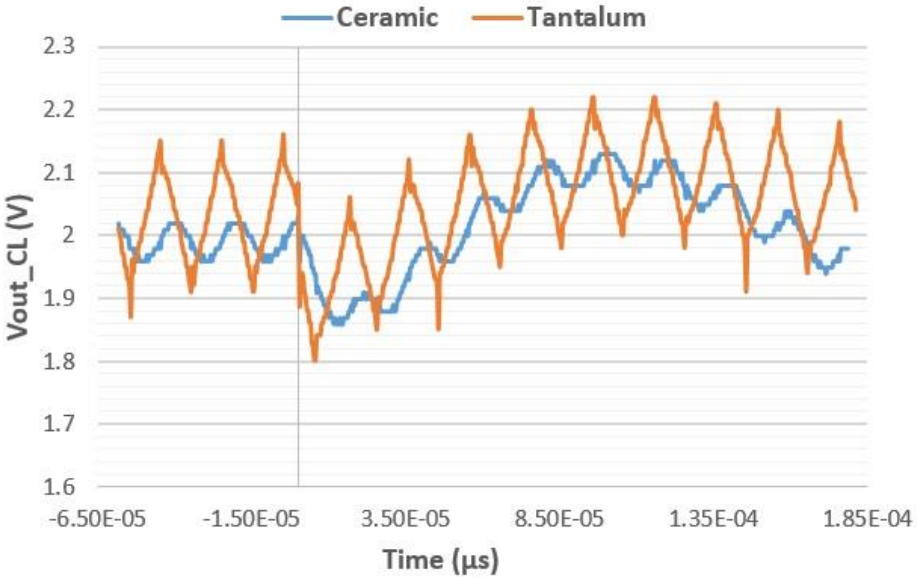


Figure 4.20: Closed Loop Response in Hardware

4.2.3 Impact of Interrupt Lockout and Task Blocking

Next the impact of increasing amounts of interrupt lockout or blocking is examined. Recall that the control loop is implemented in software. As long as that control loop code is not able to execute, the converter will operate in open-loop mode rather than closed loop.

We incur blocking by disabling interrupts briefly when the load is switched on. The duration of the interrupt lockout time is increased gradually to show its impact on the minimum output voltage drop (change in ΔV_{peak}). Here the control loop period and switching period are both 20 µs. We also compare the effect of having the control loop triggered by PWM positive edge (normal sample) with that of JIT sample. JIT sample allows the transient detection window to be expanded and thus significantly decreasing the response time.

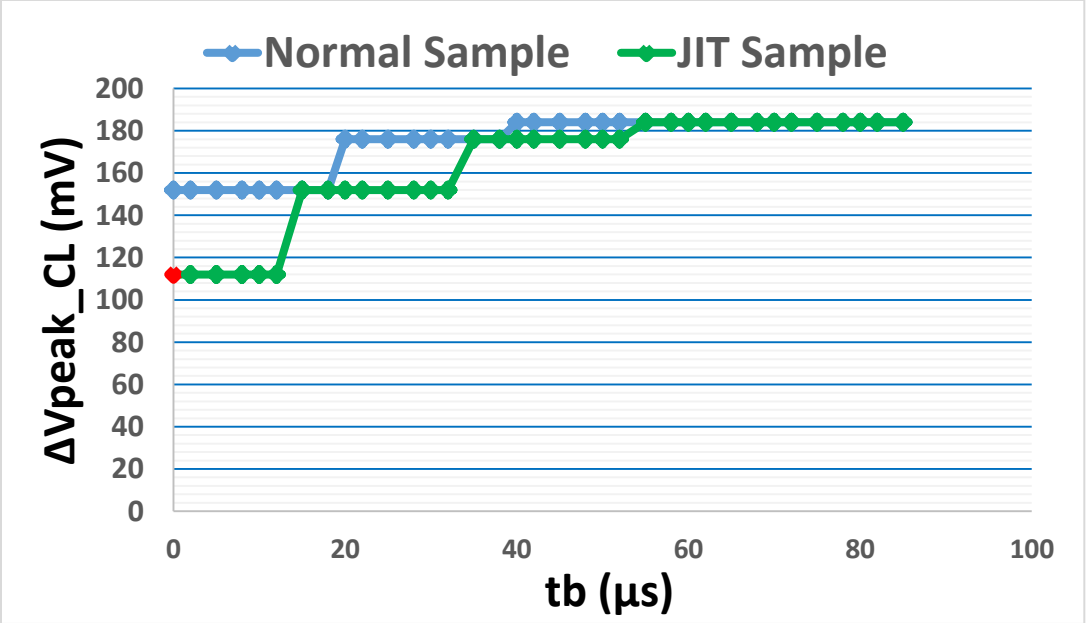


Figure 4.21: Rising blocking time t_b after current transient of 136 mA increases peak voltage transient ΔV_{peak} until reaching open-loop peak.

Examining Figure 4.21, for the normal sampling case (blue), we see that the peak voltage drop with no blocking time is 152 mV. As the blocking time increases from zero, the 20 µs sampling period of the control loop quantizes the peak voltage drop, leading to step increases. Eventually ΔV_{peak} stabilizes at the level of $\Delta V_{peak,OL}$ (184mV). At this point, control loop execution has been delayed so much that the response is that of the open-loop circuit. The response of the converter with normal sampling, w.r.t. the blocking time is also highlighted in figures below.

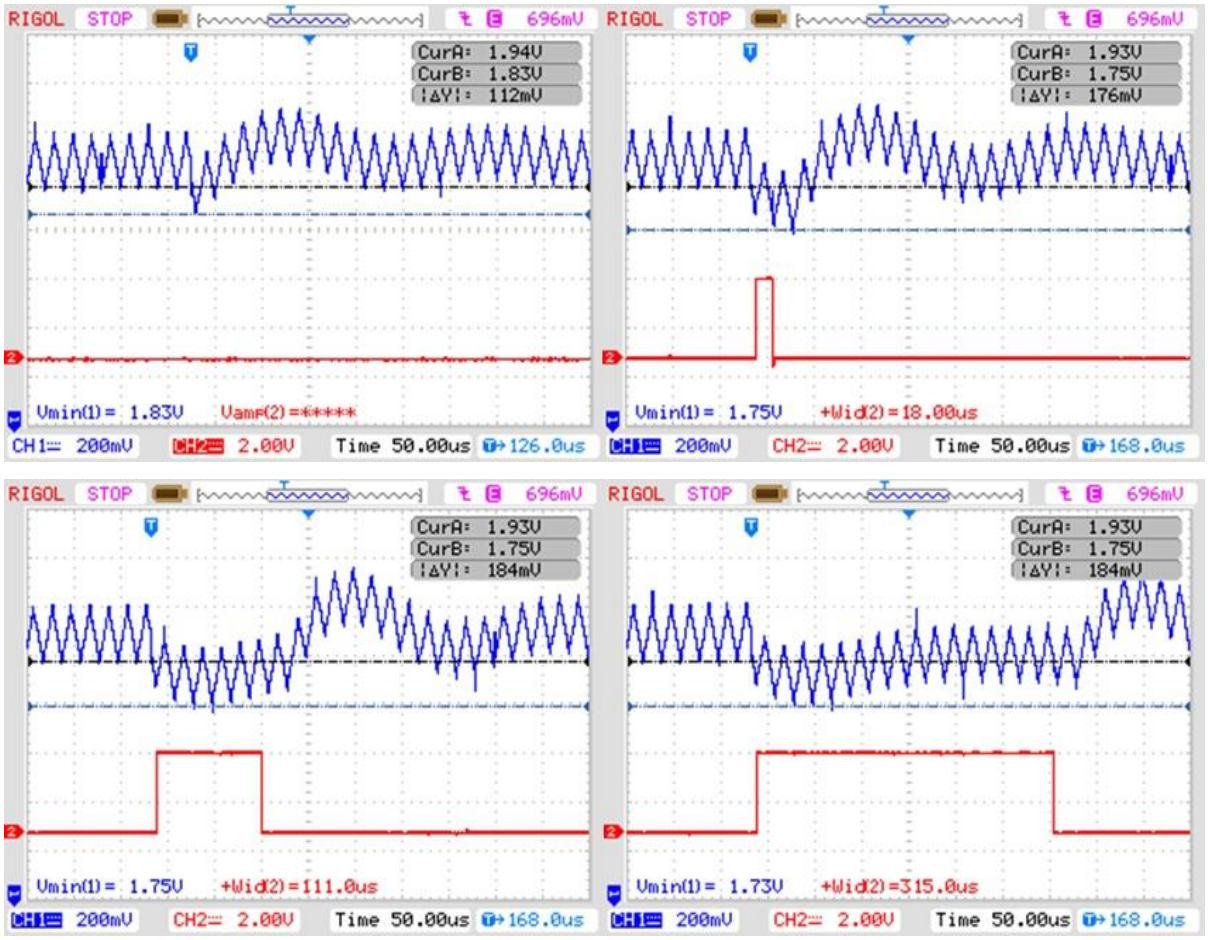


Figure 4.22: Response of Buck with normal sampling with varying blocking times

On the other hand, JIT allows more room for blocking. The delayed sampling (JIT), detects and updates the duty cycle just before the next positive edge of PWM. This allows more time for a transient to occur and still be corrected before next update. This effect is noticed as the plot in Figure 4.21 shifts to the right by the JIT delay time.

Hence, for the normal sampling case, the control loop processing has a hard deadline of $T_{peak,OL}$=80 µs which is then sampled every 20 µs, leading to a deadline of 40 µs. This is because, additional 2 cycles (40 µs) are required for detection and recovery before voltage drops to an open loop condition. In case of JIT, the deadline is 54 µs, due to the delay of 14 µs ($20 - t_{sample} - t_{control}$) introduced in sampling, where the sampling and update occurs in the same cycle. We can define a utility function for the control task based on the voltage drop $\Delta V_{OL}(t)$.

This can be used to quantify the impact of interrupt lockout and task blocking on the system. In this case, if the total blocking time for the control loop with JIT is 54 µs or greater, then the system operates as open-loop during that time. If the resulting voltage levels are unacceptable, then the modifications discussed in Section 3.4.3 above can be followed.

4.2.4 Effects of Control Loop Update rate

As explained earlier in Section 3.4.2.2, slowing down the control loop w.r.t. the switching period, T_{sw} will have an effect on V_{margin} and hence on the allowable scaling of voltage to reduce energy consumption. This section provides a simple experiment conducted by varying the control loop update rate, $1/\boldsymbol{f_c}$ and observe its effect on $\Delta V_{peak.}$ Figure 4.23 depicts the effect of varying the control loop frequency, $\boldsymbol{f_c}$ on ΔV_{peak} for various step load current transients. It is observed that on slowing the control loop frequency, the peak voltage reaches a maximum value, which is same as that of the open loop response. The effect of increasing the control loop frequency beyond the switching frequency, $\boldsymbol{f_{sw}}$ will not have an effect on $\Delta V_{peak,}$ but might make the system unstable as the duty cycle can at most frequently be updated in hardware at every switching period. Even if the control loop executes more often, it will see no effect on output and will try to steer the control to an extreme.

Another point to notice is that, with increasing step load, the values of ΔV_{peak} increases. This is natural as the rate of drop depends on ΔI. The larger the transient, larger will be the voltage deviation.

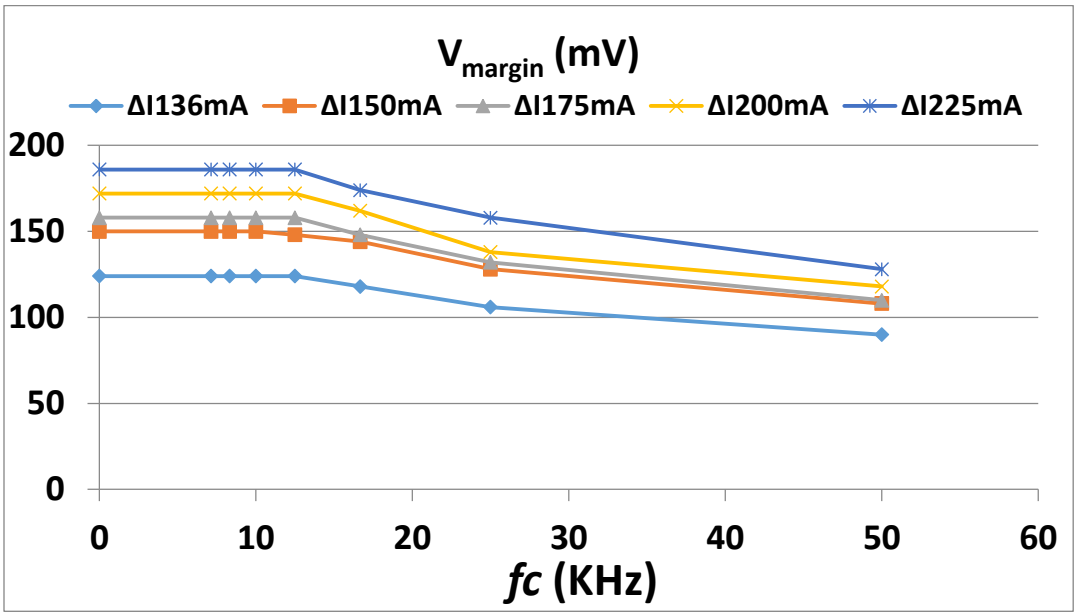


Figure 4.23: Effect of varying Control Loop frequency on ΔV_{peak} for various step load conditions

4.2.5 Changing SetPoint

We evaluated the time needed for the buck converter to transition between operating voltages. Raising the 'set-point' from 2.0 V to 3.3 V took about 1 ms, while lowering it back took 1.3 ms. This is depicted in Figure 4.24 and Figure 4.25.

We can reduce this time if needed by decreasing the output capacitance, but we will pay a price in increased output ripple and tighter response time requirements for the controller to handle transients.

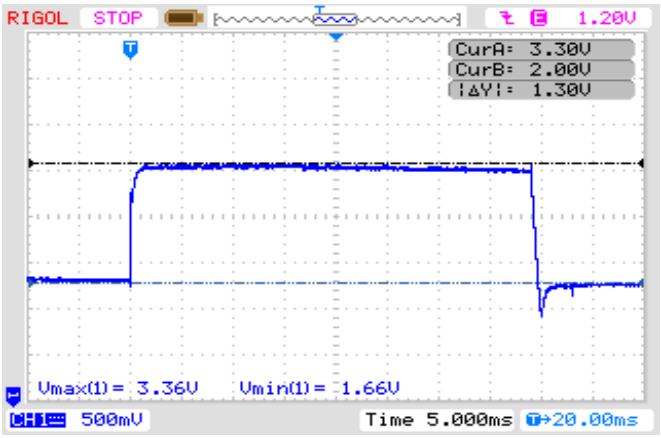


Figure 4.24: Output voltage transient due to change in set-points

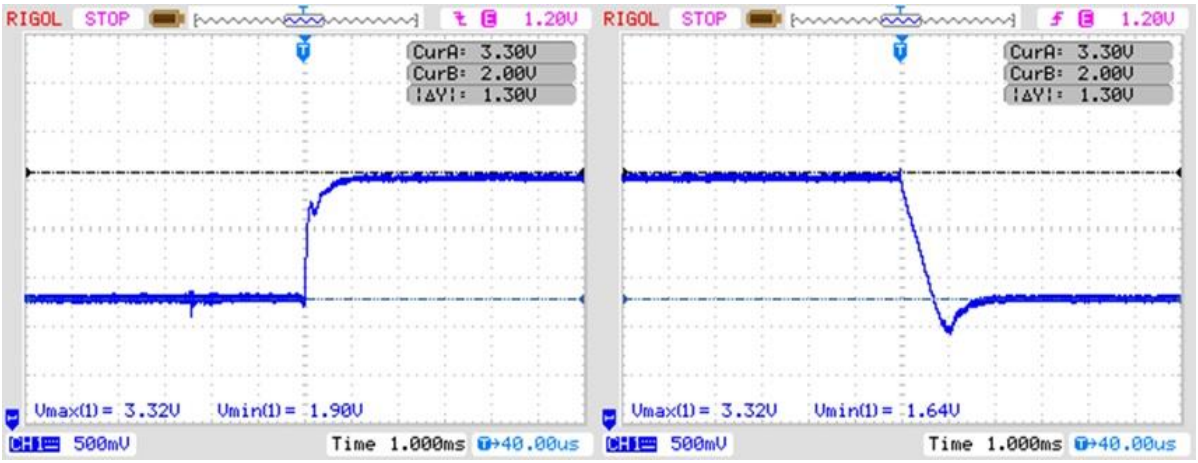


Figure 4.25: Transient response on increasing set-point (Left); Transient response due to decreasing set-point (Right)

4.2.6 Mr. Buck Efficiency

As discussed earlier in Section 3.7, the efficiency of a buck converter various with the magnitude of load current. Depending on load conditions, as highlighted in Figure 3.14, the efficiency is punctured by the dominating circuit parasitics, such as ESRs of passive components and loss elements of passive components (switching losses etc.) [7] [32]. In

order to improve the efficiency at a targeted load condition, two methods are proposed. One targets the light load conditions by implementing control in bursts of time, thus reducing static and switching losses. The second targets the high load losses, which are increased due to high value of currents in the circuit. The losses in high load conditions are reduced by use of synchronous switching, rather than a diode, as in the case of non-synchronous buck [34].

4.2.6.1 Burst mode control

This section presents the improvement in the efficiency of a non-synchronous buck converter under light load conditions. As you can see in Figure 4.26, the efficiency of the buck converter can be improved up to 18.4%. This is down to 1.6mA of load current. At higher loads, the efficiency tends to converge with the inherent efficiency of the converter.

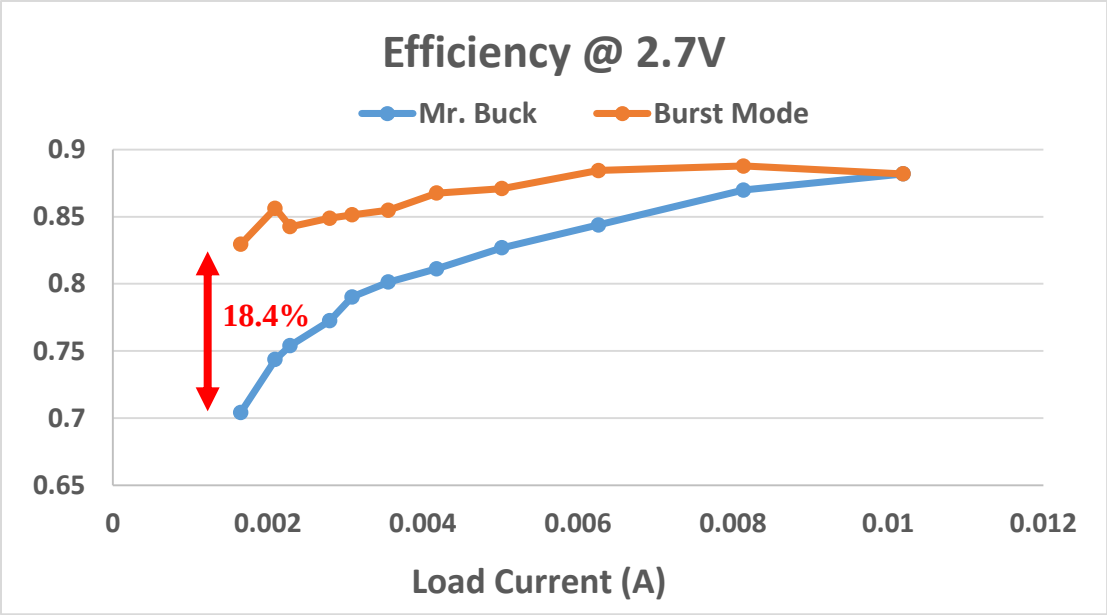


Figure 4.26: Efficiency improvement due to Burst Mode control

The implementation of the burst mode and the limit on allowable voltage deviation is explained earlier in Section 3.7.1.

Figure 4.27 depicts the efficiency of the non-synchronous buck converter in burst mode, at various operating voltages for the entire range of load conditions. It is observed that the overall efficiency is less variable, and higher at low and heavy load conditions, as compared to the conventional control.

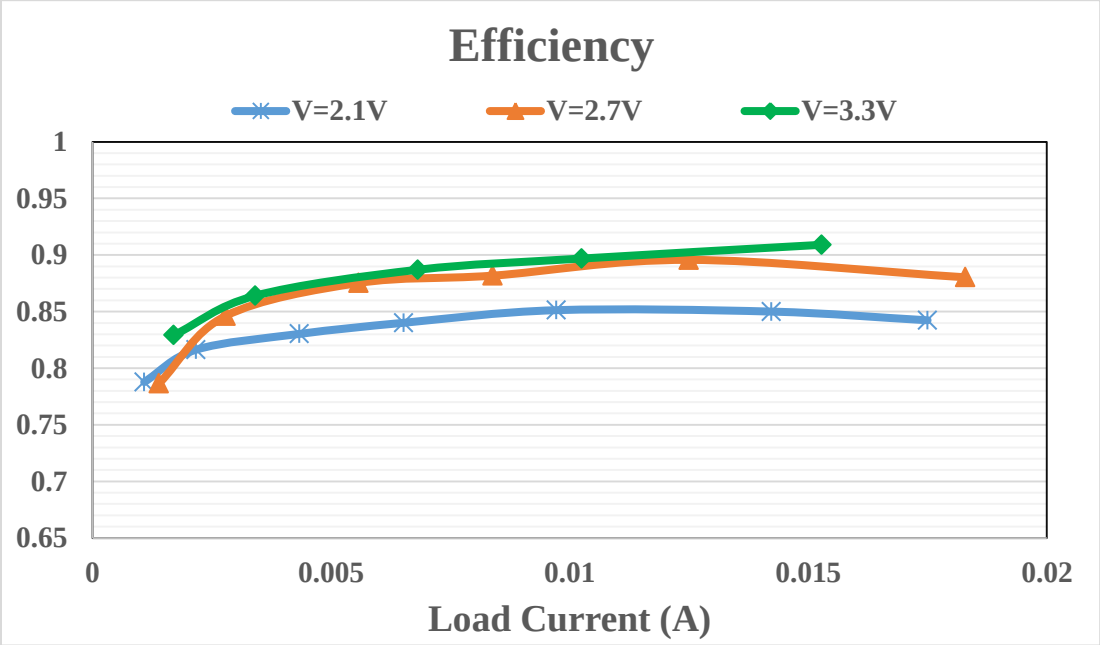


Figure 4.27: Efficiency due to burst mode at various operating voltages

4.2.6.2 Synchronous Buck design

Section 3.7.2 discusses the use of synchronous switching in a buck converter to improve the efficiency in heavy load conditions. [5]The following graph highlights the efficiency of synchronous implementation of the buck converter by replacing the diode with a MOSFET. This requires complementary switching of the two transistors. It can be observed that in high load conditions, the synchronous buck is more efficient. The disadvantages of using the

[5] The efficiency measurements on synchronous buck were made by Zhi Qu, an MS student working under the guidance of Dr. Alex Dean in the ECE department at North Carolina State University

synchronous buck are: 1) the increased complexity in generating complementary PWM signals for the two MOSFETs, 2) increased cost of MOSFET over a diode, and 3) circuit complexity in ensuring that both transistors are never turned on at the same time.

For our system and analysis, we will stick with only non-synchronous buck converters to maintain consistency with load variations.

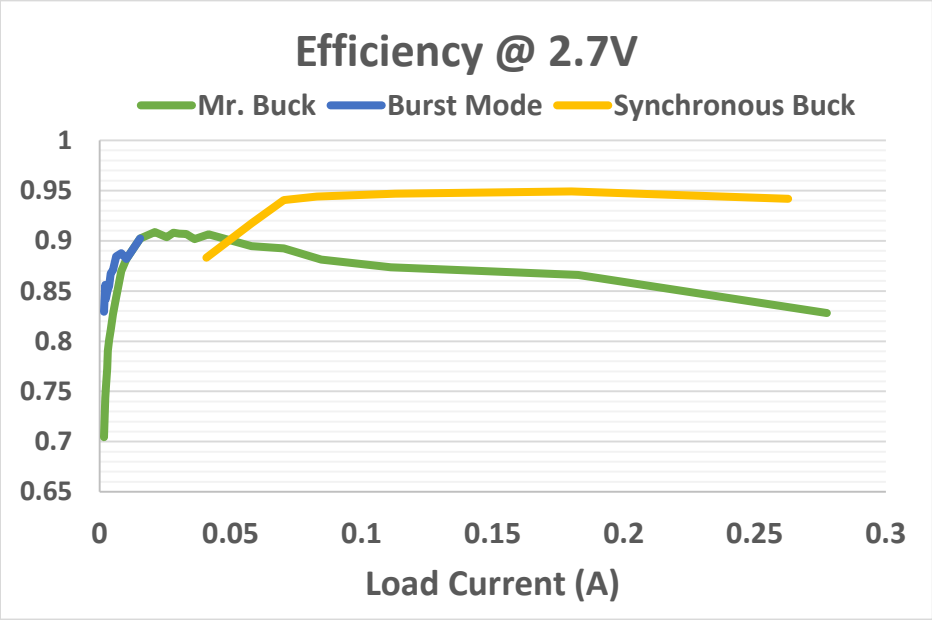


Figure 4.28: Efficiency comparison of various SMPS design modes

4.2.7 Schedulability

This section analyzes the advantages and drawbacks of employing two known scheduling techniques for SMPS control. This analysis is performed based on the schedulability analysis provided in Section 3.4.1. Two preemptive schedulers, one based on dynamic priority (earliest deadline first, EDF) and the other on fixed priority (rate monotonic, RM) are discussed briefly. RM and EDF are priority-assignment approaches, and can be applied to

either preemptive or non-preemptive scheduling. In this case we are just examining the preemptive scheduling versions since they give much better performance.

4.2.7.1 Earliest Deadline First

With EDF, the tasks with earliest deadlines are assigned the highest priorities and can interrupt tasks lower priorities (deadlines later in time). The schedulability calculation for EDF implementation is fairly simple and requires information of the overall utilization of the task set, given in equation (3.25). If the total utilization is less than 1.0 (100%), then the task set is considered to be schedulable. Note that the overhead due to the scheduler, context switches and non-periodic events is ignored for now. Since the technique relies on dynamic priority, it is difficult to determine the exact number of context switches in the program execution.

Considering SMPS control on the RDK, the utilization due to the control task with $f_c=f_{sw}$, is calculated to be 1.26/20 = 6.3%. The utilization of the control task for the KL25Z is 2.32/20 = 11.6%. These are very small amounts. On incorporating embedded application software, the overall utilization is bound to increase. If the control loop frequency is lowered, the utilization will decrease, as given by equation (3.26), and its corresponding hardware response, as given in Section 4.2.4.

4.2.7.2 Rate Monotonic Scheduler

RM scheduling provides a fixed priority assignment. The priorities of the tasks remain the same throughout the execution of the program, unless explicitly altered. The condition for schedulability under RM scheduling, is a little different, as explained in Section 3.4.1.3.

In order to meet the schedulability criteria, without having to determine WCRT, the utilization must be kept below U_{max}. U_{max} starts at 100% for a system of one task, and then approaches 69.3% as the number of tasks increases. As mentioned earlier, the utilization can be lowered by slowing down the control loop update rate, if the tasks in the embedded application can't be altered. Again, this will have an adverse effect on the transient response. If U lies between U_{max} and 1.0, the WCRT analysis is required. The task priority of the

control loop is made to be highest. Thus, the schedulability would depend on the WCRT of all tasks in the task set and is given by equation (3.27).

4.3 Experiments and Analysis – EPS

In this section, some high level implementations on the EPS board are presented to validate the proof of concept. The load characterization is primarily done around a servo motor, which has a smaller load current slew rate. The CubeSat system has a wide spectrum of peripherals and power requirements. In this section, a multi-domain voltage regulation is performed with loads requiring as low as 3.3V (Bluetooth module) to 20 Volts of a resistive load drawing close to 1A of current. Initial experiments involve studying effects of voltage scaling, control loop update rate and MCU utilization, with a servo motor operating at 5V.

4.3.1 Load Characterization

The Servomotor was powered through a power stage of the EPS. The speed and direction were, however, controlled using the dedicated PWM signals from a test-bed controller. The relevant data and observations are given below.

It is clear from Figure 4.29 (bottom left) that there is a dip in voltage by 500mV. This is due to the extra current drawn by the servomotor at the very start of rotation. This drop is restored by the PID control within 1ms (bottom right), without any overshoot and almost no steady state error. As evident, the system becomes critically damped under load conditions, as opposed to under-damped with no load. This is due to the dampening effect of the load and characteristic impedance of the servomotor.

Bottom two figures in Figure 4.29 depict the exact instance of start of rotation of the servomotor, whereas, the upper two highlight the interval from the point of start of rotation, till it comes to a halt. The waveform in blue is the voltage across the motor and the one in red is the current drawn by the motor. The initial drop in voltage is due to increase in current and succeeding peaks in upper figures are due to PWM interference. The interference can be reduced, on increasing the resolution PWM counter bits in the test-bed controller. As the motor stops to rotate, the voltage shoots up and fluctuates due to sudden halt, leading to

decrease in current. Although, the PID controller is quite stable and doesn't cause any fluctuations in the output voltage, the small fluctuations observed are due to the low resolution PWM signals. This can be observed by comparing the two waveforms, upper two, in Figure 4.29. The PWM cause the current to fluctuate and hence the voltage follows similar behavior.

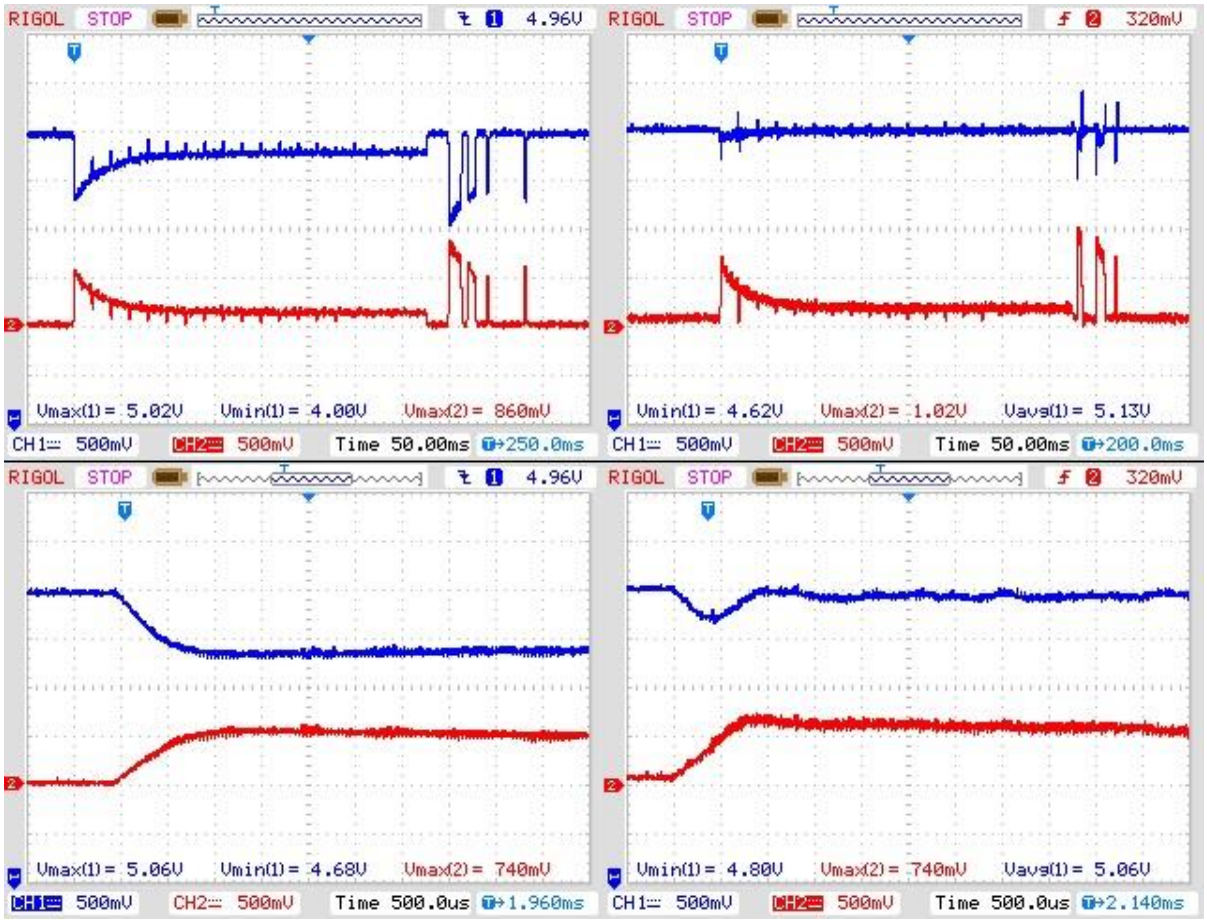


Figure 4.29: Open and Closed Loop response of the Buck to a load of servo motor.
Red: Current Drawn by servo; Blue: Voltage response to load change.
Top: Open (Left) and Closed (Right) Loop response over entire rotation span of servo.
Bottom: Open (Left) and Closed (Right) Loop response at the start of rotation

4.3.2 Effects of Control Loop Update rate

In order to demonstrate the effects of scheduling parameters, a servo motor operating at 5V was considered. The buck converter with, V_{in} = 10V and V_{ref} = 5V, results in an output voltage, V_O = 5V, as a source to the load. The DSP ran a single control task executing the control difference equation given by equation (3.10). The control task frequency, f_{task}, was varied and its effect on the voltage margin, V_{margin} at V_O, is plotted in Figure 4.30. The corresponding data along with the CPU Utilization is summarized in Table 4-8. The relation between U for a single running task on the CPU and the controlling task frequency (control loop update frequency, $\boldsymbol{f_c}$) is given by the equation (4.6).

$$U = \frac{T_{exec}}{T_{ctl}} = T_{exec} \times f_C = 2.5 \times 10^{-6} \times f_C \qquad (4.6)$$

For easier understanding of the concept, V_{min} is assumed to be 5-ΔV_{max}, where ΔV_{max} is the maximum voltage drop under open loop condition, when the load is introduced. ΔV_{max} is obtained to be 540mV.

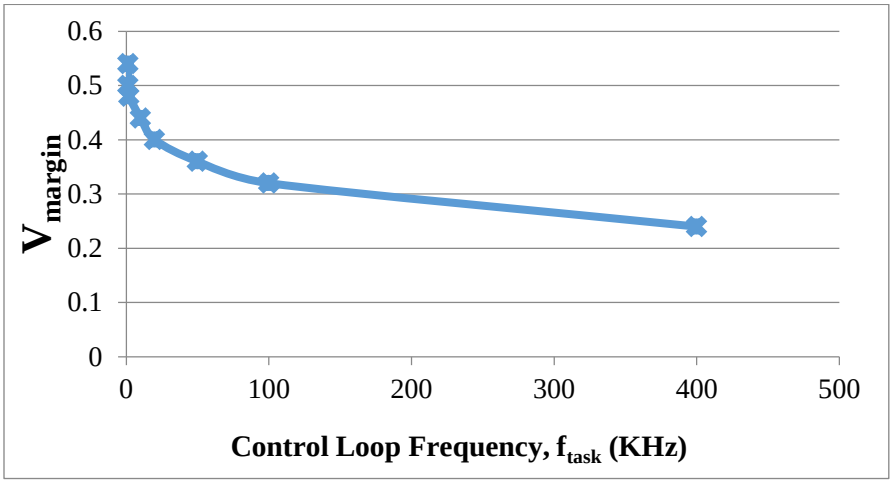


Figure 4.30: Voltage Margin Vs Control Loop Update Frequency

It should also be noted that this plot is quite similar to Figure 4.23. The difference lies in range of values of the control loop frequency, $\boldsymbol{f_c}$ or $\boldsymbol{f_{task}}$, for which the system can recover. In case of servo motor, due to a lower value of the slew rate, the control loop has more flexibility in terms of recovering the voltage at various points. In the case of step load, the T_{peak} is quite small and hence $\boldsymbol{f_c}$ can't be decreased to the same extent as in the case of lower load current slew rate. The execution time for the control task, T_{exec} is 2.5µs without any software optimizations. The effect of f_{task} on V_{margin} and the MCU utilization is shown in Table 4-8.

Table 4-8: Effects of Control Loop Update Rate on V_{margin} and CPU Utilization

Control Loop update rate (µs)	f_{task} *(KHz)*	*U (%)*	V_{margin} *(V)*
2.5	**400**	**100**	**0.24**
10	**100**	**25**	**0.32**
20	**50**	**12.5**	**0.36**
50	**20**	**5**	**0.40**
100	**10**	**2.5**	**0.44**
500	**2**	**0.5**	**0.48**
750	**1.33**	**0.3**	**0.50**
800	**1.25**	**0.3**	**0.54**

4.3.3 Multiple Voltage Domains

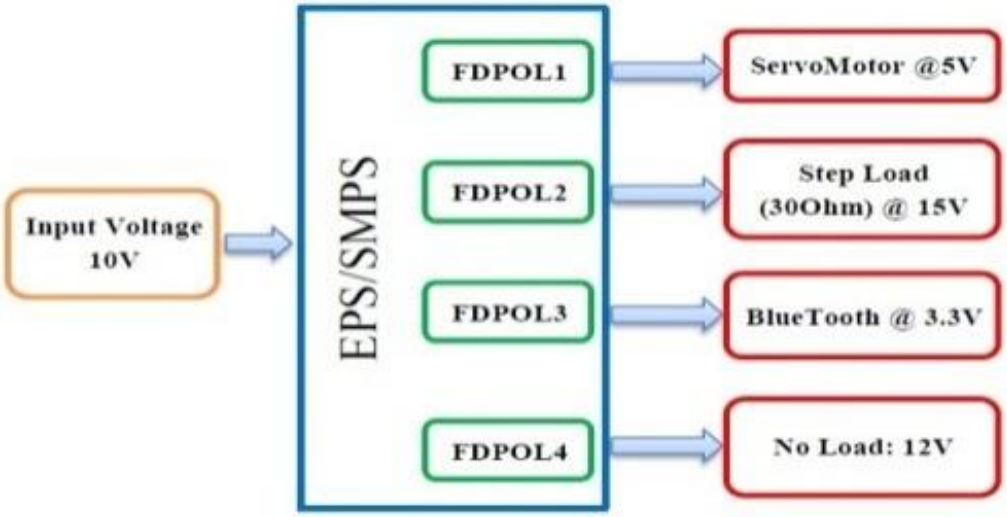


Figure 4.31: Multichannel Load setup for the SMPS

In this section, we present a system which is capable of running four voltage domains using a single controller. The four domains cover a range of operating voltages and current requirements. The unregulated and the regulated system's voltage responses are depicted Figure 4.32 and Figure 4.33, respectively. The test bed used in this system consists of a Servomotor running at 5V, drawing up to 370mA (purple), a Bluetooth module operating at 3.3V sending and receiving data at irregular intervals (orange), a step load of 30Ω being switched on and off using a MOSFET, which draws 1A at 20V (Blue) and an open circuit voltage of 12V (green). The voltage domains working under 10V are controlled using Buck converters, whereas, the ones operating above 10V require Boost Converters, as the input voltage is restricted to 10V. The compensator design for the Boost converters were carried out using the methodologies discussed in the previous sections. The overall test setup is illustrated in Figure 4.31.

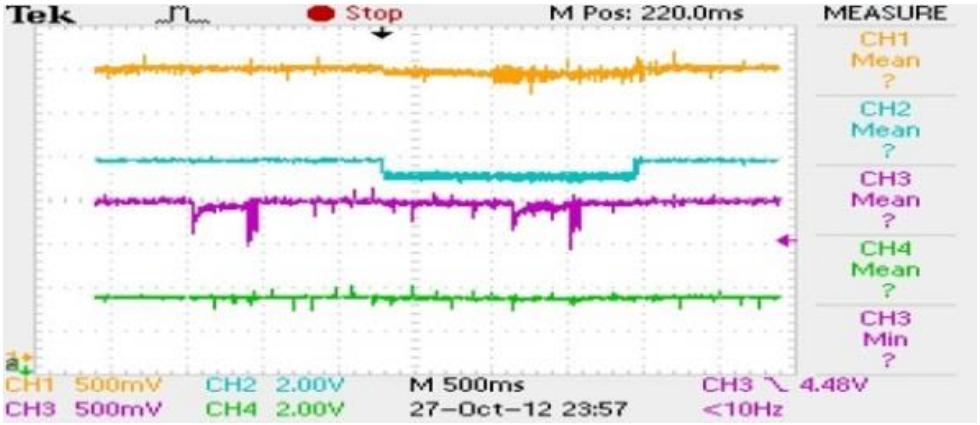


Figure 4.32: Open loop response of system.

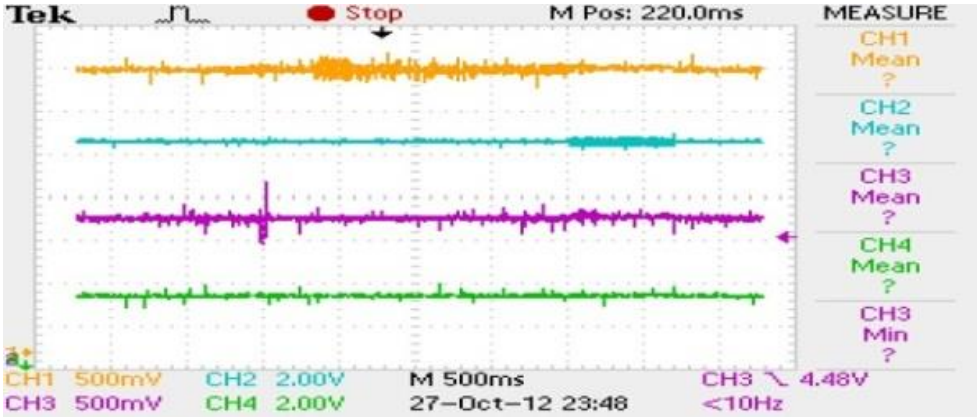


Figure 4.33: Closed-loop response of system.

Chapter 5

System Integration and Results

A number of open ends must be met in order to present a complete system with an embedded SMPS control in an application MCU. This chapter discusses the next steps in this study which aimed at constructing a standalone embedded application, replacing the existing on-board linear regulator with multiple SMPS devices powering segregated peripherals based on their optimal operating voltage ratings. The results confirm the increased efficiency and lower average power consumption by the proposed multi-domain system as compared to the conventional single domain and linear regulator based systems.

In order to present a holistic system, an embedded application consisting of a wide range of peripherals was selected. These peripherals have a different operating voltages and currents. The firmware for the embedded application is implemented on an RTOS resulting in a predictable hardware and firmware behavior as with any *real-time* application. This results in firm deadlines and periodicity in task occurrences. The durability of this standalone embedded system powered off an ultra-cap determines the average power consumed, representing the battery life of this application.

Our target application is a tracking device which uses its GPS receiver to determine location and velocity and logs it to a micro SD card once per second. A graphical monochrome LCD with white LED backlight is the main display. The system also includes three LEDs for indicating additional status. There are various modes of operation of this device. The application is implemented on two separate platforms (RDK and KL25Z) running two different RTOS (Run to completion (RTC), and RTX, respectively), validating a wide range applicability of the proposed power conversion technique.

5.1 Application Design

The tracking device employed as a sample embedded real-time application is, in simple terms, a GPS data logger featuring a GPS receiver, Wi-Fi interface, LCD and a micro SD card. This system has three voltage domains (3.3 V, 2.7 V, 1.7/1.8 V), each powered by a software-controlled buck converter.

Following steps describe the high–level functionality of the device.

1. The GPS calculates coordinates based on data from various satellites.
2. The NMEA message constructed by the GPS is sent to the MCU through a UART interface.
3. The NMEA message is decoded by a task in software.
4. The decoded information is saved and logged onto the SD card.
5. The LCD is updated with relevant information based on screen selection through the use of user switches.
6. The LEDs indicate system status.
7. The module Wi-Fi is powered through the SMPS controlled by the MCU, but the firmware for the Wi-Fi is implemented on a separate board due to code size limitations.

The hardware and firmware implementation are briefly discussed in the following subsections for each platform.

5.1.1 Hardware

The tracking device embedded application, as mentioned earlier, consists of a variety of peripherals with a wide range of operating voltages and power requirements. These set of peripherals represent commonly used devices in embedded applications. Thus, an efficient mechanism to manage power in these devices demonstrates a wide scope of such power management techniques. The proposed methodology in powering these devices results in an efficient system with extended battery life.

The operating voltage ranges of the peripherals used in the tracking device implemented on the RL78/G14 are depicted in Figure 5.1. In an off the shelf low-cost MCU platforms, only a

couple of voltage domains, usually 3.3V and supply 5.0V are available to power these devices. A linear regulator in such systems is commonly used for converting 5.0V to 3.3V. Although, cost effective, this conversion technique results in great loss of energy and reduced batter life. As can be seen, if these devices are allowed to run at their optimal voltages, the energy/power consumption can be significantly reduced.

Later in this chapter we introduce the use of single as well as multiple switching converters for power conversion, replacing the linear regulator. The result is increased efficiency and reduced average power consumption, without compromising the cost by implementing the SMPS control in software within the application MCU, also running application tasks.

It should be noted that the peripheral modules themselves might not be as efficient, as they are only employed to build a sample system. The main purpose of this work is to distinguish and present an efficient power management technique given an embedded system, and not worry about the power efficiency of the peripherals. These peripherals are externally connected to the development board, MCUs (RL78/G14 and KL25Z). The MCUs on the respective development boards (RDK and FRDM) were isolated from the board's power supply, allowing them to be connected to the external power supply and peripheral modules. All components except the MCU and the LEDs are same for both implementations.

The optimal operating voltages of the peripherals with the KL25Z MCU along with their corresponding average power are tabularized in Table 5-1.

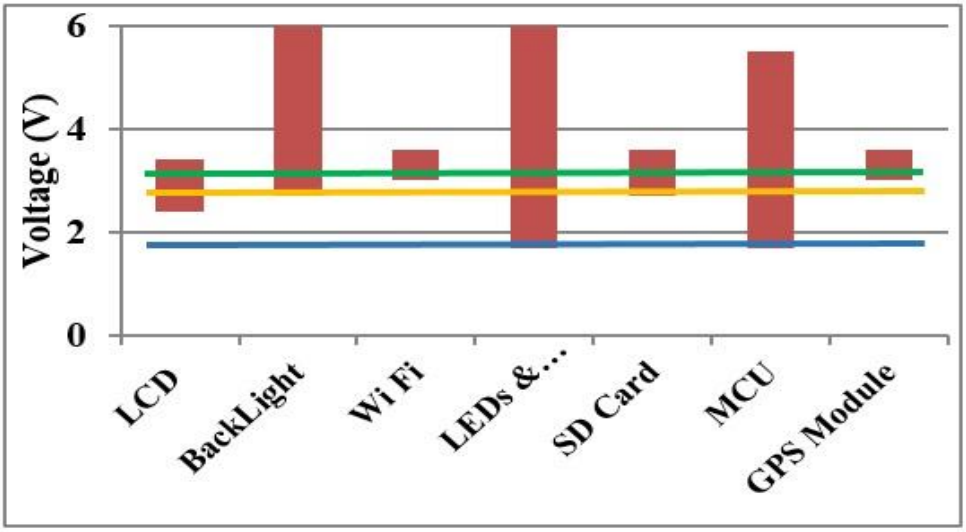


Figure 5.1: Operating voltage ranges of various devices for the RL78/G14 based system

Table 5-1: Current and Power requirements of various peripherals at their optimal operating voltages for KL25Z platform based system

Device	Supply Voltage (V)	Avg. Current (mA)	Max. Current Transient (mA)	Avg. Power (mW)
GPS	3.3 V	64.00	126.00	211.20
Backlight	3.3	11.00	11.00	36.30
Micro SD	2.7	1.00	67.27	2.70
LCD	2.7	1.00	1.00	2.70
LEDs	2.7	21.00	21.00	56.70
MCU	1.8	9.00	3.00	16.20

The two platforms used for an example tracking device, employed the Renesas RL78/G14 MCU and an ARM cortex M0+ based KL25Z MCU by Freescale, which is described in detail in Section 4.1.24.1.2.3. Inherently, the MCUs are powered using respective on-board

linear regulators convertering input 5.0V to 3.3V output. The MCUs are then isolated from the on board 3.3V rails through jumpers and then powered using an SMPS either operating at 3.3V (in a single domain system), or optimally at 1.7/1.8V (in a multi-rail system).

On both MCUs, UART and SPI channels are used to communicate with tracking device modules (GPD, LCD, microSD card). GPIOs are used for powering LEDs and for status monitoring. On board potentiometer and switches are used for configuration purposes.

For RL78/G14, the ADC is capable of performing 10-bit conversion with an option of scan mode to sample multiple channels in sequence. The timer modules are used to generate PWM signals for SMPS control. The Renesas MCU has a wider operating range as depicted in Figure 5.1. The lowest operating voltage required for maximum frequency operation is 2.6V. Thus, based on maximum voltage deviation observed on this rail, MCU requires an operating votlage of around 2.7V.

In case of KL25Z, the ADC performs a 16-bit conversion. Although, this ADC doesn't not have a scan mode, multiple channels and their respective result registers are available to store several conversion simultaniously. The MCU has hardware PWM modules which can be programmed for desired periods and duty cycles. The Cortex M0+ core is a low poer core and can operate at maximum frequency with 1.6V supply. Based on maximum voltage deviations on the supply rail, the optimal operating voltage for the MCU is 1.8V, as depicted in Table 5-1.

The different operating modes for each platform implementation with respective states of the peripherals are tabularized in Table 5-2 and Table 5-3.

Table 5-2: Operating modes of the tracking device on RL78/G14

Mode	**MCU**	**GPS**	**Wi-Fi**	**LCD**	**BL**	**SD**	**LED+Switches**
Operating	ON	ON	ON	ON	ON	ON	ON
Stand-alone	ON	ON	OFF	ON	OFF	ON	ON
Low Power	ON	0.1Duty	OFF	ON	OFF	ON	ON
Sleep	ON	OFF	OFF	OFF	OFF	OFF	ON

Table 5-3: Operating modes of the tracking device on KL25Z

Mode	MCU	GPS	LCD	BL	SD	LED+Switches
Active - Night	ON	ON	ON	ON	ON	ON
Active - Day	ON	ON	ON	OFF	ON	ON
Low Power	ON	0.1Duty	ON	OFF	ON	ON
Configuration	ON	OFF	OFF	OFF	OFF	OFF
Standby	ON	OFF	OFF	OFF	OFF	OFF

5.1.2 Firmware

In common cases, a real-time embedded application exploits an RTOS for firmware implementation, to ensure reliability in terms of functionality and schedulability of tasks. Based on an application, either a pre-emptive or a non-preemptive RTOS can by employed. A priority based RTOS determines the precedence of certain events or task executions over others, as only one task can utilize the MCU.

The RTOS used for the tracking device embedded application consists of ISRs, and tasks (event triggered and periodic). Some of the tasks for the corresponding devices are highlighted in Table 5-4.

Table 5-4: Firmware tasks for the position tracking device

Device	Task/ISR	MCU Peripheral	Trigger
Scheduler	**Tick_timer ISR**	**Timer**	**Periodic**
GPS	**UART_rcv ISR**	**UART**	**Interrupt**
	Decode_Task	**-**	**Event**
SD	**spi1 _irq ISR**	**SPI**	**Interrupt**
	SD_write_task	**-**	**Event**
LCD	**spi1 _irq ISR**	**SPI**	**Interrupt**
	Update LCD Task	**-**	**Periodic**
Backlight	**Sw_ISR**	**GPIO**	**Interrupt**
Switches	**Sw_ISR**	**GPIO**	**Interrupt**

The details of the firmware and the respective tasks/ISRs are tabularized for each RTOS implementation later.

5.1.2.1 RTOS for RL78/G14 - RTC

The firmware on the RL78/G14 is implemented using a non-preemptive, static priority based RTOS, also known as run-to-completion (RTC) scheduler. RTC allows periodic as well as event triggered tasks to be defined. A scheduler table is used to maintain a state-machine for the tasks along with their priorities. A hardware timer generates an interrupt, which serves as a scheduler tick and updates the scheduler table. The scheduler table is constantly read to allow the highest priority tasks to be executed when ready. There is no pre-emption allowed, thus, the currently running task must finish executing before a higher priority task can be allowed to run. In case of periodic tasks, the programmer can define the period of the task, which is stored in the scheduler table. The scheduler uses a software timer per task, which is decremented at every scheduler tick. RTC also supports the placing the MCU into HALT or STOP mode when no task is ready to run. This allows lower power and energy consumption during large idle periods. It should be noted that the *in and out* latencies to enter and exit the HALT/STOP modes must be considered when these modes are to be leveraged.

RTC is a simple scheduler and does not inherently provide synchronization or data sharing mechanism. The footprint of this scheduler is very small. This allows an effective demonstration of SMPS control software to be implemented on modest schedulers.

5.1.2.2 RTOS for KL25Z (ARM Cortex-M0+) – RTX

In order to demonstrate the SMPS control software's portability in a more complex scheduler, real-time executive (RTX) was employed for firmware implementation on the KL25Z MCU. This RTOS allows following properties to be leveraged in firmware development.

1. Inter-process communication and synchronization (safe data sharing)
2. time management
3. I/O abstractions

4. memory management
5. file system
6. GUI
7. networking support

Many scheduling approaches are supported, including purely prioritized, round robin, and non-preemptive

Synchronization primitives such as mutex, semaphores and message queues were used to ensure correctness in data sharing, avoiding race conditions and for signaling between multiple tasks/ISRs.

Similar to RTC, RTX maintains a state machine for tasks and the appropriate task is chosen for execution based on priorities and other rules. It also supports low power states when no task is ready to run. This allows power savings based on the various run/wait/stop modes supported by the MCU hardware (Section 4.1.2.6) and also depending on which peripheral can be power/clock gated. Again, to ensure significant power savings, the latencies involved in these power mode transitions must be taken into account.

5.2 Multiple Voltage Rails

One of the primary objectives is to demonstrate the power reduction and increased efficiency of a system by operating peripherals of the embedded device at their optimal operating voltages. This is established by determining an optimal number of voltage rails in a system and grouping the peripherals to be powered from the appropriate rail. Since a linear regulator is inherently inefficient, SMPS is used for each rail. The scalability comes at a price of increased cost of having multiple rails. In order to target the cost and flexibility of implementation it is suitable to incorporate the control of multiple voltage domains into central control software as a part of the embedded MCU firmware. The motivation behind is to present a low cost system with efficient power conversion and control flexibility. This can be accomplished in the test real-time embedded application developed on the Renesas and the ARM platforms.

Embedded systems often include multiple peripheral devices in addition to the MCU. The operating voltage ranges of the components are depicted in Figure 5.1.with vertical bars**.** One approach is to power all peripherals at 3.3V (green horizontal line in Figure 5.1.). However, we can save significant power by lowering the operating voltages, as $P \alpha V^2$ for resistive loads, and $P \alpha V$ for constant current loads. For example, we can add two lower voltage domains (1.7V (blue), 2.7V (yellow)) to power the peripherals to operate at lower voltages.

Firstly, it requires employing analytical methods to identify the bounds for worst-case loading and unloading transients of all devices on the rail, and then grouping them according to those bounds, their operating voltage ranges, and power limits, as tabularized in Table 5-1. Then the linear regulator earlier used to power the MCU platform will be replaced with independent SMPS devices for those segregated domains. The control of these devices will be central to one software task for maximum control efficiency and low software overhead.

To demonstrate the proposed implementation, we implement a test embedded application (tracking device) on two platforms with different firmware. One is based on the Renesas MCU running a non-preemptive RTC scheduler, and the other is an ARM MCU running a preemptive RTX scheduler. The system architecture for each implementation is described in the following subsections. The overall system performance and energy efficiency will be compared with that of the Linear Regulator, as explained below in Section 5.4.

5.2.1 Renesas RL78/G14

The power architecture for the position tracking and logging device implemented on the RL78/G14 platform is depicted in Figure 5.2.

The system uses an RL78/G14 MCU controlling the SMPS consisting of three buck converters. There are three voltage domains as highlighted in blue (3.3V), red (2.7V) and green (1.7V). Based on the V-f curve of the MCU, minimum of 2.V is required to power the MCU for maximum operating frequency of 32MHz. Thus, the 2.7 V rail also powers the MCU. The firmware implemented on the MCU consists of the GPS application firmware and the SMPS control task within the RTC scheduler.

The three buck converters have a common input (5.0V). In order to demonstrate the average power consumed, the 5.0V is supplied through an ultra-cap, which discharges with time. The time to discharge gives an estimate of the average power consumed, as quantified in equation (3.44).

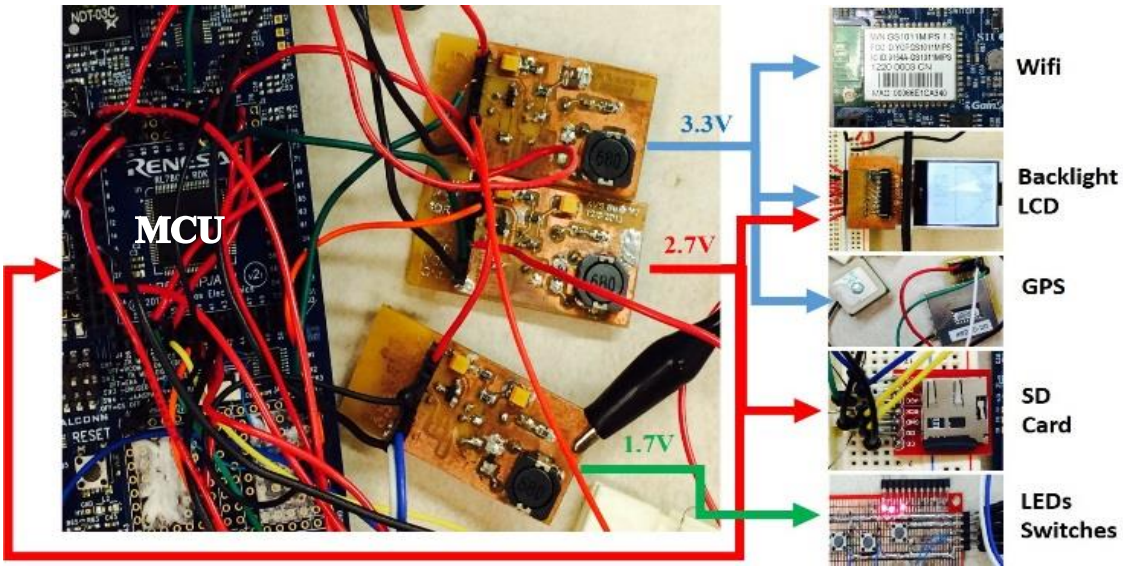


Figure 5.2: RL78/G14 MCU based GPS data logger's power architecture with multiple voltage rails

5.2.2 ARM Cortex M0+ → KL25Z

A similar approach is used in implementing the tracking device on KL25Z platform. The power architecture for the tracking and logging device is outlined in Figure 5.3.

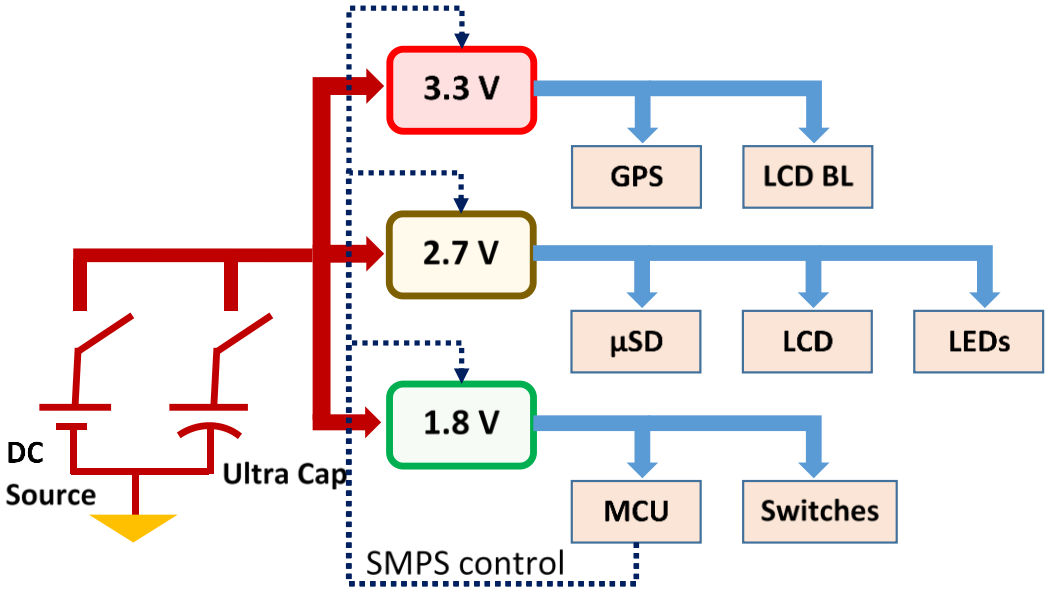


Figure 5.3: KL25Z MCU based GPS data logger's power architecture with multiple voltage rails

The difference lies in the MCU and LEDs voltage rails. As opposed to the Renesas MCU, the ARM MCU only requires only 1.8V for operation, at maximum frequency of 48MHz (Figure 4.13). The onboard LEDs present on the KL25Z FRDM development board consume much more power and require a minimum of 2.7V. Whereas, for the Renesas based implementation, the LEDs have lower threshold voltage and current ratings (1.7V).

As can be seen from the power architecture (Figure 5.3), the three bucks power the voltage rails connected to the peripherals. The input to the three bucks is a common rail connected to an ultra-cap as well as a DC source, via two switches. Based on the configuration, the DC source can be used to supply the buck converters directly and/or charge the ultra-cap. Once

the ultra-cap is completely charged, the DC source can be isolated and the ultra-cap can power the system through the three bucks. This setup is required for average power calculations of the system, by measuring the discharge time of the ultra-cap, as explained earlier.

A similar setup is used, where the three bucks are replaced with a single linear regulator with 3.3V output to all loads. This helps in analyzing the average power in conventional power architecture. Detailed explanations would be given in Section 5.4.

5.3 RTOS Porting - SMPS

With the goal of embedding SMPS control on the application MCU, some sort of scheduling technique using a real-time operating system can help expand its application spectrum and applicability. Many existing embedded applications employ RTOS for timing various tasks. If the control software can be ported to these schedulers, it would allow efficient power conversion and voltage scaling to be performed on a wide range of real-time applications. Thus, the real-time aspects must be quantified and accurately measured. The analysis presented in Section 3.4.1 allows further investigation into the schedulability and incorporation of the SMPS control software into an RTOS, as well as its effect on the hardware response and regulation in real-time. For the purpose of demonstration of portability of the control firmware, and functional viability, two RTOS, one preemptive and another simple non-preemptive, were employed on two separate platforms. Both RTOS implement static priority for scheduling tasks in the system. The RTOS porting on two platforms are discusses in the following two subsections.

5.3.1 Renesas RL78/G14

As discussed in Section 5.1.2.1, a run-to-completion (RTC) scheduler is used as an RTOS on the Renesas MCU. In this section we discuss porting of the SMPS control firmware on RTC and its analysis.

In order to have multiple voltage rails, there must be separate buck converters for each rail. Each buck requires a separate set of control signals (PWM signals) and corresponding

control law. Fundamentals involved in software control of buck converter have been discussed in detail in Section 4.1.2.4.

The overall control sequence is outlined in Figure 5.4 and is implemented with both hardware peripherals and software (in ADC_ISR). The control task was implemented in a high-priority interrupt service routine (ISR). The use of an Event Link Controller (ELC) allowed peripherals to trigger each other.

The following figure represents the sequence of events taking place in the SMPS control task for regulation of three voltage domains. The ADC is configured in scan mode, and converts three channels in sequence when triggered by a hardware timer (TAU10) through an event link controller (ELC). Here, TAU10 serves as a PWM master and generates a positive edge for the three PWM slave timer modules (TAU11, TAU12 and TAU13). The PWM master determines the period of all three PWM channels and each slave determines the duty cycle for their respective buck converters. Thus, all three bucks have same switching frequency, but separate duty cycles, based on their operating voltages and corresponding control laws.

The first channel to be sampled by the ADC is the voltage rail corresponding to the MCU. At the end of the conversion of first channel (ch 4), an ADC interrupt (INTAD) is generated. The control software is implemented in an ADC interrupt service routine (ADC_ISR) triggered by this INTAD. While ADC converts ch 5, result from ch 4 is used to calculate the new duty cycle for voltage regulation using PI control law, as explained in Section 4.1.2.4. This new duty cycle is updated on TAU11. The execution time for ADC_ISR is shorter than the ADC conversion time. Thus, at the end of conversion, for ch 5, another ADC_ISR is invoked by the INTAD. Similarly, the duty cycles of TAU12 and TAU13 are updated sequentially.

The ADC conversion and update of the three channels is performed well within one period of TAU10, also the switching period of the three bucks.

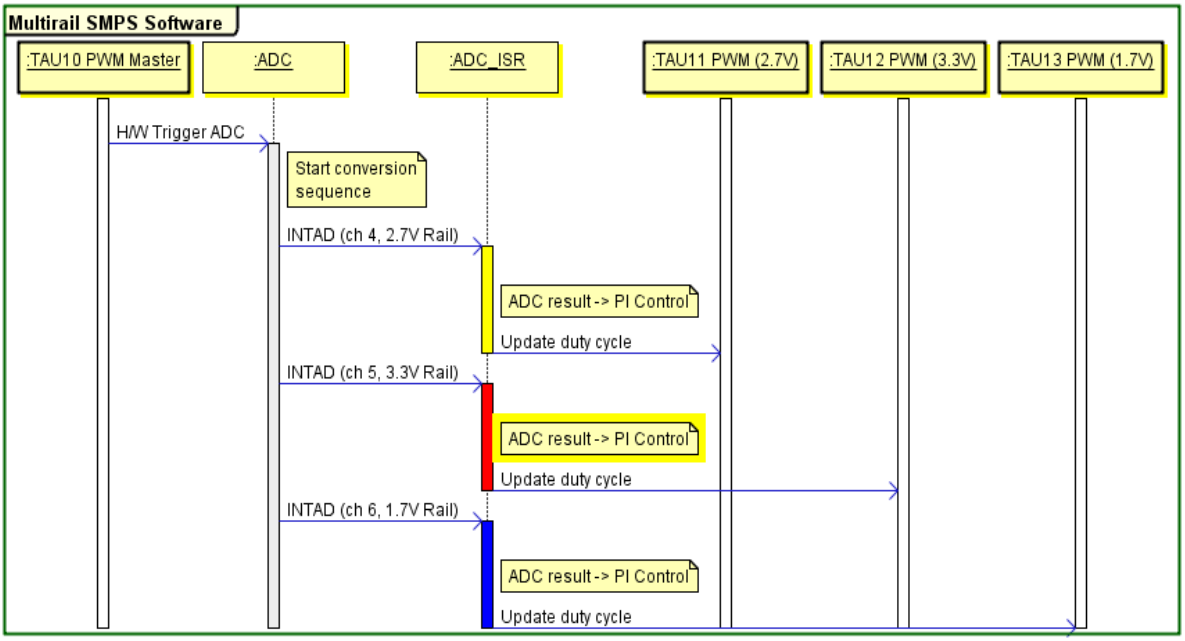


Figure 5.4: SMPS control sequence on RL78/G14 for three buck SMPS

The computation time $\mathbf{t_{control}}$ for one buck was brought down from over 50 µs to only **2.24µs** with the MCU operating at 32MHz. For three domains the total computation time is **4.68µs**. Thus, running these domains consumes only **23.4% of the MCU's time**, leaving most of it free for application processing.

The real-time characteristics of the MCU's software workload are characterized inTable 5-5. WCETs were derived by inspecting the execution of the code on the target hardware with an oscilloscope and debug twiddle bits and identifying the worst case.

Table 5-5: Task and ISR timing details for schedulability analysis for RTC

Sub-system	Task/ISR	WCET (us)	Freq. (Hz)	CPU Util.
RTC Kernel	Tick_timer ISR	16.4	1000	0.0164
SMPS (ADC_ISR)	MCU_SD_LCD ISR	2.24	50000	0.112
	LED ISR	1.72	50000	0.086
	GPS_Wifi_BL ISR	1.72	50000	0.086
NMEA	UART_rcv ISR	4.56	4800	0.021888
	Decode_Task	304	1	0.000304
SD	spi1 _irq ISR	0.94	132000	0.12408
	SD_write_task	203492	1	0.203492
LCD	spi1 _irq ISR	0.94	125000	0.1175
	Update LCD Task	3256	5	0.01628

The ADC_ISR code is given in Appendix A.1.1.

It is worth pointing that the first ADC conversion of ch4 takes a longer time as it incurs a division operation. This is due to the fact that the MCU voltage is being regulated, which is variable and needs to be regulated. Thus, a constant voltage of 1.45V is sampled, whereas the reference voltage is treated to be a variable. Equation (3.1) defining this methodology is explained in Section 3.8.

The remaining two channels use the MCU voltage V_{MCU} determined by first conversion as the reference, V_{ref}, and require a simple multiplication operation for determining the respective rail voltages, V_{buck} (3.3V and 1.7V). These respective voltages for each channel are used to perform PI calculations and determine the new duty cycle for voltage regulation.

The computational load due to these ISR execution times was reduced by performing optimizations explained in Section 4.1.2.4.

It is not possible perform precise schedulability analysis of the task set using the techniques described in Section 4.2.7. This is because the utilization bounds and response time analysis explained earlier assume purely independent tasks, with no data dependencies, no context switch latencies, and purely periodic tasks. In case of the presented system, the tasks have

data dependencies, some tasks are event triggered, and the priorities are purely based on functionality, and do not employ RMS approach when assigning priorities.

Due to data dependencies and manual priority assignments, the deadlines become a little relaxed. For instance, the **SD_write_task** is an event triggered task and not purely periodic as it is '*ready*' to run only after completion of NMEA's **Decode_Task**. This leads to delayed deadlines for this task; therefore, the serialization of tasks allows relaxed deadlines. Thus, it can be safely concluded that if the task set's total MCU utilization, U_{MCU} (equation (3.25)) is approximately close to the utilization bound, U_{max} (=$m(2^{1/m}-1)$). Thus, for this task set of 10 tasks including the ISRs, $\mathbf{U_{MCU}}$ **is 0.78**, whereas, $\mathbf{U_{max}}$ **= 0.72**.

It is quite clear from the utilization test that that task set is schedulable. Without the SMPS task, the system was surely schedulable based on the utilization bound test and its behavior was observed for correct functionality using various GPIO to track extended task activities. In order to ensure timing correctness, the SMPS task is made as highest priority and is enabled within all other ISRs. This allows the ADC_ISR to interrupt any other ISRs, thus the WCRT of the SMPS task becomes 2.24µs for the '*MCU_SD_LCD'* domain, and hence, ensuring good voltage regulation quality. On incorporating the SMPS task, the system application activity was unaffected and the system behaved as expected, with no missed deadlines along with consistent voltage regulation quality.

5.3.2 ARM Cortex M0+ → KL25Z

Similar to the RL78/G14 implementation, the SMPS control software is incorporated in the ADC's ISR with highest priority. The RTOS used for firmware implementation is a scheduler provided by Keil IDE, known as RTX (introduced in Section 5.1.2.2).

Similar to the Renesas based system, there are three voltage rails and for each rail one dedicated buck converter is required. Each of the converters is controlled in software. As mentioned earlier, each buck requires a separate set of control signals (PWM signals) and corresponding control law. Fundamentals involved in software control of buck converter have been discussed in detail in Section 4.1.2.4.

In the KL25Z MCU, due to the absence of scan mode, sequential A/D conversion for the three voltage rails was implemented partly in software with an initial hardware trigger. There are independent PWM modules. The PWM channels belonging to the same module have same periods.

The control sequence for the SMPS is depicted in Figure 5.4. TPM0 is the PWM module with channels 2, 3 and 4. PWM channel 4 controls the 1.8V rail, which powers the MCU, whereas, channels 2 and 3 control 2.7V and 3.3V rails, respectively.

At every positive edge of the TPM0 Ch4, a hardware trigger initiates an ADC conversion, sampling SE0. An ADC_ISR is invoked at the end of conversion and this ISR is responsible for triggering successive A/D conversion of 2.7V rail using a software trigger. During the second A/D conversion, the ISR continues to perform PI calculations based on the 1.7V rail voltage and update the corresponding duty cycle for TPM0 ch4. Subsequently, at the end of A/D conversion of 2.7V rail, the ADC_ISR is again invoked. This leads to successive software trigger for 3.3V rail A/D conversion and PI calculations and update of PWM duty cycle for TPM0 Ch2 (2.7V rail). Similarly, the duty cycle for 3.3V rail is also updated during the next invocation of ADC_ISR. During the third invocation of ADC_ISR, the ADC is again initialized for hardware trigger, to be invoked by TPM0 Ch4 PWM's positive edge.

The implemented C-code for the ADC_ISR is presented in the Appendix A.2.1.

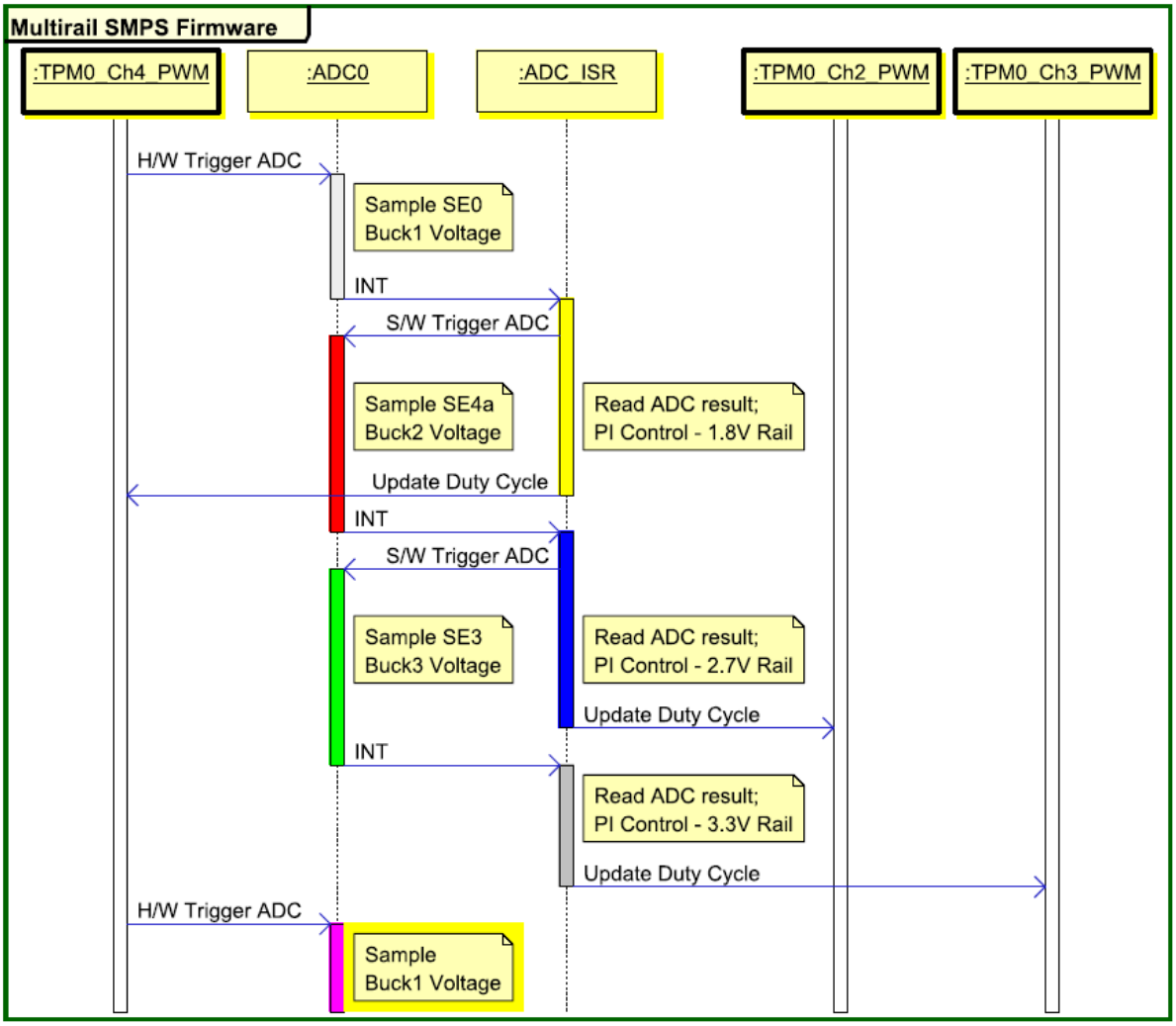


Figure 5.5: SMPS control sequence as implemented on KL25Z for three buck SMPS

Similar to the Renesas implementation, Ch4 requires a constant voltage to be sampled with the variable MCU reference voltage, which requires a division operation to determine the MCU voltage. On the KL25Z, the division operation takes a very long time and makes it impossible to complete sampling and control of three buck converters within a switching period.

In order to eliminate the division operation, a prepopulated lookup table was used for determining MCU voltage. Due to data memory segment limitations, a small lookup table

with an offset and reduced precision ADC values was created. This means that only 8 most significant bits out the 16-bit ADC result were used to determine the actual voltage using the lookup table. In the current system, as per the operating points and control precision required, there was no loss of data in this process and the regulation quality was retained. The pseudo code for creating the lookup table at compile time is given below:

```
#define ADC_BITS (uint32_t)(8)
#define MAX_ADC (uint32_t)(1<<(ADC_BITS))
#define CONST_VOLT (float)(1.5)
#define MAX_VOLTAGE (float)(5.0)
#define ADC_OFFSET (uint32_t)((CONST_VOLT)*(MAX_ADC)/MAX_VOLTAGE)
#define TABLE_SIZE (uint32_t)(MAX_ADC - ADC_OFFSET)+2
#define FXP_Qn (12)
uint32_t lookup_MCU_volt[TABLE_SIZE] = {0};
```

The following formula was used to populate the look-up table during the ADC initialization process, which occurs only once at boot-time.

```
lookup_MCU_volt[i] = (uint32_t) ( ((CONST_VOLT * MAX_ADC)* (1<<FXP_Qn) ) / (float)(i+ADC_OFFSET));
```

The following formula was used to access the table with the *ADC_result* value:

```
lookup_MCU_volt[ (ADC_result>>(16-ADC_BITS)) - ADC_OFFSET ];
```

A control flow diagram presented in Figure 5.6 for the assembly code generated by Keil for ADC_ISR was created using a commercially available code analysis tool known as IDA PRO (introduced in Section 3.8.1). This CFG helped in determining the WCETs of various occurrences of the ADC_ISR based on the methodology explained earlier in Section 3.8.1. The details of all tasks and ISRs implemented on RTX are tabularized in Table 5-6.

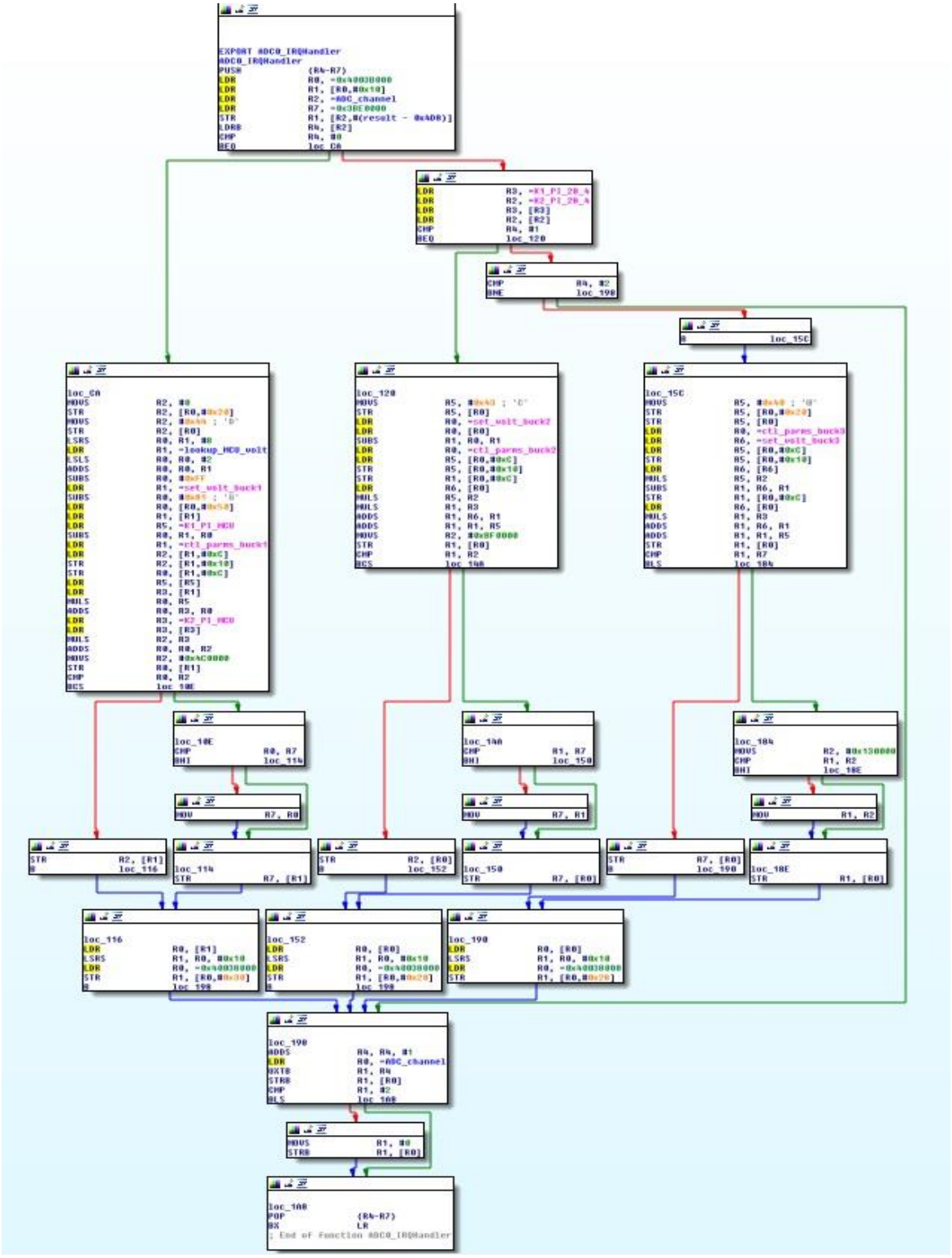


Figure 5.6: CFG generated by IDA PRO for ADC_ISR's assembly code for KL25Z

Table 5-6: Task and ISR timing details for schedulability analysis for RTX

Sub-system	Task/ISR	WCET (us)	Freq. (Hz)	CPU Util.
RTX Kernel	Tick_timer ISR	4.72	1000	0.005
SMPS	MCU ISR	2.72	50000	0.136
	LED_SD_LCD ISR	2.32	50000	0.116
	GPS_BL ISR	2.32	50000	0.116
NMEA	UART_rcv ISR	7	4800	0.034
	Decode_Task	4786	1	0.005
SD	spi1 _irq ISR	1	132000	0.132
	SD_write_task	57288	1	0.057
LCD	spi1 _irq ISR	1	125000	0.125
	Update LCD Task	12250	5	0.061

The WCRT for the SMPS task was determined to be (2.72+7)µs. The ISRs in RTX are not nested. Even though the SMPS task is of highest priority, the WCRT is affected by the longest ISR, which might lead to blocking. The effects of blocking are discussed in Section 3.4.2.3. In this case, the longest ISR is **UART_rcv ISR,** with 7µs of WCET, contributing to the WCRT of the SMPS. Even with this amount of WCRT, the system was stable and schedulable, with good voltage regulation.

A conservative Rate Monotonic analysis tool [38] was employed to determine the scheduling feasibility of the task set. In one case, it was assumed that only periodic tasks are ready to run simultaneously at the start of simulation, whereas the event triggered tasks become ready at a later stage. In the second case, the worst case scenario was simulated, where; it was assumed that all tasks are ready to run simultaneously at the start of simulation. In both cases, the task set was found to be schedulable with the worst case response times and simulation screenshot highlighted in Figure 5.7 for the first case and Figure 5.8 for the second. The unit of time is a microsecond in this simulation.

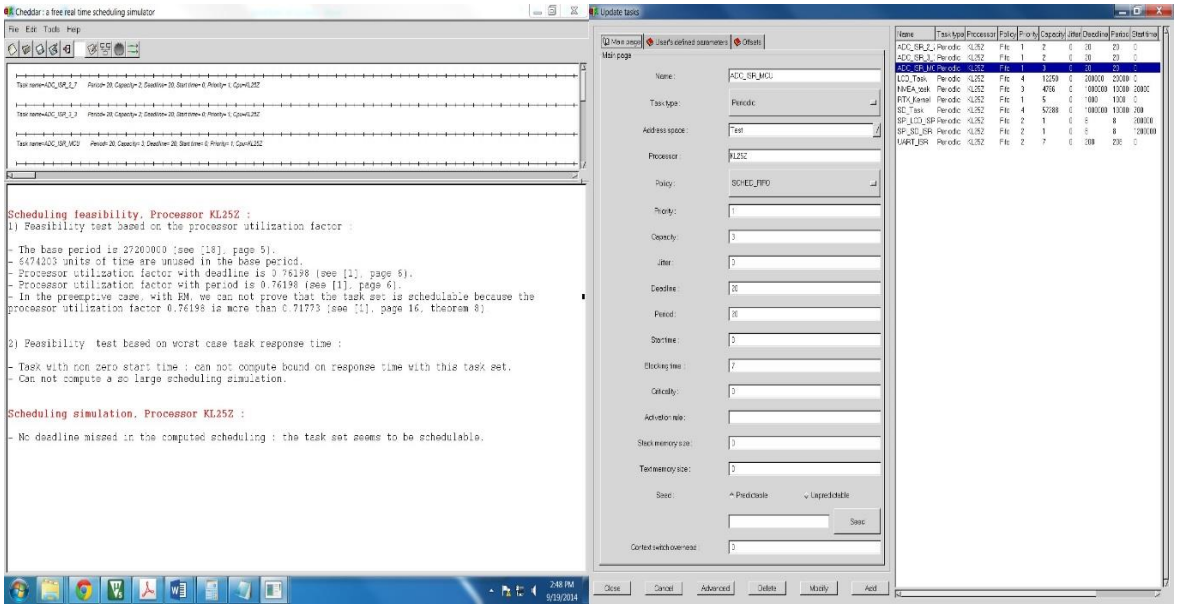


Figure 5.7: Schedulability analysis simulation for all periodic tasks ready at start

```
Scheduling simulation, Processor KL25Z :

- No deadline missed in the computed scheduling : the task set seems to be schedulable.


Scheduling feasibility, Processor KL25Z :
1) Feasibility test based on the processor utilization factor :

- The base period is 13000000 (see [18], page 5).
- 3094288 units of time are unused in the base period.
- Processor utilization factor with deadline is 0.76198 (see [1], page 6).
- Processor utilization factor with period is 0.76198 (see [1], page 6).
- In the preemptive case, with RM, we can not prove that the task set is schedulable because the
processor utilization factor 0.76198 is more than 0.71773 (see [1], page 16, theorem 8).


2) Feasibility  test based on worst case task response time :

- Bound on task response time :  (see [2], page 3, equation 4).
    NMEA_task => 239598
    SD_Task => 192478
    LCD_Task => 33920
    RTX_Kernel => 54
    UART_ISR => 32
    ADC_ISR_2_7 => 20
    ADC_ISR_3_3 => 16
    ADC_ISR_MCU => 14
    SPI_LCD_ISR => 2
    SPI_SD_ISR => 1
- All task deadlines will be met : the task set is schedulable.
```

Figure 5.8: Schedulability analysis simulation where all tasks ready at start

5.4 Comparison with Linear Regulator

After enabling multiple voltage rails in the system, it is essential to highlight the implementation benefits and drawbacks in comparison to the system with conventional linear regulation of single voltage domain. In order to see the effects of having multiple rails, we also draw comparison with a system having single voltage rail, as the linear regulator, instead using a single buck converter to power the rail.

The purpose is to compare the same embedded application powered using three different conversion methodologies. One will have the conventional linear regulator to power a common 3.3V bus rail, second will replace the linear regulator with a single buck controlled in software, whereas, the third will have multiple voltage rails controlled by a dedicated SMPS consisting of three bucks converters operating at different voltages for devices grouped as explained in Section 5.2.

In order to show a wide range of applicability of the proposed methodologies, we employ two different platforms implementing the same application, but using different RTOS and MCUs.

We compare and contrast the power conversion methodologies by powering these implementations with a fully charged ultra-capacitor and determining their respective operating durations given that constant energy. The theory behind this is explained earlier in Section 3.6.

5.4.1 Renesas RL78/G14

Three ultra-caps, totaling 3F capacity (1F each) were employed to power the system. The total energy available is constant for all test cases. The ultra-caps were charged for a fixed amount of time to voltage $\mathbf{V_{init}}$ and then allowed to discharge by powering the system. The amount of time, $\boldsymbol{t_{discharge}}$, taken by the system to drop the voltage of the ultra-caps to $\mathbf{V_{final}}$ determines the rate of energy consumption or the average power, $\boldsymbol{P_{avg}}$ consumed by the system, as given in equation (3.44)

For the three power conversion methodologies, this constant energy was used to power the system and the discharge time was recorded. The intermediate discharge times of dropping

voltages when the device is operating in the sleep mode are presented for the three methods (scaled down for 1F total capacity) in Figure 5.9.

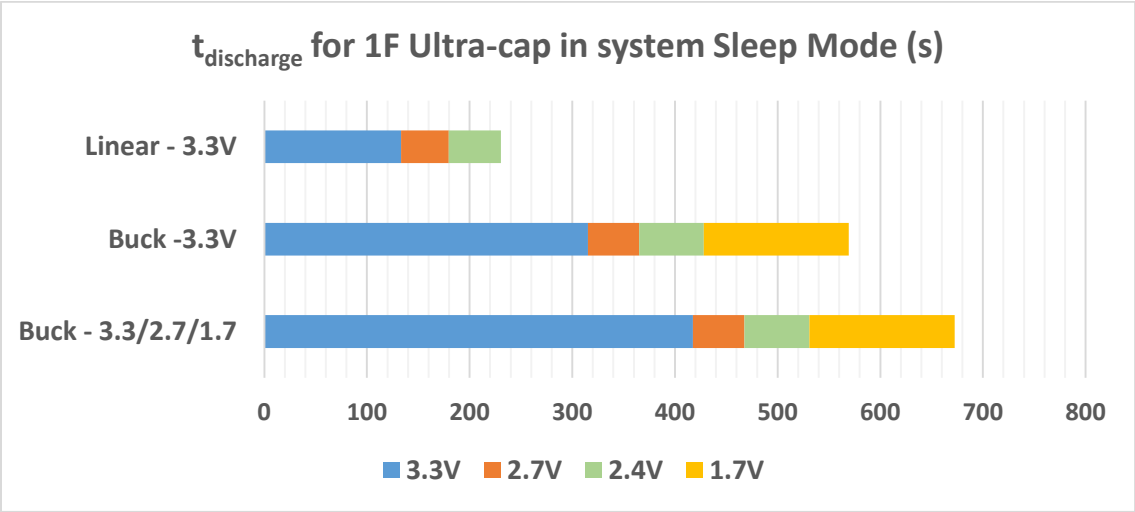


Figure 5.9: 1F Ultra-cap discharge times in system sleep mode - RL78/G14

The average power reduction in various operating modes of the embedded application is normalized and depicted in Figure 5.10 and Figure 5.11. It can be clearly seen that using an SMPS with single voltage domain over a linear regulator results in up to **59% of power savings**. Whereas, having multiple domains results in **66% of power savings**. When comparing the use of single vs multiple voltage rail SMPS design, up to **26% power savings** can be observed, as depicted in **Error! Reference source not found.**. Although these power ains are modest, they were achieved with three non-synchronous buck converters with limited efficiency. Converting them to synchronous converters would improve efficiency significantly with negligible software impact. Furthermore, load balancing on supply rails can significantly improve the average power consumption in all modes of operation. In the current system, most of the load is on 3.3V rail (89.15%), resulting in little improvement of

multi-rail over single rail SMPS. Depending on low power applications, the load can be distributed to 1.7V and 2.7V rails.

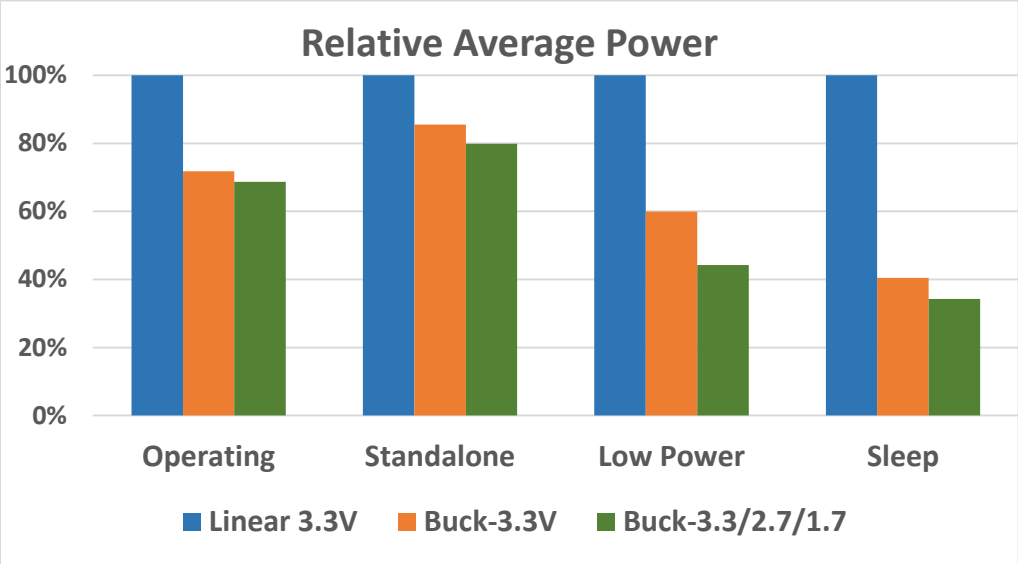


Figure 5.10: Relative average power of the three converter setups in various operating modes – RL78/G14

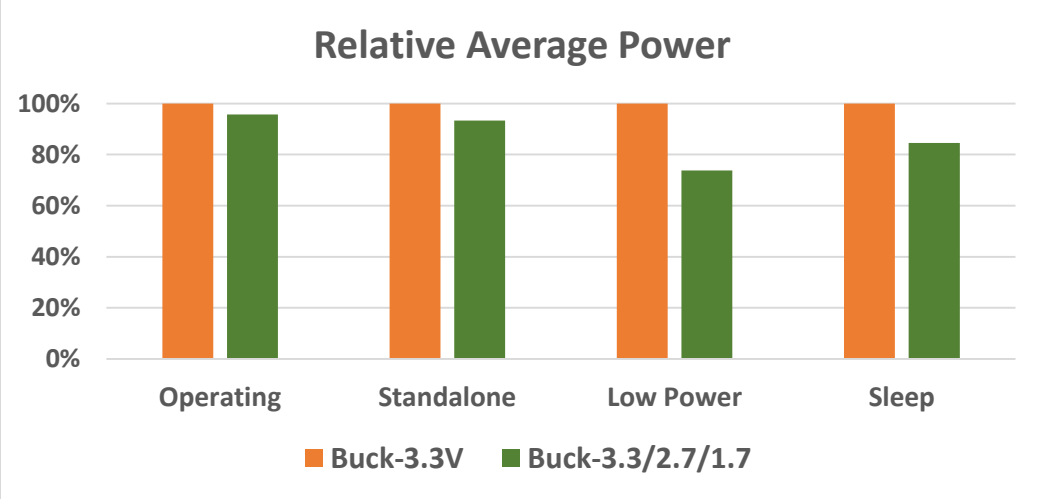


Figure 5.11: Comparison of average power between single SMPS VS Multi-domain SMPS system – RL78/G14

5.4.2 ARM Cortex M0+ → KL25Z

Similar experiments were carried out on the KL25Z platform based implementation, comparing the three power conversion methodologies. The discharge times for the 1F ultra-cap charged to 4.3V are presented in Figure 5.12.

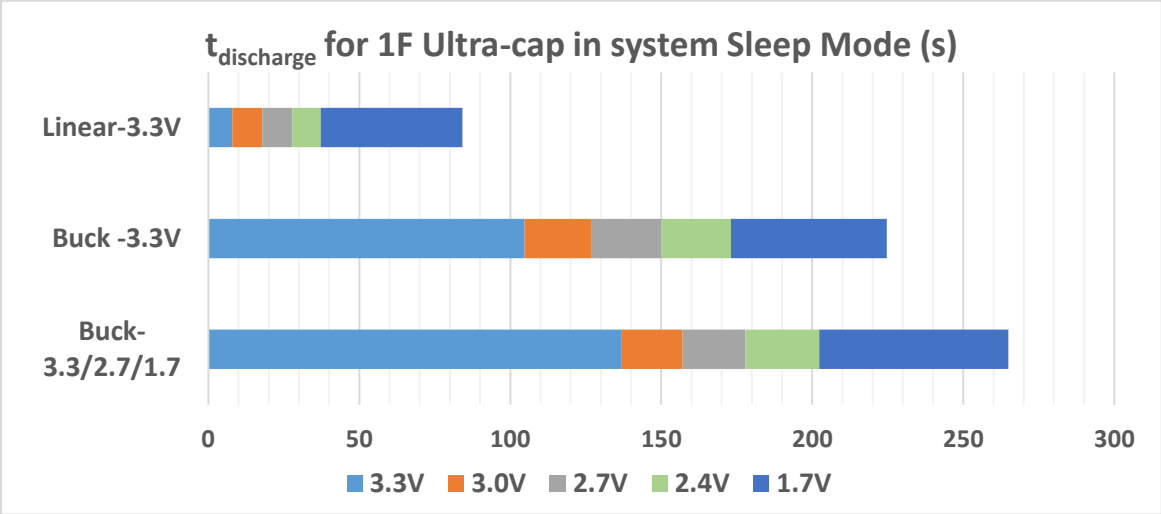


Figure 5.12: Ultra-cap discharge times in system sleep mode - KL25Z

The power used by the system in each domain in each operating mode is evaluated using previously discussed methodology. The calculated average power by measuring the discharge time of a 3 F ultra-capacitor from 4.3 V to the terminal voltage is shown in Figure 5.13. Note that this includes the power consumed by the three buck converters and the MCU. Figure 5.14 shows that no only replacing the linear regulator, but providing three supply rails rather than one (with a buck converter) further decreases power significantly (by up to nearly **one half**). The relatively low improvement in the configuration mode results from the very light load (about 2 mA) on the 2.7V rail and can be addressed by improving the low power efficiency of the buck converter with software methods (e.g. burst-mode control).

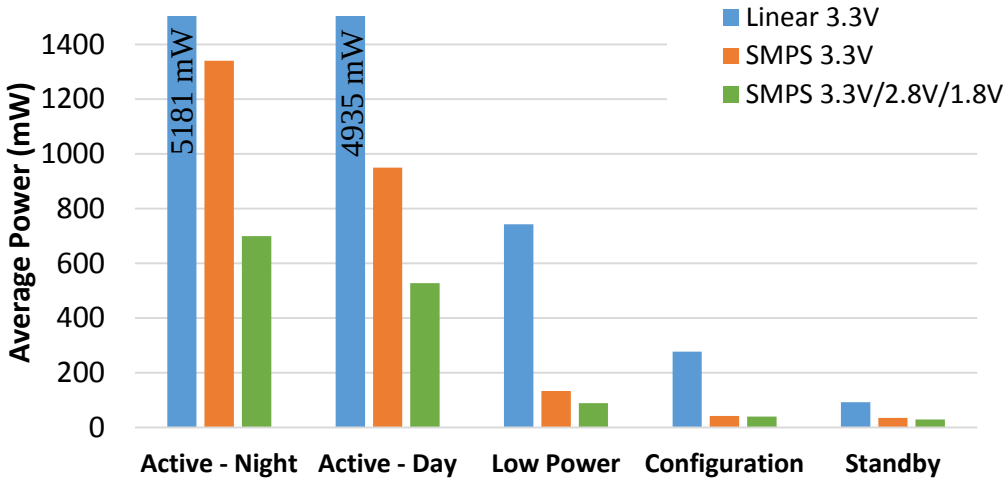


Figure 5.13: Average power of the three converter setups in various operating modes – KL25Z

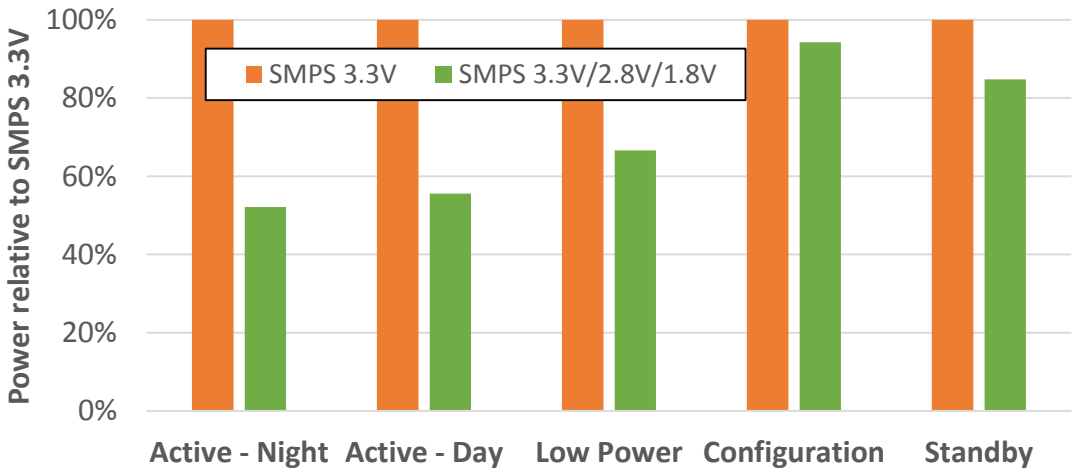


Figure 5.14: Comparison of average power between single SMPS VS Multi-domain SMPS system - KL25Z

The reduction in power consumption by incorporating multiple domains with switching converters as compared with a single domain switching converter, in different modes of device operating on the KL25Z based system is up to **48%**, as shown in Figure 5.15.

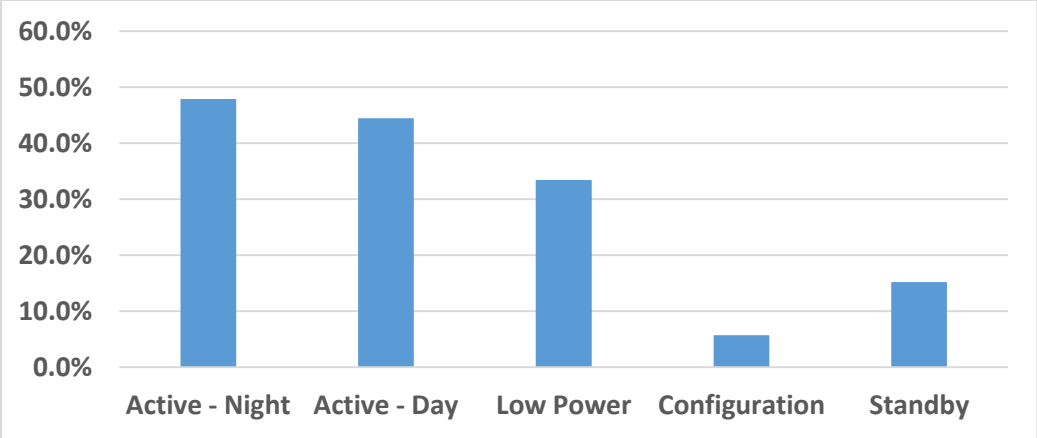


Figure 5.15: Power reduction in multi-rail vs single rail buck converter implementation

5.5 Cost Analysis

In this section a cost comparison is drawn between different power conversion techniques. A tool by Texas Instruments, known as Webench, was used to determine commercially available LDOs, Buck converter controller ICs, and buck converter components and a cost comparison was made between various configurations of power supplies and compared with respect to the proposed techniques. A detailed background is presented in Section 3.9.

An operating range was chosen as a benchmark for comparison between various power supplies. Here, 1ku (1000 units) is a base quantity used for cost comparison. The following table presents some of the commercially available LDOs in the required operating range.

Amongst the following LDOs, LM-1587 is a suitable match and its cost is approximately $0.94 1ku.

Table 5-7: List of commercially available LDOs in required operating range

Compare Parts	LP2951-N	LP2951JAN-SP	LP2952A	LP2953A	LP2952-N	LP2953	LP38691-Q1	LP38690	LP38692	LMS8117A	LMS1587
Output Options	Adjustable Output Fixed Output	Adjustable Output Fixed Output	Adjustable Output Fixed Output	Adjustable Output Fixed Output	Adjustable Output Fixed Output	Adjustable Output Fixed Output	Fixed Output	Fixed Output	Fixed Output	Adjustable Output Fixed Output	Adjustable Output Fixed Output
Iout (Max) (A)	0.1	0.1	0.25	0.25	0.25	0.25	0.5	1	1	1	3
Vin (Min) (V)	-0.3	-0.3	-20	-20	-20	-20	2.7	2.7	2.7	2.5	2.5
Vin (Max) (V)	30	30	30	30	30	30	10	10	10	15	13
Fixed Output Options (V)	3 3.3 5	3 3.3 5	3.3 5	3.3 5	3.3 5	3.3 5	1.8 2.5 3.3 5	1.8 2.5 3.3 5	1.8 2.5 3.3 5	1.8 3.3	1.5 3.3
Vout (Min) (V)	1.24	1.24	1.23	1.23	1.23	1.23	1.8	1.8	1.8	1.5	1.25
Vout (Max) (V)	29	29	29	29	29	29	5	5	5	13.75	5.75
Additional Features	Enable Overcurrent Protection Thermal Shutdown Power Good	Enable Overcurrent Protection Thermal Shutdown Power Good	Thermal Shutdown Overcurrent Protection Enable Power Good Reverse Voltage Protection	Enable Overcurrent Protection Thermal Shutdown Power Good	Enable Overcurrent Protection Thermal Shutdown Power Good	Enable Overcurrent Protection Thermal Shutdown Power Good	Enable Overcurrent Protection Thermal Shutdown Foldback Overcurrent Protection	Enable Overcurrent Protection Thermal Shutdown Foldback Overcurrent Protection	Enable Overcurrent Protection Thermal Shutdown Foldback Overcurrent Protection	Overcurrent Protection Thermal Shutdown	Overcurrent Protection Thermal Shutdown
Rating	Catalog	Space	Catalog	Catalog	Catalog	Catalog	Automotive	Catalog	Catalog	Catalog	Catalog
Operating Temperature Range (C)	-40 to 125				-40 to 125	-40 to 125	-40 to 125	-40 to 125	-40 to 125	0 to 125	-40 to 125 0 to 125
Package Group	SOIC VSSOP PDIP WSON				SOIC	PDIP SOIC	WSON	TO-252 WSON	SOT-223 WSON	SOT-223 TO-252	DDPAK/TO-263 TO-220
Estimated Package Size (WxL) (mm²)	8VSSOP: 3 x 3: 15 mm2 8WSON: 4 x 4: 16 mm2 8SOIC: 3.91 x 4.9: 29 mm2 8PDIP: 6.35 x 9.81: 82 mm2				16SOIC: 3.91 x 9.9: 59 mm2	16SOIC: 3.91 x 9.9: 59 mm2 16PDIP: 6.35 x 21.755: 182 mm2	6WSON: 3 x 3: 9 mm2	6WSON: 3 x 3: 9 mm2	6WSON: 3 x 3: 9 mm2 5SOT-223: 3.56 x 6.5: 36 mm2	4SOT-223: 3.5 x 6.5: 36 mm2	3DDPAK/TO-263: 8.41 x 10.18: 106 mm2 3TO-220: 10.16 x 14.986: 182 mm2
Approx. Price (US$)	0.25 | 1ku				0.80 | 1ku	1.01 | 1ku	0.65 | 1ku	0.53 | 1ku	0.53 | 1ku	0.41 | 1ku	0.94 | 1ku

Table 5-8 draws a comparison between a commercially available switching regulator and an LDO of similar ratings. It is clearly seen that although the switching converter is 40% more efficient than the LDO, the disadvantage due to the cost of a switching regulator eliminates its use in cost-sensitive systems. It should be noted that the regulator IC requires additional components for power supply components, which further increase the cost of the system.

Table 5-8: Comparison between a Switching Regulator and LDO

Switching Regulator	
TPS84621	
Design Note	6A Synchronous...
Topology	Buck
Footprint (mm2)	0
Efficiency (%)	95%
Frequency (kHz)	780
IC Cost	$4.15

LDO Regulator	
LMS1587-3.3	
Design Note	3A Low Dropout ...
Topology	LDO
Max Current	3.00
Pk Efficiency	55%
IC Cost	$0.94

Table 5-9 shows the characteristics of some commercially available switching converter ICs in the suitable range. Table 5-10 show the cost comparison of those ICs and the total cost of the power converter (BOM cost) with converter components.

Table 5-9: Various switching converter ICs and their operating characteristics

Part	**Design Consideration**	**Iout Max(A)**	**Vin Min(V)**	**Vin Max(V)**	**Vout Min(V)**	**Vout Max(V)**	**Min Freq(kHz)**
TPS40305	Synchronous Buck Controller	25	3	20	0.6	18	1000
LM27403	High Current Synchronous Buck Controller	40	3	20	0.6	18.6	50
LM25118	Wide Range BuckBoost controller	20	3	42	1.23	38	50
LM25118	Wide Range BuckBoost controller	20	3	42	1.23	38	50

Table 5-10: Various switching converter ICs with efficiency and cost comparison

Part	Design Consideration	Efficiency (%)	IC Cost ($)	BOM Cost ($)
TPS40305	Synchronous Buck Controller	90	1.1	2.11
LM27403	High Current Synchronous Buck Controller	94	1.2	2.09
LM25118	Wide Range BuckBoost controller	76	2.4	15.44
LM25118	Wide Range BuckBoost controller	80	2.4	4.6

It is clear that the efficiency of switching converters exceed those of the LDOs by up to 40%. Although, the cost of controller ICs for these switching converters is comparable to that of the LDO (~$1.00), the total cost of the switching converter (BOM cost) is quite high (approx. double).

Thus, if the cost of the converter is only confined to the components' cost, and the IC cost can be eliminated, the switching converters are ideal for power conversion in cost sensitive systems. The cost of the IC is eliminated by use of the application MCU for controlling the converter. Hence, the proposed methodologies for incorporating SMPS control in application MCU is an appropriate and efficient approach in increasing the battery life of such real-time systems as well as maintaining the low cost of the system.

Another trade off to be considered is that of incorporating multiple domain SMPS vs a single voltage domain. With multiple domains, the cost of the components increases and so does the MCU utilization. The disadvantage of increased cost is compensated by increase in the efficiency and power reduction of the system (Section 0). Hence, a right balance has to made and various trade-offs must be kept in mind while coming up with a suitable power architecture and power conversion methodology in a system.

Chapter 6

Potential Future Work and Optimizations

There are various other techniques to enable the SMPS control for higher efficiency, performance and flexibility of control. These optimizations are based on existing techniques. With a complete understanding of the system (hardware and software interactions), it is possible to take the research further to enhance the system. Some of the existing techniques are listed below and were also explained in out prior analysis.

6.1 Aggressive Voltage Scaling

Analysis and some experiments on voltage scaling and voltage margin were presented earlier in Sections 3.5 and 4.3.2. Changing control loop update rate and seeing its effect on voltage margin gives insight into how much voltage scaling can be performed in controlling a voltage domain without hindering its performance and correctness. This analysis enables and equips the real-time design community to implement voltage scaling in low-cost embedded systems, which is out of reach in current low cost applications.

6.2 Deadband Control

Some hardware possesses the capability of performing selective AD conversions using voltage comparators. What this means is that if the sampled voltage lies within a dead-band, the ADC ISR will not be triggered. This property can be exploited for SMPS control by reducing the amount of unnecessary control calculations for voltage regulation. Thus, in a steady state, when there are no voltage fluctuations, it makes sense to operate at a constant duty cycle without having to recalculate and execute the control algorithm. This area can be further explored to identify its impact on software and hardware response.

6.3 Voltage Positioning

Voltage positioning is a technique of re-positioning the set-point or the reference voltage of the SMPS to a level such that it tolerates possible loading and unloading transients without violating the operating range. If a device has a narrow operating voltage range, or it operates near its threshold, it is impossible for it to tolerate any voltage transients without malfunctioning. For instant, if the device is operating at a constant voltage and encounters a loading transient, the voltage will drop momentarily and will retain its original level. But with voltage positioning, on encountering a loading transient the operating point can be lowered. This is because, after loading, the maximum additional loading transients have been reduced, reducing the peak possible voltage deviation, allowing operation at a lower voltage set-point. If the set-point is lowered, the voltage reached during the overshoot when responding to unloading transients will be smaller, improving voltage regulation performance. Similarly, after encountering an unloading transient, the set-point is raised to tolerate future loading transients. Thus, it is necessary to study the response of the system on lowering and raising the set-point, as was done in 4.2.5.

This technique can be further supplemented with aggressive voltage scaling to allow higher tolerance and change the set-point accordingly.

6.4 Pulse Frequency Modulation (PFM)

PFM is primarily used for increasing converter efficiency at light loads. But the primary objective of this study is to use PFM for reducing computational loading. This work has made an attempt to understand the connection between the transient, voltage set-point, minimum voltage, and the minimum control frequency. The prior work, in this research, on the effect of changing control loop update rate, $1/f_c$ (3.4.2.2 and 4.2.4) and WCET (3.4.1.1) forms the basis of the analysis of PFM in this area. It is intended to extend the real-time model to support PFM, and then run experiments to validate the model and demonstrate the benefits (reduced computational requirements for the controller).

There are existing techniques which aim at changing the control/switching frequency according to the state of transient. In a steady state, the duty cycle is updated at a much slower rate as compared to the time of occurrence of the load transient. At the time of transient, the control frequency can either be increased to a fixed higher amount or increased proportional to the magnitude of the transient. The latter technique requires a more complex control and frequency determination algorithm, whereas switching between to fixed values of control frequency allows a more predictable control and reduces software and control complexity.

Chapter 7

Conclusion

This work targets embedding the control of switch mode power converters within a low-cost Embedded System, enabling efficient power conversion, dynamic voltage and frequency, and at the same time increasing system's flexibility and scalability in terms of supporting various voltage domains for optimal operating modes.

By using real-time system design methods to integrate control software for an SMPS with the application software, the SMPS designers as well the embedded system design engineers can take advantage of a simple switching power converter in exploiting it for either a tight power management or loosening the control effort in order to extract best CPU (MCU) utilization for other tasks in hand. This significantly empowers low cost system designers to extract more efficiency with significant flexibility and scalability for wide range of applications.

This work has allowed us to understand and answer the two fundamental questions posed for basic systems: 1) How do load transients translate into computational requirements; 2) How can the circuit design and control system are adjusted, in order to ease computational requirements.

A major portion of this study concentrated at examining the co-design of digitally-controlled switched mode power supplies (SMPS) using real-time system methods. Initially, a domain of low-end embedded systems was targeted, where cost pressures limit the available hardware and computational throughput.

It was also demonstrated with successful implementation of a multiple buck SMPS control processing on a microcontroller and validated it with working hardware and software. These methods enable the composition of a computing system which controls one or more buck converters while balancing processing load with robust output voltage levels. The results confirmed the system improvement with the presented energy and power analysis on a real-

time embedded application. The cost of the system was also justified with a comparison drawn between switching converters and LDOs with respect to the proposed methodology.
There are many exciting next steps possible using this work as a foundation. One is to reduce computational loading using analog comparators or conditional ADC interrupts available on many MCUs. Another approach is to reduce the control loop frequency dynamically.
To further strength this power conversion methodology, various enhancements, optimizations and improvements can be explored, such as those mentioned in Chapter 6.

Some of the key advantages as well as limitations of this work in managing power in an embedded system are summarized below.

7.1 Advantages

7.1.1 Dynamic Voltage Scaling

Incorporating the control of power supply in software provides flexibility in controlling and managing power dynamically. This feature can be exploited based on embedded workload activity due to its predictability and firm deadlines.

7.1.2 Scalability of voltage domains

A centralized software control of the SMPS allows incorporation of multiple voltage domains in a system having wide range of components with different optimal operating points. The scalability of the voltage domains and classification of these devices into optimum number of voltage rails is, thus, easily enabled by the proposed method.

7.1.3 Power and Energy Reduction

As proved earlier, multiple voltage domains with a balanced workload results in significant power savings, as opposed to having a single rail powering a wide spectrum of components with different operating ranges.
Even with a single voltage domain, a switching converter is far more efficient than a linear regulator. The proposed methods allow the use of an SMPS at the same cost as a linear

regulator with same number of voltage domains. Thus, power and energy reductions are quite evident without any cost inflation in cost as compared to the linear regulator based system.

7.1.4 Area Cost

The software inclusion of SMPS control within the application MCU eradicates the need of dedicated hardware for the control of power converters. Thus, the area requirements for a software controlled SMPS vs a linear regulator are fairly comparable. On the other hand, a dedicated SMPS control hardware increases the footprint of a power converter quite significantly.

7.1.5 Portability

Portability is a desired feature in any fast evolving system. Having dedicated power converter control hardware restricts upgrading of any embedded system and results in inefficient coupling of application with the power supply. Software control provides flexibility and portability, and results in fast evolution of the system with increased efficiency.

7.2 Limitations

Apart from the advantages, there are quite a few limitations of the proposed methodologies, which must be addressed for future improvements and for providing a holistic solution.

7.2.1 Computational Loading

Software implementation of SMPS control requires MCU resources to be utilized and occupied. This leads to increased MCU utilization and can also starve other application tasks for the time being. Thus, careful RT analysis is required to ensure schedulability and correctness of not only the SMPS firmware, but also the embedded application.

7.2.2 Undesired effects of DVFS

While DVFS is a very desirable knob for efficient power management, it also results in complexity of control. Since the MCU running the embedded application is now responsible

for powering itself, DVFS will result in computational limitations in the control software. For example, if the MCU is placed in a low power mode, it might affect critical MCU resources (PWM frequency, ADC sampling and ISR execution time) responsible for SMPS control. This affects the quality of control and response time of the SMPS to load transients. Thus, the interplay between control software and the embedded application must be carefully taken into account during the implementation of control algorithm with DVFS enabled features in the MCU.

7.2.3 Power state transition latencies

Power state transitions are encountered while dynamically managing power (voltage) during various phases of the application. The state transitions encounter latencies, both, due to hardware and software components. An efficient algorithm is only developed when these latencies are taken into account. As explained earlier, various interdependencies arise between SMPS hardware, software and embedded application, owing to software implementation of SMPS control. These dependencies result in amplified latency calculations for state transitions, thus, complicating the design procedure and decision making for state changes.

7.2.4 Load Characterizations

A reliable software control of SMPS requires a comprehensive analysis of the embedded application, components acting as electrical loads and the SMPS itself. It is a painstaking process to develop an understanding of the system which involves intricate interdependencies between various components of power electronics, embedded firmware, hardware peripherals, and digital control. This raises the complexity of the system quite significantly.

7.2.5 Optimal Load/rail distribution

Multiple voltage rail distribution requires a good balance of electrical load on the rails, in order to see any reasonable efficiency improvement or reduction in power and energy consumption. The operating current range knowledge of all components is necessary along

with their worst case current-voltage transients. This information is used to ensure whether a particular rail regulation can tolerate and compensate for these transients in time. This also provides a fair statistics on the percentage of overall workload which can operate at lower voltages while maintaining correctness in operation and save power.

7.2.6 Development time

Due to many of the above mentioned limitations, the complexity of the systems leads to increased development time. This study can provide quantitative as well as qualitative knowledge necessary for the development of an automated tool, which can automate the design process. Thus, knowledge of embedded loads, power and voltage requirements, MCU features, RTOS, and application firmware can be fed into SMPS design software, which generates the entire power architecture of the system as well as the required SMPS control firmware. If achieved, the development time can be significantly reduced while allowing the user to select various operating points balanced between performance and energy efficiency.

7.2.7 Cost of Multiple rails

Replacing a linear regulator with a switching converter controlled in software doesn't have much impact on the cost and area. However, the division of voltage domains into multiple rails will increase the cost and area requirements proportional to the number of rails. But, the increased number of rails increases power savings due to reduced power and energy consumption. Thus, there's a clear tradeoff between the cost of incorporating multiple rails and the target energy consumption. Hence, an optimal number of power rails have to be determined as per the application requirements.

7.2.8 MCU limitations

There are various limitations posed by the architectural features of the MCU in the control of SMPS. Some of these are enumerated below.

(i) Frequency of operation – The maximum MCU clock frequency leads to a minimum encountered execution latency and response time for the SMPS control software. This in turn restricts the frequency at which the SMPS control can be executed. The control loop frequency is the switching frequency, which is inversely proportional to the voltage ripple. Thus, the MCU frequency has multiple effects on the quality of regulation and the MCU utilization due to the SMPS control software.

(ii) ALU units – The ALU units of the MCU determine the minimum number of clock cycles to execute the SMPS control software. Floating point hardware results in faster and efficient mathematical operations. Whereas, absence of such hardware requires explicit optimizations like implementation of fixed point math operations at the cost of precision, for faster response time and lower MCU utilization. Specific hardware features like Multiply-Accumulate unit, 32-bit multiplication, etc. allows MCU specific optimizations.

(iii) Response time (latencies) – As discussed earlier, many architectural and application specific features can result in increased response times of the control software. Some of the contributors are MCU frequency, task priority, pre-emption, architectural features, sequence of steps for DVFS, operating voltage/frequency, etc... The latencies encountered might result in degradation of regulation quality, and reliability issues.

(iv) PWM and ADC channels - MCU resource availability poses restrictions on the scalability and flexibility of SMPS control. The number of ADC and PWM channels allows multiple domains to be regulated simultaneously. Clock sharing/distribution in these modules determine the flexibility of placing the MCU in various low-power modes and helps isolate the MCU operations from ADC sampling and PWM control signals. Various ADC sampling modes, such as *'scan mode'*, *'sequential sampling'*, *'one-shot mode'*, *'windowed sampling'*, etc. allow various optimizations and implementation design space to be explored.

Largely, this work provides various handles in designing software controlled SMPS for cost-sensitive embedded systems. Detailed quantitative, as well as, qualitative analysis provides a holistic view of the system and offers insight into the advantages and limitations posed by this methodology. Hence an engineer can optimally tailor the power architecture for a real-time embedded application for performance and power targets.

REFERENCES

[1] M. Y. Lim and V. W. Freeh, "Determining the Minimum Energy Consumption using Dynamic Voltage and Frequency Scaling," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, Long Beach, CA, 2007.

[2] W. Kim, M. S. Gupta, G.-Y. Wei and B. D., "System level analysis of fast, per-core DVFS using on-chip switching regulators," in *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, Salt Lake City, UT, 2008.

[3] J. R. Farmer, "A comparison of power harvesting techniques and related energy storage issues," Master's Thesis, Virginia Polytechnic Institute and State University, May 2007.

[4] J. A. Paradiso and T. Starner, "Energy scavenging for mobile and wireless electronics," in *Pervasive Computing*, 2005.

[5] A. Peterchev and S. Sanders, "Load-Line Regulation With Estimated Load-Current Feedforward: Application to Microprocessor Voltage Regulators," *Power Electronics, IEEE Transactions,* vol. 21, no. 6, pp. 1704-1717, 2006.

[6] K. Yao, Y. Ren and F. Lee, "Critical bandwidth for the load transient response of voltage regulator modules," *Power Electronics, IEEE Transactions,* vol. 19, no. 6, pp. 1454 - 1461, 2004.

[7] R. W. Erickson and D. Maksimovic, Fundamentals of Power Electronics, Second Edition, Kluwer Academic Publishers, 2001.

[8] R. Redl, B. Erisman and Z. Zansky, "Optimizing the load transient response of the buck converter," in *Applied Power Electronics Conference and Exposition, 1998. APEC '98. Conference Proceedings 1998., Thirteenth Annual*, 1998.

[9] A. Peterchev, "Digital Control of PWM Converters: Analysis and Application to Voltage Regulation Modules," University of California, Berkeley.

[10] S. Cuk, "Modelling, Analysis and Design of Switching Converters," PhD Thesis, California Institute of Technology, 1976.

[11] R. Middlebrook and S. Cuk, "A General Unified Approach to Modelling Switching-Converter Power Stages," *Internation Journal of Electronics*, vol. 42, pp. 521-550, 1977.

[12] M. Hagen and V. Yousefzadeh, "Applying digital technology to PWM control-loop designs," 2008-2009 Power Supply Design Seminar, 2009.

[13] Y. Duan and H. Jin, "Digital controller design for switchmode power converters," in *Applied Power Electronics Conference and Exposition, 1999. APEC '99. Fourteenth Annual*, 1999.

[14] S. Kapat and P. Krein, "PID controller tuning in a DC-DC converter: A geometric approach for minimum transient recovery time," in *Control and Modeling for Power Electronics (COMPEL), 2010 IEEE 12th Workshop*, 2002.

[15] T. Martin, "Digital control for switching converters," in *Industrial Electronics, 1995. ISIE '95., Proceedings of the IEEE International Symposium on*, 1995.

[16] A. Prodic and D. Maksimovic, "Design of a digital PID regulator based on look-up tables for control of high-frequency DC-DC converters," in *Computers in Power Electronics, 2002. Proceedings. 2002 IEEE Workshop*, 2002.

[17] A. Prodic, D. Maksimovic and R. Erickson, "Design and implementation of a digital PWM controller for a high-frequency switching DC-DC power converter," in *Industrial Electronics Society, 2001. IECON '01. The 27th Annual Conference of the IEEE*, 2001.

[18] M. Khanniche, "An intelligent real-time microcontrolled UPS system," in *Power Electronics and Variable-Speed Drives, 1991., Fourth International Conference*, 1990.

[19] M. Sorescu, "Real time microprocessor control loop for switched mode power supply," in *Semiconductor Conference, 1997. CAS '97 Proceedings., 1997 International*, 1997.

[20] A. Cervin, B. Lincoln, J. Eker, K.-E. Arzen, Buttazzo and Giorgio, "The Jitter Margin and Its Application in the Design of Real-Time Control Systems," in *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Goteborg, 2004.

[21] N. Audsley, I. J. Bate and A. Burns, "Putting fixed priority scheduling theory into engineering practice for safety critical applications," in *Proceedings of the IEEE Real-Time Technology and Applications Symposium (RTAS)*, 1996.

[22] T. Bund, S. Moser, S. Kollman and F. Slomka, "Jitter Considerations for Worst-Case Performance Generation in Digital Controller Design," *Cyber-Physical Systems - Enabling Multi-Nature Systems,* 2012.

[23] V. Reddi, M. Gupta, G. Holloway, G.-Y. Wei, M. Smith and D. Brooks, "Voltage emergency prediction: Using signatures to reduce operating margins," in *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium*, 2009.

[24] R. Joseph, D. Brooks and M. Martonosi, "Control techniques to eliminate voltage emergencies in high performance processors," in *High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings. The Ninth International Symposium*, 2003.

[25] W. Kim, D. Brooks and G.-Y. Wei, "A Fully-Integrated 3-Level DC-DC Converter for Nanosecond-Scale DVFS," *Solid-State Circuits, IEEE Journal,* vol. 47, no. 1, pp. 206 - 219, 2012.

[26] M. Khan and C. Manning, "Microprocessor controlled inverter for UPS applications," in *Power Electronics and Variable-Speed Drives, Third International Conference*, 1988.

[27] N. Rehman, A. Parayandeh, K. Wang and A. Prodic, "Multimode digital SMPS controller IC for low-power management," in *International Symposium on Circuits and Systems, ISCAS*, 2006.

[28] S. Sachidananda and A. G. Dean, "EMI-and Energy -Aware Scheduling of Switching Power Supplies in Hard Real-Time Embedded Systems," in *Real-Time and Embedded Technology and Applications Symposuim (RTAS), 17th IEEE,* 2011.

[29] S. Sachidananda and A. Dean, "Scheduling Swith-Mode Power Supply Noise for Real-Time Systems," in *FREEDM Systems Center Second Annual Conference*, Tallahassee, Florida, 2010.

[30] S. Pontarelli, M. Ottavi, A. Salsano and K. Zarrineh, "Feedback based droop mitigation," in *Design, Automation & Test in Europe Conference & Exhibition*, 2011.

[31] J. Alimeling and W. Hammer, "PLECS-Piece-wise linear electrical circuit simulation for Simulink," in *Power Electronics and Drive Systems (PEDS), Proceedings of the IEEE 1999 International Conference*, 1999.

[32] Z. Leng, Q. Liu, J. Sun and J. Liu, "A research of efficiency characteristic for Buck converter," in *Industrial Mechatronics and Automation (ICIMA), 2010 2nd International Conference on*, Wuhan, China, 2010.

[33] D. Maksimovic and R. Zane, "Small-Signal Discrete-Time Modeling of Digitally Controlled PWM Converters," *Power Electronics, IEEE Transactions on*, vol. 22, no. 6, pp. 2552 - 2556, 2007.

[34] R. Nowakowaski and N. Tang, "Efficiency of synchronous versus nonsynchronous buck converters," Texas Instruments Incorporated, 2009.

[35] "https://www.hex-rays.com," Hex-rays, [Online]. Available: https://www.hex-rays.com/products/ida/ida-executive.pdf. [Accessed 24 01 2015].

[36] "TI's WEBENCH Design Center," Texas Instruments, [Online]. Available: http://www.ti.com/lsds/ti/analog/webench/overview.page?DCMP=analog_power_mr&HQS=webench-pr. [Accessed 24 01 2014].

[37] ARM, "Cortex-M0+ Technical Reference Manual," 16 December 2012. [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0484c/DDI0484C_cortex_m0p_r0p1_trm.pdf. [Accessed 06 2014].

[38] F. Singhoff, "The Cheddar project : a free real time scheduling analyzer," [Online]. Available: http://beru.univ-brest.fr/~singhoff/cheddar/#Ref3.

[39] E. Grochowski, D. Ayers and V. Tiwari, "Microarchitectural simulation and control of di/dt-induced power supply voltage variation," in *High-Performance Computer Architecture, 2002. Proceedings. Eighth International Symposium*, 2002.

[40] M. Anand and I. Lee, "Robust and Sustainable Analysis of Embedded Software," in *ACM SIGPLAN-SIGBED conference on Languages, compilers, and tools for embedded systems*, New York, 2008.

[41] E. Bini and G. C. Buttazzo, "Schedulability analysis of periodic fixed priority systems," *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1462 - 1473, 27 September 2004.

[42] R. Bril, J. Lukkien and W. Verhaegh, "Worst-Case Response Time Analysis of Real-Time Tasks under Fixed-Priority Scheduling with Deferred Preemption Revisited," in *ECRTS '07. 19th Euromicro Conference on Real-Time Systems*, Pisa, 2007.

[43] J. R. Farmer, "A Comparison of harvesting techniques and related energy storage issues.," Master's Thesis, Virginia Polytechnic Institute and State University, May 2007.

[44] S. Han, M. Park and M. Park, "A Sufficient Condition for Rate Monotonic Schedulability Based on Response Time Analysis," in *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference*, Bradford, 2010.

[45] D. Katcher, H. Arakawa and J. Strosnider, "Engineering and analysis of fixed priority schedulers," *IEEE transactions on Software Engineering*, vol. 19, no. 9, pp. 920 - 934, 06 August 2002.

[46] I.-G. Kim, K.-H. Choi, S.-K. Park and D.-Y. Kim, "Real-time scheduling of tasks that contain the external blocking intervals," in *Second International Workshop on Real-Time Computing Systems and Applications, Proceedings*, Tokyo, 1995.

[47] W.-C. Lu, J.-W. Hsieh, W.-K. Shih and T.-W. Kuo, "A faster exact schedulability analysis for fixed-priority scheduling," *ournal of Systems and Software*, vol. 79, no. 12, pp. 1744-1753, December, 2006.

[48] K. Yao, Y. Ren, J. Sun, L. Kisun, X. Ming, J. Zhou and F. C. Lee, "Adaptive Voltage Position Design for Voltage Regulators," in *Applied Power Electronics Conference and Exposition, APEC '04, Nineteenth Annual IEEE*, 2004.

APPENDIX

Appendix A

SMPS CODE

A.1 Renesas – RL78/G14 – IAR Workbench

A.1.1 C-Code

```
__interrupt static void r_adc_interrupt(void)
{

#if FXP
   int16_t t;
#else
   volatile float error; // int16_t
#endif


   switch (ADC_channel){
   case 4:

     ctl_parms_2_7.e[1] = ctl_parms_2_7.e[0];

     t = ((uint32_t) 5939<<16)/ADCR;    //5939.2 = 1.45 in 4_12 format; 389231411 =
5939.2*1024*64

     ctl_parms_2_7.e[0] = set_value_2_7 - t;


     MACRL = 0;
     MACRH = ctl_parms_2_7.d[0];

     // MAC new error and k1
     __mach(K1_PI_12_4, ctl_parms_2_7.e[0]);

     // MAC old error and k2
     __mach(K2_PI_12_4, ctl_parms_2_7.e[1]);

     ctl_parms_2_7.d[0] = MACRH;

     if (ctl_parms_2_7.d[0] > D_MAX) {
       ctl_parms_2_7.d[0] = D_MAX;
     } else if (ctl_parms_2_7.d[0] < D_MIN) {
       ctl_parms_2_7.d[0] = D_MIN;
     }

     TDR11 = ctl_parms_2_7.d[0];//D_local;
     break;
```

```
case 5:

  ctl_parms_3_3.e[1] = ctl_parms_3_3.e[0];

  t = set_value_3_3 - ADCR;

  ctl_parms_3_3.e[0] = (int16_t) (( (int32_t) t * 27034) >> 16) ;

  MACRL = 0;
  MACRH = ctl_parms_3_3.d[0];

  // MAC new error and k1
  __mach(K1_PI_12_4, ctl_parms_3_3.e[0]);

  // MAC old error and k2
  __mach(K2_PI_12_4, ctl_parms_3_3.e[1]);

  ctl_parms_3_3.d[0] = MACRH;

  if (ctl_parms_3_3.d[0] > D_MAX) {
    ctl_parms_3_3.d[0] = D_MAX;
  } else if (ctl_parms_3_3.d[0] < D_MIN) {
    ctl_parms_3_3.d[0] = D_MIN;
  }

  TDR12 = ctl_parms_3_3.d[0];//D_local;
  break;

case 6:

  ctl_parms_1_7.e[1] = ctl_parms_1_7.e[0];

  t = set_value_1_7 - ADCR;

  ctl_parms_1_7.e[0] = (int16_t) (( (int32_t) t * 27034) >> 16) ;

  MACRL = 0;
  MACRH = ctl_parms_1_7.d[0];

  // MAC new error and k1
  __mach(K1_PI_12_4, ctl_parms_1_7.e[0]);

  // MAC old error and k2
  __mach(K2_PI_12_4, ctl_parms_1_7.e[1]);

  ctl_parms_1_7.d[0] = MACRH;

  if (ctl_parms_1_7.d[0] > D_MAX) {
    ctl_parms_1_7.d[0] = D_MAX;
  } else if (ctl_parms_1_7.d[0] < 30) {
    ctl_parms_1_7.d[0] = 30;
  }
```

```
    TDR13 = ctl_parms_1_7.d[0];//D_local;
    break;

  case 7:
    break;

  default:
    break;
  }

  ADC_channel++;
  if(ADC_channel>7)
    ADC_channel = 4;
}
```

A.1.2 Assembly

```
   99         __interrupt static void r_adc_interrupt(void)
  \                 r_adc_interrupt:
   100        {
  \   000000 C1               PUSH     AX              ;; 1 cycle
  \   000001 C3               PUSH     BC              ;; 1 cycle
  \   000002 C5               PUSH     DE              ;; 1 cycle
  \   000003 C7               PUSH     HL              ;; 1 cycle
   101          /* Start user code. Do not edit comment generated here */
   102        #if FXP
   103          int16_t t;
   104        #else
   105          volatile float error; // int16_t
   106        #endif
   107
   111          switch (ADC_channel){
  \   000004 8F....           MOV      A, N:ADC_channel   ;; 1 cycle
  \   000007 2C04               SUB      A, #0x4           ;; 1 cycle
  \   000009 DD0C               BZ       ??r_adc_interrupt_0  ;; 4 cycles
  \   00000B                  ; ------------------------------------- Block: 10 cycles
  \   00000B                  ; * Stack frame (at entry) *
  \   00000B                  ; Param size: 0
  \   00000B                  ; Auto size: 0
  \   00000B 91               DEC      A               ;; 1 cycle
  \   00000C DD5E               BZ       ??r_adc_interrupt_1  ;; 4 cycles
  \   00000E                  ; ------------------------------------- Block: 5 cycles
  \   00000E 91               DEC      A               ;; 1 cycle
  \   00000F 61F8               SKNZ                      ;; 4 cycles
  \   000011 ED....             BR       N:??r_adc_interrupt_2  ;; 4 cycles
  \   000014                  ; ------------------------------------- Block: 5 cycles
  \   000014 ED....             BR       N:??r_adc_interrupt_3  ;; 3 cycles
  \   000017                  ; ------------------------------------- Block: 3 cycles
   112          case 4:    114
   115            ctl_parms_2_7.e[1] = ctl_parms_2_7.e[0];
  \                 ??r_adc_interrupt_0:
  \   000017 AF....           MOVW     AX, N:ctl_parms_2_7+6  ;; 1 cycle
  \   00001A BF....           MOVW     N:ctl_parms_2_7+8, AX  ;; 1 cycle
   116
```

```
  117            t = ((uint32_t) 5939<<16)/ADCR;  //5939.2 = 1.45 in 4_12 format; 389231411
= 5939.2*1024*64
  118
  121            ctl_parms_2_7.e[0] = set_value_2_7 - t;
  \   00001D F6               CLRW      AX              ;; 1 cycle
  \   00001E EA1E              MOVW      DE, S:0xFFF1E      ;; 1 cycle
  \   000020 16               MOVW      HL, AX           ;; 1 cycle
  \   000021 323317             MOVW      BC, #0x1733       ;; 1 cycle
  \   000024 CEFB0B             DIVWU                      ;; 17 cycles
  \   000027 16               MOVW      HL, AX           ;; 1 cycle
  \   000028 AF....             MOVW      AX, N:set_value_2_7  ;; 1 cycle
  \   00002B 27               SUBW      AX, HL           ;; 1 cycle
  \   00002C BF....             MOVW      N:ctl_parms_2_7+6, AX  ;; 1 cycle
  122
  129            MACRL = 0;
  \   00002F CBF00000           MOVW      0xFFFF0, #0x0      ;; 1 cycle
  130            MACRH = ctl_parms_2_7.d[0];
  \   000033 AF....             MOVW      AX, N:ctl_parms_2_7  ;; 1 cycle
  \   000036 BEF2              MOVW      0xFFFF2, AX        ;; 1 cycle

  132            // MAC new error and k1
  133            __mach(K1_PI_12_4, ctl_parms_2_7.e[0]);
  \   000038 DB....             MOVW      BC, N:ctl_parms_2_7+6  ;; 1 cycle
  \   00003B AF....             MOVW      AX, N:K1_PI_12_4   ;; 1 cycle
  \   00003E CEFB06             MACH                       ;; 3 cycles
  134
  135            // MAC old error and k2
  136            __mach(K2_PI_12_4, ctl_parms_2_7.e[1]);
  \   000041 DB....             MOVW      BC, N:ctl_parms_2_7+8  ;; 1 cycle
  \   000044 AF....             MOVW      AX, N:K2_PI_12_4   ;; 1 cycle
  \   000047 CEFB06             MACH                       ;; 3 cycles
  137
  138            ctl_parms_2_7.d[0] = MACRH;
  \   00004A AEF2              MOVW      AX, 0xFFFF2        ;; 1 cycle
  139
  140            if (ctl_parms_2_7.d[0] > D_MAX) {
  \   00004C 447F02             CMPW      AX, #0x27F         ;; 1 cycle
  \   00004F BF....             MOVW      N:ctl_parms_2_7, AX  ;; 1 cycle
  \   000052 DC05               BC        ??r_adc_interrupt_4  ;; 4 cycles
  \   000054                  ; ------------------------------------- Block: 47 cycles
  141              ctl_parms_2_7.d[0] = D_MAX;
  \   000054 307E02             MOVW      AX, #0x27E         ;; 1 cycle
  \   000057 EF08               BR        S:??r_adc_interrupt_5  ;; 3 cycles
  \   000059                  ; ------------------------------------- Block: 4 cycles
  142            } else if (ctl_parms_2_7.d[0] < D_MIN) {
  \                          ??r_adc_interrupt_4:
  \   000059 443200             CMPW      AX, #0x32          ;; 1 cycle
  \   00005C DE06               BNC       ??r_adc_interrupt_6  ;; 4 cycles
  \   00005E                  ; ------------------------------------- Block: 5 cycles
  143              ctl_parms_2_7.d[0] = D_MIN;
  \   00005E 303200             MOVW      AX, #0x32          ;; 1 cycle
  \   000061                  ; ------------------------------------- Block: 1 cycles
  \                          ??r_adc_interrupt_5:
  \   000061 BF....             MOVW      N:ctl_parms_2_7, AX  ;; 1 cycle
```

```
\  000064                  ; ------------------------------------ Block: 1 cycles
 144            }

 147            TDR11 = ctl_parms_2_7.d[0];//D_local;
\               ??r_adc_interrupt_6:
\  000064 AF....             MOVW      AX, N:ctl_parms_2_7  ;; 1 cycle
\  000067 BE72                MOVW      0xFFF72, AX        ;; 1 cycle
 150            break;
\  000069 ED....              BR        N:??r_adc_interrupt_3  ;; 3 cycles
\  00006C                   ; ------------------------------------ Block: 5 cycles
 151
 152          case 5:
 153
 154            ctl_parms_3_3.e[1] = ctl_parms_3_3.e[0];
\               ??r_adc_interrupt_1:
\  00006C AF....             MOVW      AX, N:ctl_parms_3_3+6  ;; 1 cycle
\  00006F BF....             MOVW      N:ctl_parms_3_3+8, AX  ;; 1 cycle
 155
 156            t = set_value_3_3 - ADCR;
 157
 158            ctl_parms_3_3.e[0] = (int16_t) (( (int32_t) t * 27034) >> 16) ;
\  000072 AF....             MOVW      AX, N:set_value_3_3  ;; 1 cycle
\  000075 261E                SUBW      AX, S:0xFFF1E      ;; 1 cycle
\  000077 329A69              MOVW      BC, #0x699A        ;; 1 cycle
\  00007A CEFB02              MULH                         ;; 2 cycles
\  00007D 13                 MOVW      AX, BC             ;; 1 cycle
\  00007E BF....             MOVW      N:ctl_parms_3_3+6, AX  ;; 1 cycle
 159
 160            MACRL = 0;
\  000081 CBF00000            MOVW      0xFFFF0, #0x0      ;; 1 cycle
 161            MACRH = ctl_parms_3_3.d[0];
\  000085 AF....             MOVW      AX, N:ctl_parms_3_3  ;; 1 cycle
\  000088 BEF2                MOVW      0xFFFF2, AX        ;; 1 cycle
 162
 163            // MAC new error and k1
 164            __mach(K1_PI_12_4, ctl_parms_3_3.e[0]);
\  00008A AF....             MOVW      AX, N:K1_PI_12_4   ;; 1 cycle
\  00008D CEFB06              MACH                         ;; 3 cycles
 165
 166            // MAC old error and k2
 167            __mach(K2_PI_12_4, ctl_parms_3_3.e[1]);
\  000090 DB....              MOVW      BC, N:ctl_parms_3_3+8  ;; 1 cycle
\  000093 AF....             MOVW      AX, N:K2_PI_12_4   ;; 1 cycle
\  000096 CEFB06              MACH                         ;; 3 cycles
 168
 169            ctl_parms_3_3.d[0] = MACRH;
\  000099 AEF2                MOVW      AX, 0xFFFF2        ;; 1 cycle
 170
 171            if (ctl_parms_3_3.d[0] > D_MAX) {
\  00009B 447F02              CMPW      AX, #0x27F         ;; 1 cycle
\  00009E BF....             MOVW      N:ctl_parms_3_3, AX  ;; 1 cycle
\  0000A1 DC05                BC        ??r_adc_interrupt_7  ;; 4 cycles
\  0000A3                   ; ------------------------------------ Block: 28 cycles
 172              ctl_parms_3_3.d[0] = D_MAX;
```

```
\   0000A3 307E02             MOVW      AX, #0x27E        ;; 1 cycle
\   0000A6 EF08               BR        S:??r_adc_interrupt_8  ;; 3 cycles
\   0000A8                  ; ------------------------------------ Block: 4 cycles
 173            } else if (ctl_parms_3_3.d[0] < D_MIN) {
\               ??r_adc_interrupt_7:
\   0000A8 443200             CMPW      AX, #0x32         ;; 1 cycle
\   0000AB DE06               BNC       ??r_adc_interrupt_9  ;; 4 cycles
\   0000AD                  ; ------------------------------------ Block: 5 cycles
 174              ctl_parms_3_3.d[0] = D_MIN;
\   0000AD 303200             MOVW      AX, #0x32         ;; 1 cycle
\   0000B0                  ; ------------------------------------ Block: 1 cycles
\               ??r_adc_interrupt_8:
\   0000B0 BF....             MOVW      N:ctl_parms_3_3, AX  ;; 1 cycle
\   0000B3                  ; ------------------------------------ Block: 1 cycles
 175            }
 178            TDR12 = ctl_parms_3_3.d[0];//D_local;
\               ??r_adc_interrupt_9:
\   0000B3 AF....             MOVW      AX, N:ctl_parms_3_3  ;; 1 cycle
\   0000B6 BE74               MOVW      0xFFF74, AX       ;; 1 cycle
 180            break;
\   0000B8 EF4C               BR        S:??r_adc_interrupt_3  ;; 3 cycles
\   0000BA                  ; ------------------------------------ Block: 5 cycles
 181
 182          case 6:
 183
 184            ctl_parms_1_7.e[1] = ctl_parms_1_7.e[0];
\               ??r_adc_interrupt_2:
\   0000BA AF....             MOVW      AX, N:ctl_parms_1_7+6  ;; 1 cycle
\   0000BD BF....             MOVW      N:ctl_parms_1_7+8, AX  ;; 1 cycle
 185
 186            t = set_value_1_7 - ADCR;

 200            ctl_parms_1_7.e[0] = (int16_t) (( (int32_t) t * 27034) >> 16) ;
\   0000C0 AF....             MOVW      AX, N:set_value_1_7  ;; 1 cycle
\   0000C3 261E               SUBW      AX, S:0xFFF1E     ;; 1 cycle
\   0000C5 329A69             MOVW      BC, #0x699A       ;; 1 cycle
\   0000C8 CEFB02             MULH                        ;; 2 cycles
\   0000CB 13                 MOVW      AX, BC            ;; 1 cycle
\   0000CC BF....             MOVW      N:ctl_parms_1_7+6, AX  ;; 1 cycle
 201
 202            MACRL = 0;
\   0000CF CBF00000           MOVW      0xFFFF0, #0x0     ;; 1 cycle
 203            MACRH = ctl_parms_1_7.d[0];
\   0000D3 AF....             MOVW      AX, N:ctl_parms_1_7  ;; 1 cycle
\   0000D6 BEF2               MOVW      0xFFFF2, AX       ;; 1 cycle
 204
 205            // MAC new error and k1
 206            __mach(K1_PI_12_4, ctl_parms_1_7.e[0]);
\   0000D8 AF....             MOVW      AX, N:K1_PI_12_4   ;; 1 cycle
\   0000DB CEFB06             MACH                        ;; 3 cycles
 207
 208            // MAC old error and k2
 209            __mach(K2_PI_12_4, ctl_parms_1_7.e[1]);
\   0000DE DB....             MOVW      BC, N:ctl_parms_1_7+8  ;; 1 cycle
```

```
\  0000E1 AF....              MOVW      AX, N:K2_PI_12_4   ;; 1 cycle
\  0000E4 CEFB06               MACH                        ;; 3 cycles
 210
 211            ctl_parms_1_7.d[0] = MACRH;
\  0000E7 AEF2                MOVW      AX, 0xFFFF2       ;; 1 cycle
 213
 216            if (ctl_parms_1_7.d[0] > D_MAX) {
\  0000E9 447F02               CMPW      AX, #0x27F        ;; 1 cycle
\  0000EC BF....              MOVW      N:ctl_parms_1_7, AX  ;; 1 cycle
\  0000EF DC05                 BC        ??r_adc_interrupt_10  ;; 4 cycles
\  0000F1                    ; ------------------------------------- Block: 28 cycles
 217              ctl_parms_1_7.d[0] = D_MAX;
\  0000F1 307E02               MOVW      AX, #0x27E        ;; 1 cycle
\  0000F4 EF08                 BR        S:??r_adc_interrupt_11  ;; 3 cycles
\  0000F6                    ; ------------------------------------- Block: 4 cycles
 218            } else if (ctl_parms_1_7.d[0] < 30) {
\                            ??r_adc_interrupt_10:
\  0000F6 441E00               CMPW      AX, #0x1E         ;; 1 cycle
\  0000F9 DE06                 BNC       ??r_adc_interrupt_12  ;; 4 cycles
\  0000FB                    ; ------------------------------------- Block: 5 cycles
 219              ctl_parms_1_7.d[0] = 30;
\  0000FB 301E00               MOVW      AX, #0x1E         ;; 1 cycle
\  0000FE                    ; ------------------------------------- Block: 1 cycles
\                            ??r_adc_interrupt_11:
\  0000FE BF....              MOVW      N:ctl_parms_1_7, AX  ;; 1 cycle
\  000101                    ; ------------------------------------- Block: 1 cycles
 220            }
 221
 231            TDR13 = ctl_parms_1_7.d[0];//D_local;
\                            ??r_adc_interrupt_12:
\  000101 AF....              MOVW      AX, N:ctl_parms_1_7  ;; 1 cycle
\  000104 BE76                 MOVW      0xFFF76, AX       ;; 1 cycle
 233            break;
\  000106                    ; ------------------------------------- Block: 2 cycles
 234
 235          case 7:
 236            break;
 237
 238          default:
 239            break;
 240          }
 241
 242          ADC_channel++;
\              ??r_adc_interrupt_3:
\  000106 A0....              INC       N:ADC_channel      ;; 2 cycles
 243          if(ADC_channel>7)
\  000109 8F....              MOV       A, N:ADC_channel   ;; 1 cycle
\  00010C 4C08                 CMP       A, #0x8           ;; 1 cycle
\  00010E 61C8                 SKC                         ;; 4 cycles
\  000110                    ; ------------------------------------- Block: 8 cycles
 244            ADC_channel = 4;
\  000110 CF......            MOV       N:ADC_channel, #0x4  ;; 1 cycle
\  000114                    ; ------------------------------------- Block: 1 cycles    247
 248           /* End user code. Do not edit comment generated here */
```

```
249         }
```

A.2 ARM – Cortex M0+ - Keil

A.2.1 C-code

```
ADC0_IRQHandler(){
        int32_t t;
        result= ADC0->R[0];
        switch(ADC_channel)
        {
        case 0:
                //Make it Software Trigger
                ADC0->SC2 = 0; //&~ADC_SC2_ADTRG_MASK;
                //ADC0SE4
                ADC0->SC1[0] = 0x44;


        t = set_volt_buck1 - (int32_t)lookup_MCU_volt[ (result>>(16-ADC_BITS)) - ADC_OFFSET
                ];

        ctl_parms_buck1.e[1] = ctl_parms_buck1.e[0];
        ctl_parms_buck1.e[0] = t; //set_volt_buck1 - (int32_t)t;
    //// d[n] = d[n-1] + K1*e[n] + K2*e[n-1];
                        ctl_parms_buck1.d[0] += K1_PI_MCU*ctl_parms_buck1.e[0];
                        ctl_parms_buck1.d[0] += K2_PI_MCU*ctl_parms_buck1.e[1];

    if (ctl_parms_buck1.d[0] < D_MIN_BUCK1) {
      ctl_parms_buck1.d[0] = D_MIN_BUCK1;
     }
     else if (ctl_parms_buck1.d[0] > D_MAX) {
      ctl_parms_buck1.d[0] = D_MAX;
     }
                        TPM0->CONTROLS[4].CnV = ctl_parms_buck1.d[0]>>16;
                        break;
        case 1:
                //ADC0SE0
                ADC0->SC1[0] = 0x43;
                t = set_volt_buck2 - (int32_t)result;
                ctl_parms_buck2.e[1] = ctl_parms_buck2.e[0];
                ctl_parms_buck2.e[0] = t; //set_volt_buck2 - (int32_t)result;
                ctl_parms_buck2.d[0] += K1_PI_28_4*ctl_parms_buck2.e[0];
                ctl_parms_buck2.d[0] += K2_PI_28_4*ctl_parms_buck2.e[1];

                if (ctl_parms_buck2.d[0] < D_MIN_BUCK2) {
                        ctl_parms_buck2.d[0] = D_MIN_BUCK2;
                }
                else if (ctl_parms_buck2.d[0] > D_MAX) {
                ctl_parms_buck2.d[0] = D_MAX;
                }
                TPM0->CONTROLS[2].CnV = ctl_parms_buck2.d[0]>>16;
                break;
```

```
        case 2:
                //Make it HW Trigger
                ADC0->SC2 = ADC_SC2_ADTRG_MASK;
                //ADC0SE0 and EI_ADC_INT
                ADC0->SC1[0] = 0x40;

                ctl_parms_buck3.e[1] = ctl_parms_buck3.e[0];
                ctl_parms_buck3.e[0] = set_volt_buck3 - (int32_t)result;
                ctl_parms_buck3.d[0] += K1_PI_28_4*ctl_parms_buck3.e[0];
                ctl_parms_buck3.d[0] += K2_PI_28_4*ctl_parms_buck3.e[1];

                 if (ctl_parms_buck3.d[0] > D_MAX) {
                          ctl_parms_buck3.d[0] = D_MAX;
                   }
                  else if (ctl_parms_buck3.d[0] < D_MIN_BUCK3) {
                          ctl_parms_buck3.d[0] = D_MIN_BUCK3;
                 }
                TPM0->CONTROLS[3].CnV = ctl_parms_buck3.d[0]>>16;
                break;

                default:
                        break;
        }
        ADC_channel++;
        if(ADC_channel>2)
                ADC_channel = 0;
}
```

A.2.2 Assembly

```
.text:000000A6 ; ========= S U B R O U T I N E ============================
.text:000000A6
.text:000000A6
.text:000000A6              EXPORT ADC0_IRQHandler
.text:000000A6                  ADC0_IRQHandler
.text:000000A6              PUSH        {R4-R7}
.text:000000A8              LDR         R0, =0x4003B000
.text:000000AA              LDR         R1, [R0,#0x10]
.text:000000AC              LDR         R2, =ADC_channel
.text:000000AE              LDR         R7, =0x3BE0000
.text:000000B0              STR         R1, [R2,#(result - 0x4D8)]
.text:000000B2              LDRB        R4, [R2]
.text:000000B4              CMP         R4, #0
.text:000000B6              BEQ         loc_CA
.text:000000B8              LDR         R3, =K1_PI_28_4
.text:000000BA              LDR         R2, =K2_PI_28_4
.text:000000BC              LDR         R3, [R3]
.text:000000BE              LDR         R2, [R2]
.text:000000C0              CMP         R4, #1
.text:000000C2              BEQ         loc_120
.text:000000C4              CMP         R4, #2
.text:000000C6              BNE         loc_198
.text:000000C8              B           loc_15C
.text:000000CA ; ---------------------------------------------------------------------------
```

```
.text:000000CA
.text:000000CA loc_CA                          ; CODE XREF: ADC0_IRQHandler+10j
.text:000000CA              MOVS         R2, #0
.text:000000CC              STR          R2, [R0,#0x20]
.text:000000CE              MOVS         R2, #0x44 ; 'D'
.text:000000D0              STR          R2, [R0]
.text:000000D2              LSRS         R0, R1, #8
.text:000000D4              LDR          R1, =lookup_MCU_volt
.text:000000D6              LSLS         R0, R0, #2
.text:000000D8              ADDS         R0, R0, R1
.text:000000DA              SUBS         R0, #0xFF
.text:000000DC              LDR          R1, =set_volt_buck1
.text:000000DE              SUBS         R0, #0x81 ; 'ü'
.text:000000E0              LDR          R0, [R0,#0x50]
.text:000000E2              LDR          R1, [R1]
.text:000000E4              LDR          R5, =K1_PI_MCU
.text:000000E6              SUBS         R0, R1, R0
.text:000000E8              LDR          R1, =ctl_parms_buck1
.text:000000EA              LDR          R2, [R1,#0xC]
.text:000000EC              STR          R2, [R1,#0x10]
.text:000000EE              STR          R0, [R1,#0xC]
.text:000000F0              LDR          R5, [R5]
.text:000000F2              LDR          R3, [R1]
.text:000000F4              MULS         R0, R5
.text:000000F6              ADDS         R0, R3, R0
.text:000000F8              LDR          R3, =K2_PI_MCU
.text:000000FA              LDR          R3, [R3]
.text:000000FC              MULS         R2, R3
.text:000000FE              ADDS         R0, R0, R2
.text:00000100              MOVS         R2, #0x4C0000
.text:00000104              STR          R0, [R1]
.text:00000106              CMP          R0, R2
.text:00000108              BCS          loc_10E
.text:0000010A              STR          R2, [R1]
.text:0000010C              B            loc_116
.text:0000010E ; ---------------------------------------------------------------------------
.text:0000010E
.text:0000010E loc_10E                         ; CODE XREF: ADC0_IRQHandler+62j
.text:0000010E              CMP          R0, R7
.text:00000110              BHI          loc_114
.text:00000112              MOV          R7, R0
.text:00000114
.text:00000114 loc_114                         ; CODE XREF: ADC0_IRQHandler+6Aj
.text:00000114              STR          R7, [R1]
.text:00000116
.text:00000116 loc_116                         ; CODE XREF: ADC0_IRQHandler+66j
.text:00000116              LDR          R0, [R1]
.text:00000118              LSRS         R1, R0, #0x10
.text:0000011A              LDR          R0, =0x40038000
.text:0000011C              STR          R1, [R0,#0x30]
.text:0000011E              B            loc_198
.text:00000120 ; ---------------------------------------------------------------------------
.text:00000120
.text:00000120 loc_120                         ; CODE XREF: ADC0_IRQHandler+1Cj
```

```
.text:00000120             MOVS         R5, #0x43 ; 'C'
.text:00000122             STR          R5, [R0]
.text:00000124             LDR          R0, =set_volt_buck2
.text:00000126             LDR          R0, [R0]
.text:00000128             SUBS         R1, R0, R1
.text:0000012A             LDR          R0, =ctl_parms_buck2
.text:0000012C             LDR          R5, [R0,#0xC]
.text:0000012E             STR          R5, [R0,#0x10]
.text:00000130             STR          R1, [R0,#0xC]
.text:00000132             LDR          R6, [R0]
.text:00000134             MULS         R5, R2
.text:00000136             MULS         R1, R3
.text:00000138             ADDS         R1, R6, R1
.text:0000013A             ADDS         R1, R1, R5
.text:0000013C             MOVS         R2, #0xBF0000
.text:00000140             STR          R1, [R0]
.text:00000142             CMP          R1, R2
.text:00000144             BCS          loc_14A
.text:00000146             STR          R2, [R0]
.text:00000148             B            loc_152
.text:0000014A ; ---------------------------------------------------------------------------
.text:0000014A
.text:0000014A loc_14A                          ; CODE XREF: ADC0_IRQHandler+9Ej
.text:0000014A             CMP          R1, R7
.text:0000014C             BHI          loc_150
.text:0000014E             MOV          R7, R1
.text:00000150
.text:00000150 loc_150                          ; CODE XREF: ADC0_IRQHandler+A6j
.text:00000150             STR          R7, [R0]
.text:00000152
.text:00000152 loc_152                          ; CODE XREF: ADC0_IRQHandler+A2j
.text:00000152             LDR          R0, [R0]
.text:00000154             LSRS         R1, R0, #0x10
.text:00000156             LDR          R0, =0x40038000
.text:00000158             STR          R1, [R0,#0x20]
.text:0000015A             B            loc_198
.text:0000015C ; ---------------------------------------------------------------------------
.text:0000015C
.text:0000015C loc_15C                          ; CODE XREF: ADC0_IRQHandler+22j
.text:0000015C             MOVS         R5, #0x40 ; '@'
.text:0000015E             STR          R5, [R0,#0x20]
.text:00000160             STR          R5, [R0]
.text:00000162             LDR          R0, =ctl_parms_buck3
.text:00000164             LDR          R6, =set_volt_buck3
.text:00000166             LDR          R5, [R0,#0xC]
.text:00000168             STR          R5, [R0,#0x10]
.text:0000016A             LDR          R6, [R6]
.text:0000016C             MULS         R5, R2
.text:0000016E             SUBS         R1, R6, R1
.text:00000170             STR          R1, [R0,#0xC]
.text:00000172             LDR          R6, [R0]
.text:00000174             MULS         R1, R3
.text:00000176             ADDS         R1, R6, R1
.text:00000178             ADDS         R1, R1, R5
```

```
.text:0000017A              STR          R1, [R0]
.text:0000017C              CMP          R1, R7
.text:0000017E              BLS          loc_184
.text:00000180              STR          R7, [R0]
.text:00000182              B            loc_190
.text:00000184 ; ---------------------------------------------------------------------------
.text:00000184
.text:00000184 loc_184                          ; CODE XREF: ADC0_IRQHandler+D8j
.text:00000184              MOVS          R2, #0x130000
.text:00000188              CMP          R1, R2
.text:0000018A              BHI          loc_18E
.text:0000018C              MOV          R1, R2
.text:0000018E
.text:0000018E loc_18E                          ; CODE XREF: ADC0_IRQHandler+E4j
.text:0000018E              STR          R1, [R0]
.text:00000190
.text:00000190 loc_190                          ; CODE XREF: ADC0_IRQHandler+DCj
.text:00000190              LDR          R0, [R0]
.text:00000192              LSRS         R1, R0, #0x10
.text:00000194              LDR          R0, =0x40038000
.text:00000196              STR          R1, [R0,#0x28]
.text:00000198
.text:00000198 loc_198                          ; CODE XREF: ADC0_IRQHandler+20j
.text:00000198                                  ; ADC0_IRQHandler+78j ...
.text:00000198              ADDS          R4, R4, #1
.text:0000019A              LDR          R0, =ADC_channel
.text:0000019C              UXTB         R1, R4
.text:0000019E              STRB         R1, [R0]
.text:000001A0              CMP          R1, #2
.text:000001A2              BLS          loc_1A8
.text:000001A4              MOVS          R1, #0
.text:000001A6              STRB         R1, [R0]
.text:000001A8
.text:000001A8 loc_1A8                          ; CODE XREF: ADC0_IRQHandler+FCj
.text:000001A8              POP          {R4-R7}
.text:000001AA              BX           LR
.text:000001AA ; End of function ADC0_IRQHandler
```
