

ABSTRACT

LI, SI. Reducing CPU Utilization for Software Implemented SMPS Controllers. (Under the direction of Dr. Alexander Dean).

There are many studies seeking reduction of CPU utilization and improvement of power consumption efficiency. For CPU utilization reduction, one effective solution is to use event-based control to avoid unnecessary CPU operations. For power efficiency, a switch mode power supply (SMPS) is widely used because it can efficiently convert an input voltage to a different output voltage with low power loss.

In this thesis, a software SMPS controller is implemented for a boost converter to regulate the output voltage. At the same time, we use event-based control and code optimization methods to effectively reduce the CPU utilization for our voltage regulating task. In the experimental section, we test the controlled boost converter with three kinds of loads with different characteristics and demonstrate correct operation with much lower CPU utilization.

© Copyright 2014 by Si Li

All Rights Reserved

Reducing CPU Utilization for Software Implemented SMPS Controllers

by
Si Li

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the degree of
Master of Science

Electrical Engineering

Raleigh, North Carolina

2014

APPROVED BY:

Dr. Alexander Dean
Committee Chair

Dr. Subhashish Bhattacharya

Dr. James Tuck

DEDICATION

To everyone who loves me in my life

Especially to my parents,

Mr. Xiaobo Li and Mrs. Yazhi Zhou,

Thank you for your unceasing love and support to me.

BIOGRAPHY

Si Li was born on September 22, 1989 in Changchun, China. She had her undergraduate study in Tongji University, Shanghai and received the degree of Bachelor in Electrical Engineering in 2012. In the third year of undergraduate study, she began internship in United Automotive Electronic Systems Co., Ltd. (UAES) and worked there for 8 months. After graduation from Tongji University, she continued to pursue master degree in Electrical Engineering in North Carolina State University since 2012. In the summer of 2013, she joined Dr. Dean's research group and was under his direction for thesis. In the thesis, her concentration was embedded software implementation and power electronics. In the future career, she will continue to focus herself on embedded software engineering.

ACKNOWLEDGMENTS

I would like to first show my respect and appreciation to my advisors. I would like to thank Dr. Alexander Dean for providing me the chance to work with him and for his valuable guidance. From him, I gained a lot of research and practical experience. I would like to thank Dr. Subhashish Bhattacharya for his help in power electronics. I would like to thank Dr. James Tuck for his advice in code optimization. Also, I would like to thank my research groupers. All of you have been helping me so much through my completion of thesis. Especially I would like to thank Avik Juneja, Shikhar Singh, Udai Muhammed and Zhi Qu. Thank you all for the unforgettable time working together.

TABLE OF CONTENTS

LIST OF FIGURES	viii
Chapter 1 INTRODUCTION	1
1.1 Motivation.....	1
1.2 Overview of the Research Contribution	1
1.3 Outline of the Thesis.....	2
Chapter 2 RELATED WORK	3
2.1 Boost Converter	3
2.2 PI Control.....	4
2.3 Event-Based Control.....	4
2.3.1 Ārz : Event-Based PID Controller [3].....	4
2.3.2 Durand: An Event-Based PID Controller With Low Computational Cost [4]	6
2.3.3 Durand: Further Results on Event-Based PID Controller [5].....	6
2.3.4 Xing: Some Improvements on Event-Based PID Controllers [6].....	7
Chapter 3 RESEARCH CONTRIBUTION	8
3.1 Boost Converter	8
3.1.1 Design	8
3.1.2 Operating Principles	9
3.2 Boost Converter Analysis	10
3.2.1 DC model analysis [7]	10
3.2.2 AC Small Signal Model Analysis [7]	12
3.3 PI Control Analysis and Implementation.....	13
3.3.1 Theoretical Method [7]	13

3.3.2	MATLAB Implementation	15
3.3.3	PI Control Software Implementation	16
3.4	Event-Based (Dead-Band) Control Timing Analysis and Implementation	22
3.4.1	Event-Based Control Timing Analysis	22
3.4.2	Event-Based Control Implementation	29
3.4.3	Two Reference Voltages Implementation	30
3.5	CPU Usage Calculation Software Design	32
Chapter 4	EXPERIMENTAL EVALUATION AND SIMULATION	35
4.1	Simulation	35
4.2	Time-Triggered Control and Dead-Band Control	36
4.2.1	Test Load (Constant)	37
4.2.2	Test Load (Periodic)	40
4.2.3	Wi-Fi Module as Load	45
4.3	Analysis	52
4.3.1	Analysis for Constant Load	52
4.3.2	Analysis for Periodic Load	52
4.3.3	Analysis for Wi-Fi Module	53
Chapter 5	CONCLUSION AND FUTURE WORK	55
5.1	Conclusion	55
5.2	Future work	56
	REFERENCES	57
	APPENDICES	58

Appendix A. Waveforms of Dead-Band Control with One Reference Voltage for the Periodically Changed Load.....	59
Appendix B. Waveforms of Dead-Band Control with Two Reference Voltages for the Periodically Changed Load.....	62
Appendix C. Waveforms of Dead-Band Control with One Reference Voltages for the Constant Load	65
Appendix D. Waveforms of Dead-Band Control with Two Reference Voltages for the Constant Load	67
Appendix E. Object Code	69

LIST OF FIGURES

Figure 2-1 Boost converter circuit	3
Figure 2-2 Event-based control system [3].....	5
Figure 3-1 Hand-built boost converter circuit	8
Figure 3-2 Boost converter circuit diagram	9
Figure 3-3 Equivalent circuit for $0 < t \leq DT$	11
Figure 3-4 Equivalent circuit for $T < t \leq T$	11
Figure 3-5 Bode Plot before compensation	14
Figure 3-6 Bode Plot after compensation	15
Figure 3-7 Sequence diagram for ADC operation and control loop.....	16
Figure 3-8 Object code control flow graph.....	20
Figure 3-9 Blue trace shows time ADC ISR is active and executing control loop.....	21
Figure 3-10 Transient response in time-triggered control mode	23
Figure 3-11 Transient response in dead-band control in case $V_{ini} - V_{refl} > \Delta V_{O,R}$	25
Figure 3-12 Transient response in dead-band control in case $V_{ini} - V_{refl} \leq \Delta V_{O,R}$	26
Figure 3-13 Transient response in dead-band control in never-fall-out case.....	27
Figure 3-14 Transient response of dead-band control in real situation, first fall-out point after ISR	28
Figure 3-15 Transient response of dead-band control in real situation, first fall-out point before ISR	29
Figure 3-16 Indication graph of three A/D conversion modes	30
Figure 3-17 Difference between one-reference-voltage mode and two-reference-voltage mode.....	31
Figure 3-18 Sequence diagram for CPU usage calculation operation	32

Figure 4-1 Simulation circuit graph with PLECS.....	35
Figure 4-2 Simulation result	36
Figure 4-3 Constant load in time-triggered control: ISR indication and output.....	37
Figure 4-4 Constant load in dead-band control with one-reference-voltage mode in bands: 3.397V to 3.185V.....	38
Figure 4-5 Constant load in dead-band control with two-reference-voltage mode in bands: 3.397V to 3.185V.....	39
Figure 4-6 Periodical load in time-triggered control: transient response	41
Figure 4-7 Periodical load in time-triggered control: ISR indication graph with output.....	41
Figure 4-8 Periodical load in dead-band control with one-reference-voltage mode in bands: 3.397V to 3.185V.....	42
Figure 4-9 Periodical load in dead-band control with one-reference-voltage mode in bands: 3.397V to 3.185V.....	44
Figure 4-10 Wi-Fi module: initial state and steady state	46
Figure 4-11 Wi-Fi module: initial state	46
Figure 4-12 Wi-Fi module: steady state.....	47
Figure 4-13 Wi-Fi module: one-reference-voltage mode with bands 3.326V to 3.255V.....	47
Figure 4-14 Wi-Fi module: two-reference-voltage mode with bands 3.326V to 3.255V.....	48
Figure 4-15 Wi-Fi module: one-reference-voltage mode with bands 3.350V to 3.232V.....	48
Figure 4-16 Wi-Fi module: two-reference-voltage mode with bands 3.350V to 3.232V.....	49
Figure 4-17 Wi-Fi module: one-reference-voltage mode with bands 3.373V to 3.208V.....	50
Figure 4-18 Wi-Fi module: two-reference-voltage mode with bands 3.373V to 3.208V.....	50
Figure 4-19 Wi-Fi module: one-reference-voltage mode with bands 3.397V to 3.185V.....	51
Figure 4-20 Wi-Fi module: two-reference-voltage mode with bands 3.397V to 3.185V.....	51
Figure 4-21 Analysis: CPU usage of constant load	52

Figure 4-22 Analysis: CPU usage of periodical load..... 53

Figure 4-23 Analysis: CPU usage of Wi-Fi module 54

Figure 5-1 Periodical load in dead-band control: one-reference-voltage, dead bands 3.326V to 3.255V, zoom-out version 59

Figure 5-2 Periodical load in dead-band control: one-reference-voltage, dead bands 3.326V to 3.255V, zoom-in version 59

Figure 5-3 Periodical load in dead-band control: one-reference-voltage, dead bands 3.350V to 3.232V, zoom-out version 60

Figure 5-4 Periodical load in dead-band control: one-reference-voltage, dead bands 3.350V to 3.232V, zoom-in version 60

Figure 5-5 Periodical load in dead-band control: one-reference-voltage, dead bands 3.373V to 3.208V, zoom-out version 61

Figure 5-6 Periodical load in dead-band control: one-reference-voltage, dead bands 3.373V to 3.208V, zoom-in version 61

Figure 5-7 Periodical load in dead-band control: two-reference-voltage, dead bands 3.326V to 3.255V, zoom-out version 62

Figure 5-8 Periodical load in dead-band control: two-reference-voltage, dead bands 3.326V to 3.255V, zoom-in version 62

Figure 5-9 Periodical load in dead-band control: two-reference-voltage, dead bands 3.350V to 3.232V, zoom-out version 63

Figure 5-10 Periodical load in dead-band control: two-reference-voltage, dead bands 3.350V to 3.232V, zoom-in version 63

Figure 5-11 Periodical load in dead-band control: two-reference-voltage, dead bands 3.373V to 3.208V, zoom-out version 64

Figure 5-12 Periodical load in dead-band control: two-reference-voltage, dead bands 3.373V to 3.208V, zoom-in version 64

Figure 5-13 Constant load in dead-band control: one-reference-voltage, dead bands 3.373V to 3.208V..... 65

Figure 5-14 Constant load in dead-band control: one-reference-voltage, dead bands 3.350V to 3.232V.....	65
Figure 5-15 Constant load in dead-band control: one-reference-voltage, dead bands 3.326V to 3.255V.....	66
Figure 5-16 Constant load in dead-band control: two-reference-voltage, dead bands 3.373V to 3.208V.....	67
Figure 5-17 Constant load in dead-band control: two-reference-voltage, dead bands 3.350V to 3.232V.....	67
Figure 5-18 Constant load in dead-band control: two-reference-voltage, dead bands 3.326V to 3.255V.....	68

Chapter 1 INTRODUCTION

1.1 Motivation

This thesis project focuses on event-based PI control and feed forward control of a switching power converter which is controlled by an RL78/G14 16-bit single-chip microcontroller. Switching power converters are used to convert power efficiently between voltages; this enables power optimization by operating circuits at lower voltages. A control system is used to improve the converter's output accuracy in the face of changes in output load and input voltage. This control system can be implemented in dedicated hardware (e.g. digital logic or analog hardware) or in software on the MCU. The purpose of the project is to reduce the CPU usage of a software implementation by reducing the number of times the control system must execute.

1.2 Overview of the Research Contribution

My first contribution is in the development and analysis of the boost converter. I studied power electronics and moved on to the boost converter we use in the project. I began with hardware by studying and changing a boost converter circuit (developed by a previous graduate student for driving high-brightness LEDs, which require current regulation) and then made changes to fit this project (voltage regulation) better. Second, to achieve a constant output voltage, the system needed a reliable control system. I analyzed the boost converter circuit, built the DC and AC model with the loss elements and derived the transfer function of the output voltage against transistor duty cycle. Next I took advantage of the sisotool function in MATLAB to generate the Bode plot, and tuned it to get a stable system.

My second and main contribution is an analysis of methods to reduce CPU usage. The control loop is very CPU-intensive. To reduce CPU usage, the system should use less time on the control loop. The solution is event-based control. Rather than trigger the control loop with a fixed frequency (time-triggered), we trigger when there is excessive error between the

actual output voltage and the desired voltage. This control method is called dead-band control.

The third contribution is how to implement dead-band control, making use of the existing peripherals in the MCU. In RL78/G14, the A/D converter has two modes of interrupts. The first mode causes an interrupt happens every time the A/D conversion finishes. The second causes an interrupt when A/D conversion result is out of a specified range. The second mode meets the dead-band control requirement, so we use it in this project.

1.3 Outline of the Thesis

The rest of the thesis has the following structure: In Chapter 2, I will describe some related research on event-based control. In Chapter 3, I will introduce my contribution in this thesis project. In Chapter 4, related experiments, simulations and application are presented. In Chapter 5, the conclusion and future work will be shown.

Chapter 2 RELATED WORK

In this chapter, I will introduce the boost converter and previous research in event-based control. The research of Årz  is the basis of the other two.

2.1 Boost Converter

The boost converter is a basic class of switch mode power supply (SMPS) (Erickson & Maksimovic, 2001). The output voltage is higher than the input voltage. The boost converter consists of at least two semiconductors, which are diode and transistor. It also contains at least one power storage element, which is inductor and capacitor. The ideal asynchronous boost converter has the circuit shown in Figure 2-1.

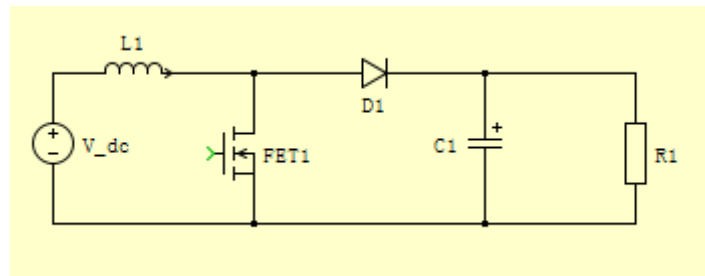


Figure 2-1 Boost converter circuit

The transistor is driven by PWM signal, whose duty cycle is D . The output and input voltage have the idealized relationship [1] [2]:

$$V_{\text{out}} = \frac{V_{\text{in}}}{1 - D}$$

This relationship ignores effects from changes in load resistance, parasitic circuit elements (resistances, inductances, capacitances) in the components and voltage ripple from switching.

2.2 PI Control

Because the idealized model above ignores various realities, a closed-loop control system is typically used to control the duty cycle in order to regulate the output voltage (minimizing the error from the set point).

PI and PID controllers are often used for feedback control. PI stands for proportional and integral, while PID adds a derivative term. The basic PID control has the format as follows [3] [4]:

$$G(s) = K_p + K_i \frac{1}{s} + K_d s$$

The proportional term K_p provides a control action proportional to the error. Increasing the proportional parameter will decrease the rising time and the settling time, but increase the overshoot and instability.

The integral term K_i will reduce the steady-state error. Increasing the integral parameter will largely decrease the steady-state error, slightly decrease the rising time, but increase the overshoot and instability.

The derivative term K_d provides a control action proportional to the rate of change of the error. This term also decreases the rising time and the settling time, but increase the overshoot and instability.

I used the PI controller because it is adequate for the boost converter control and less computationally intensive than the PID controller.

2.3 Event-Based Control

2.3.1 Árż éń: Event-Based PID Controller [3]

Árż éń's event-based control system is designed as a client-server structure, as shown in Figure 2-2. The event detector works as the client, which is triggered periodically in normal

frequency, the time period is h_{nom} . The PID controller works as the server, which is an event-triggered mechanism. It is only triggered when the output of Event Detector meets the event trigger requirements.

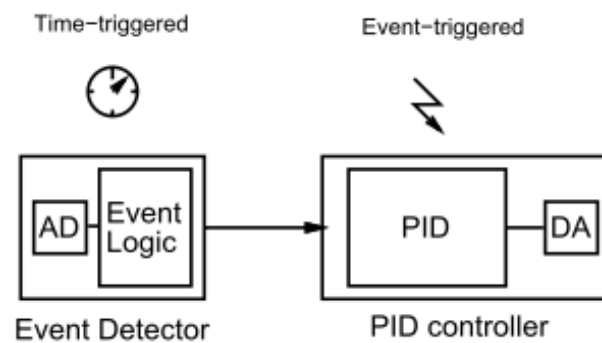


Figure 2-2 Event-based control system [3]

The PID controller is based on classic PID control theory and has the same structure as the classic PID controller. It consists of proportional part, integral part and derivative part. The difference between the conventional PID controller and the proposed one is that the PID coefficients have to be recalculated every time the control implements. The reason is that discretization needs frequency information, the event-based control has a sampling frequency which is not fixed and is irregularly changed.

The sampling frequency changes according to the following rule: at each normal sampling period, control sampling period h_{act} changes to the summation of current h_{act} and normal sampling period h_{nom} . If h_{act} becomes larger than a limit h_{max} or event-based control is triggered, reset h_{act} to zero.

2.3.2 Durand: An Event-Based PID Controller With Low Computational Cost [4]

Durand's research builds on Årz é's and is focused on lowering computational requirements. They achieve the goal by setting a minimum sampling interval condition. In Årz é's research, if the triggering event doesn't occur for several normal sampling periods, the control will execute every sampling period. Although this will achieve the desired result, it costs many computations. If control time demand is not very strict, the minimum sampling period could be defined, and execution sampling period should be longer than the minimum sampling period.

After the transient, although the output error is less than the triggering error limit, it still has some oscillations due to lack of control. Durand's solution is to add some extra samples right after the transient, to make the error smaller and the system more stable.

2.3.3 Durand: Further Results on Event-Based PID Controller [5]

In this research, a hybrid algorithm is introduced to save control samples. It focuses on integral part of PID control. In Årz é's research, when triggering error limit increases, the output oscillates after the transient response. The result comes from a problem in integral part. In Årz é's research, the forward difference approximation is implemented to calculate the integral part. This method is a good approach in time-triggered PID control, but it is not a proper choice in event-based control. In Durand's research, it changes the approach into backward difference approximation. Based on this change, it combines two algorithms to ensure the reliability of the integral part. The first algorithm is to saturate product of $h(tk)$ and $e(tk)$ when $h_{act} > h_{max}$. When implementing this algorithm along, the output has almost the same overshoot as in time-triggered PID control. The second algorithm is to add a forgetting factor of the sampling period. This will result in that the control sampling period approaches normal sampling period during transient response. In steady state intervals, the control sampling period will be exponentially decreased, which will reduce the impact to the control signal. This algorithm will cause larger overshoot than conventional discrete PID control, but will have better track with the conventional PID control in steady state intervals.

After combination of the two algorithms, the output has fewer oscillations, and has modest and reasonable overshoot.

2.3.4 Xing: Some Improvements on Event-Based PID Controllers [6]

This research is based on Årz é n's and Durand's researches and has a few improvements. They apply trapezium difference approximation other than forward difference approximation in Årz é n's research and backward difference approximation in Durand's research. For the steady-state sampling condition, besides the condition $e - e_{old} \leq e_{lim}$, they use an additional condition $e \geq e_{lim}$. When the system satisfies the first condition, if it also satisfies the second one, the system will trigger the controller to calculate the new control signal. This method will keep the static error converge to e_{max} . In addition to the improvement above, they also judge the step of sampling sensor k . If k is dividable by an unchanged integer b , the controller will be triggered. By experiment results, they also propose that increasing e_{lim} properly will reduce a lot of sampling numbers. After applying those improvements, the system could get an acceptable performance with low sampling rates.

Chapter 3 RESEARCH CONTRIBUTION

My research is in the software design of a switch mode power supply system built on a boost converter. The contributions include event-based control timing analysis and implementation; boost converter analysis; PI control; analysis of using one or two reference voltages, implementation and CPU usage calculation software design.

3.1 Boost Converter

3.1.1 Design

Figure 3-1 and Figure 3-2 shows the boost converter we use in this thesis. In which the hand-build hardware and the circuit schematics are presented respectively.

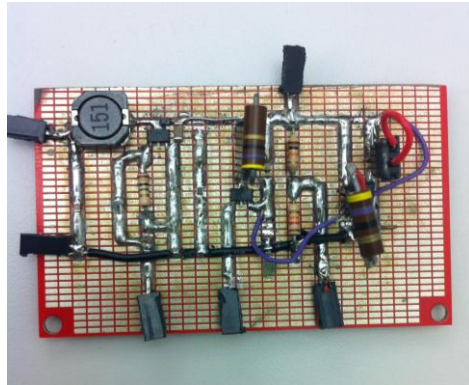


Figure 3-1 Hand-built boost converter circuit

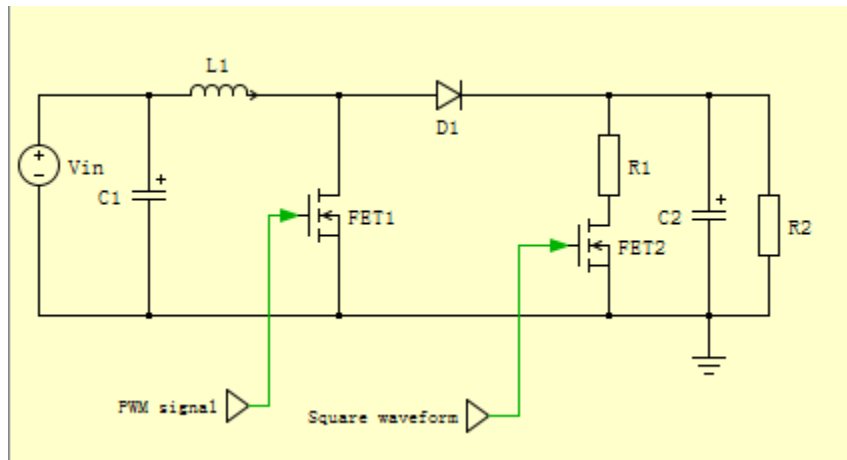


Figure 3-2 Boost converter circuit diagram

3.1.2 Operating Principles

This is an asynchronous boost converter with load switching part. The output voltage is related with MOSFET switching duty cycle and input voltage. Recall that ideal relationship between the output voltage and the input voltage and the boost converter switch's duty cycle D is [1] [2]:

$$V_{\text{out}} = \frac{V_{\text{in}}}{1 - D}$$

The input capacitor $C1$ is used to reduce the overall noise to ensure a better performance. Output capacitor $C2$ reduces the output voltage ripple.

The boost converter MOSFET FET1 is driven by a pulse-width-modulated signal generated by channels 0 and 1 of a Timer Array Unit. The duty cycle is adjusted by implementing PI control on slave channel register value.

Diode $D1$ switches on automatically when FET1 is off, enabling the current from inductor $L1$ to flow to the output $R2$ and charge $C2$.

The MOSFET FET2 on the board is used to change the load periodically, introducing load transients which are used to evaluate the system's ability to minimize the output voltage error. FET2 is driven by channel 2 of Timer Array Unit, which generates a square wave. When the output voltage level is high, the additional load resistor R1 is connected in parallel with the original load R2. When the output voltage level is low, the additional load R1 is disconnected from the circuit.

3.2 Boost Converter Analysis

In this circuit, several loss elements have to be considered.

- a. Inductor Copper Loss, Series Resistor R_L
- b. Capacitor loss R_C
- c. Semiconductor on-resistance R_{on}
- d. Forward Voltage Drop V_D

The circuit containing all the loss elements above is shown below. We consider the two states of the circuit: when FET1 is on and when it is off.

3.2.1 DC model analysis [7]

(1) $0 < t \leq DT$

In this case, the equivalent circuit is shown in Figure 3-3.

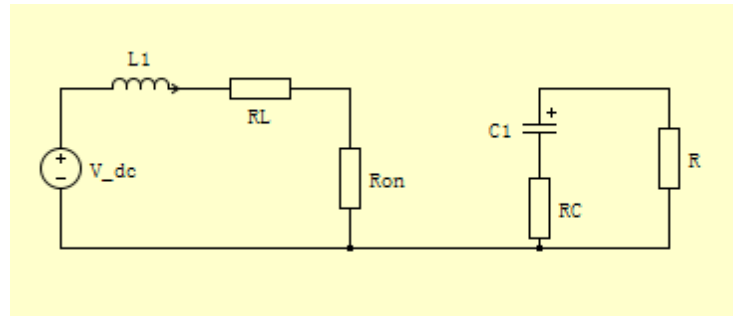


Figure 3-3 Equivalent circuit for $0 < t \leq DT$

$$v_L(t) = V_{in} - iR_L - iR_{on}$$

$$i_C(t) = -\frac{V_o}{R_o}$$

(2) $DT < t \leq T$

In this case, the equivalent circuit is shown in Figure 3-4.

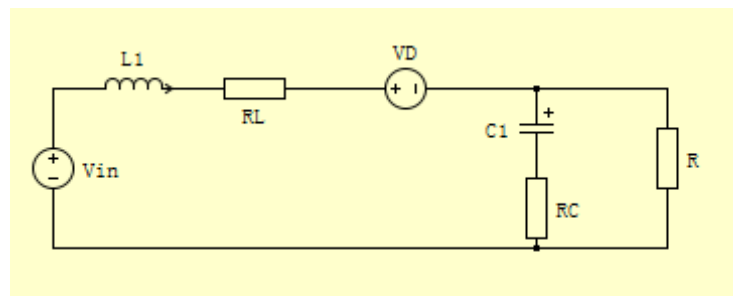


Figure 3-4 Equivalent circuit for $T < t \leq T$

$$v_L = V_{in} + V_D - iR_L - V_o$$

$$i_C = i - \frac{V_o}{R_o}$$

3.2.2 AC Small Signal Model Analysis [7]

Equalizing the DC components gives:

$$D(V_{in} - IR_L - IR_{on}) + (1 - D)(V_{in} + V_D - IR_L - V) = 0$$

$$-D\frac{V_o}{R_o} + (1 - D)(I - \frac{V_o}{R_o}) = 0$$

From the equations above, considering the loss elements, the boost converter DC model is:

$$V_o \frac{(1 - D) + (R_L + DR_{on})}{(1 - D)R_o} = V_{in} + (1 - D)V_D$$

The AC equivalent circuit model is

$$L \frac{d \langle i(t) \rangle_{T_s}}{dt} = d(t) (\langle v_{in}(t) \rangle_{T_s} - \langle i(t) \rangle_{T_s} R_{on}) + d'(t) (\langle v_{in}(t) \rangle_{T_s} - \langle i(t) \rangle_{T_s} R_L + V_D - \langle v_o(t) \rangle_{T_s})$$

$$C \left(\frac{R_o - R_C}{R_o} \right) \frac{d \langle v_o(t) \rangle_{T_s}}{dt} = -d(t) \frac{\langle v(t) \rangle_{T_s}}{R_o} + d'(t) (\langle i(t) \rangle_{T_s} - \frac{\langle v_o(t) \rangle_{T_s}}{R_o})$$

The equalized elements have these relationships with the DC and AC parts:

$$\langle i(t) \rangle_{T_s} = I + \hat{i};$$

$$\langle v_{in}(t) \rangle_{T_s} = V_{in} + \widehat{v}_{in};$$

$$\langle v_o(t) \rangle_{T_s} = V_o + \widehat{v}_o;$$

$$d(t) = D + \tilde{d};$$

$$d'(t) = D' - \tilde{d};$$

Next we convert the small-signal AC model from time domain to s domain

$$Ls\hat{i}(s) = \hat{d}(s)(V - V_D - IR_{on}) - \hat{i}(s)(R_L + DR_{on}) + \hat{v}_{in}(s) + \hat{v}_o(s)D'$$

$$Cs \frac{(R_o - R_c)}{R_o} \hat{v}_o(s) = -\frac{\hat{v}_o(s)}{R_o} - I\hat{d}(s) + D'\hat{i}(s)$$

Next, we set \hat{v}_{in} to 0

$$\frac{\hat{v}_o(s)}{\hat{d}(s)} = \frac{-\frac{ILs}{D'} + V - V_D - IR_{on} - I(R_L + DR_{on})}{\frac{LCs^2(R_o - R_c)}{D'R_o} + \frac{(C(R_o - R_c)(R_L + DR_{on}) + L)s}{D'R_o} + \frac{R_L + DR_{on}}{D'R_o} + D'}$$

in which,

$$V = \frac{V_{in} + (1 - D)V_D}{(1 - D) + \frac{R_L + DR_{on}}{(1 - D)R_o}}$$

$$I = \frac{V_{in} + (1 - D)V_D}{(1 - D)^2R_o + R_L + DR_{on}}$$

3.3 PI Control Analysis and Implementation

We use this transfer function to design a compensator (a control system) using both theoretical method and also with the help of MATLAB and its tool box.

3.3.1 Theoretical Method [7]

In this section, we first use MATLAB function `tf()` to generate the continuous control-to-output transfer function $Gvd(s)$, and take advantage of the function `sisotool()` to generate the Bode Plot before compensation. The Bode Plot before compensation is in Figure 3-5.

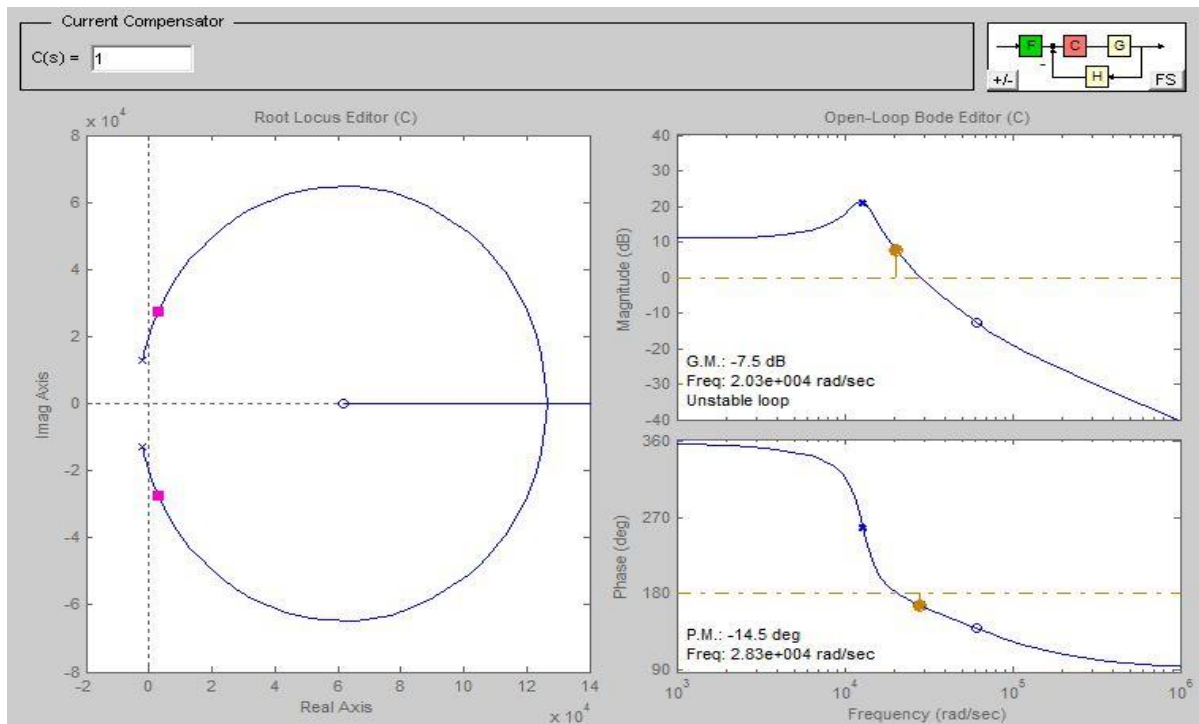


Figure 3-5 Bode Plot before compensation

According to the PI compensator design method, the compensator should be in the format of:

$$G_c = G_{c\infty} \left(1 + \frac{\omega_L}{s}\right)$$

Because the PWM transistor switching frequency is 50kHz, we need to set the crossover frequency $f_c = f_{sw}/10 = 5\text{kHz} = 31\text{k rad/sec}$. So we set the compensator gain $G_{c\infty}$ to 0.2 in order to compensate the magnitude at 31k rad/sec to 0 dB. Second, we should choose a value for ω_L which is sufficiently small. In this case, we set $\omega_L = 100$. The compensated Bode Plot is shown in Figure 3-6.

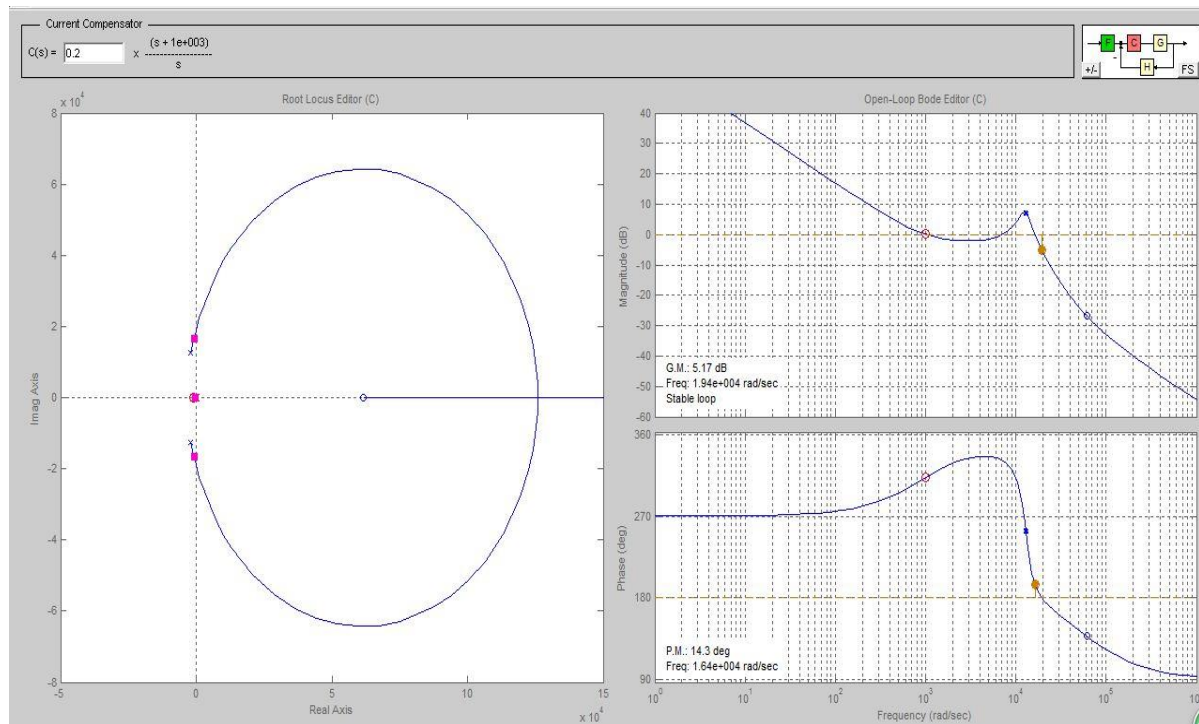


Figure 3-6 Bode Plot after compensation

3.3.2 MATLAB Implementation

We first find the discrete transfer function of output voltage to duty cycle by using MATLAB's `c2d()` function and this boost converter analysis. We then use the discrete transfer function as the input of the `sisotool()` function, and tune the PI control compensator parameters according to the Bode Plot generated. The PI compensator in MATLAB has the format:

$$C(z) = \frac{K1 \left(z + \frac{K2}{K1} \right)}{z - 1}$$

We get the following result:

$$K1 = 0.024$$

$$K2 = -0.02$$

3.3.3 PI Control Software Implementation

We implemented the control loop in C code for the RL78 MCU.

3.3.3.1 Timing Overview

The timing sequence of the software control implementation is shown in Figure 3-7.

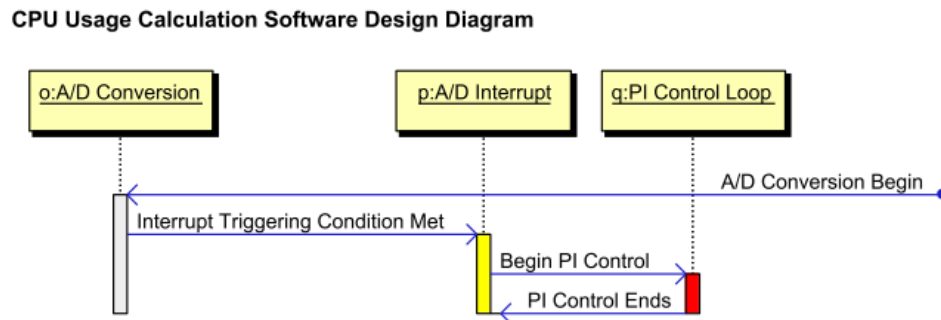


Figure 3-7 Sequence diagram for ADC operation and control loop

When the ADC result meets the interrupt triggering condition (to be described below), the A/D Interrupt is generated and the A/D interrupt service routine executes, which performs the PI Control Loop.

3.3.3.2 PI Control Loop Source Code

The PI control loop source code is shown below.

```

gADC_Result = ADCR; |
Voltage_FX = (gADC_Result>>8) + (gADC_Result>>9) + (gADC_Result>>15);

error_p=error_10_6;

/*****boost control part*****/

Mult_R = ((unsigned long)(TDR01_temp)) << 4; //2_14 to 12_20

//first calculate error_p and K2_boost
if(error_flag == 1)
{
    Mult_R += K2_boost * error_p;
}
else
    Mult_R -= K2_boost * error_p;

```

```

if(One_Vref){
    //then calculate error_10_6 and K1 boost
    if(Vref <= Voltage_FX)
    {
        error_flag = 1;
        error_10_6 = Voltage_FX - Vref;
        Mult_R -= K1_boost * error_10_6;
    }
    else if(Vref >= Voltage_FX)
    {
        error_flag = 0;
        error_10_6 = Vref - Voltage_FX;
        Mult_R += K1_boost * error_10_6;
    }
}

if(Two_Vref){
    //then calculate error_10_6 and K1 boost
    if(Vrefu <= Voltage_FX)
    {
        error_flag = 1;
        error_10_6 = Voltage_FX - Vrefu;
        Mult_R -= K1_boost * error_10_6;
    }
}

```

```

else if(Vrefl >= Voltage_FX)
{
    error_flag = 0;
    error_10_6 = Vrefl - Voltage_FX;
    Mult_R += K1_boost * error_10_6;

}
}
// normalize MAC accumulator from 12-20 down to 2-14
TDR01_temp = (unsigned short)(Mult_R >> 4);

/*****duty cycle restraints*****/
#if 1
if (TDR01_temp <=D_MIN){
    TDR01_temp = D_MIN;
}
if (TDR01_temp> D_MAX){
    TDR01_temp = D_MAX;
}
#endif
/*****settle new duty cycle*****/
if(Normal_ADC|Window_ADC)
TDR01 = (TDR01_temp>>5) + (TDR01_temp>>7);

if(Open_Loop)
    TDR01 = 320;

```

The idea to implement the software is to use fixed point math, shifts, and approximation to eliminate the float point calculation. In this way, we can avoid as many call functions as possible and reduce the code run time.

The approximations are implemented in the following two lines:

```
Voltage_FX = (gADC_Result>>8) + (gADC_Result>>9) + (gADC_Result>>15);
```

```
TDR01 = (TDR01_temp>>5) + (TDR01_temp>>7);
```

The approximation is that when an integer variable is multiplied by a float, we can split the float number into several integers and decimal numbers which are equal to two to the power of some integers.

The fixed point technique changes the float point number into a proper fixed point number format. For example, if we want to convert a float point number to a 16-bit fixed point

number and define the highest two bits as integer part, the other bits as decimal part, we need to multiply the input value by 2^{14} .

3.3.3.3 Object Code of Control Loop

The object code of the control loop part is shown in Appendix E.

According to the object code, the control flow graph reflecting the relationship between different blocks is shown in Figure 3-8.

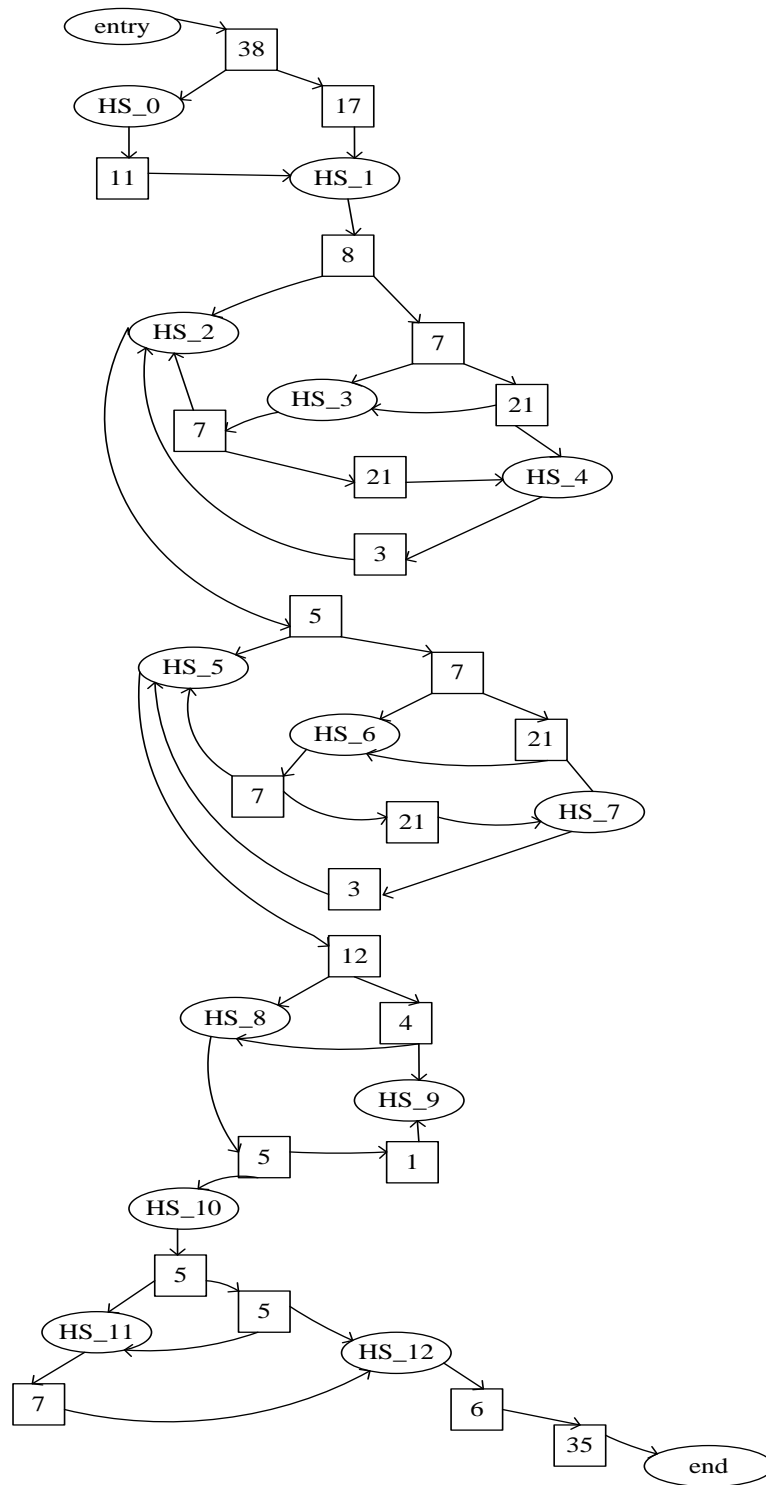


Figure 3-8 Object code control flow graph

If we take time-triggered control mode as an example, according to the control flow graph, the number of cycles for this case is 128 (before the calculation of CPU usage parameters), divide the number with the system clock frequency, which is 32MHz, the time-triggered control mode ISR needs 4 μ s.

3.3.3.4 Execution Time of Control Loop

According to the analysis of object code, the analytical ISR routine time is about 4 μ s. Besides this method, we can also get the control execution time by experiment. Figure 3-9 shows the time-triggered control execution time from the oscilloscope, in which the execution time is also about 4 μ s, which matches with the analytical result.

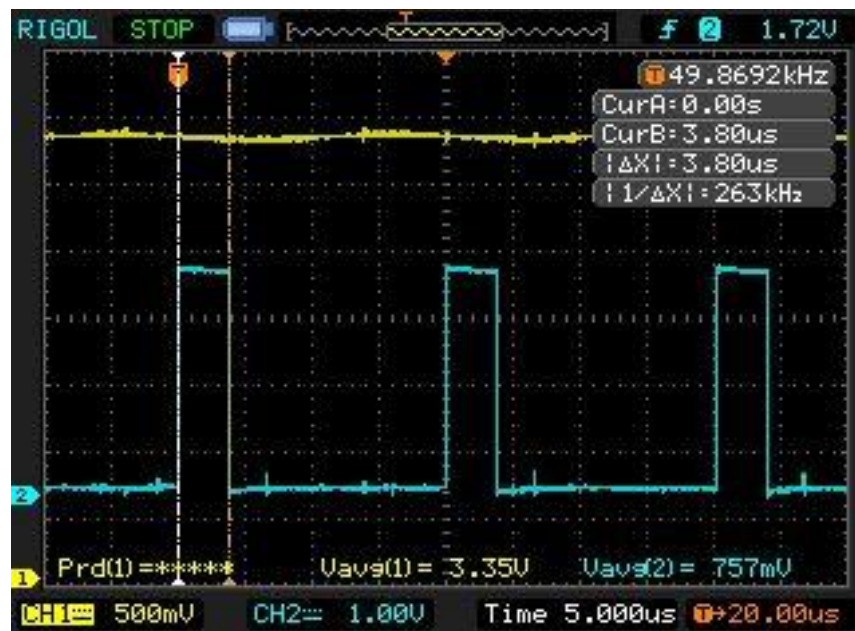


Figure 3-9 Blue trace shows time ADC ISR is active and executing control loop

3.4 Event-Based (Dead-Band) Control Timing Analysis and Implementation

With the time-triggered PI control loop code given above as a starting point, we now proceed to examine how to reduce its use through event-based control.

3.4.1 Event-Based Control Timing Analysis

We wish to understand how the boost converter and controller respond to a step increase in load current (a load transient of ΔI_{out}), which is shown in Figure 3-10. In sampling-triggered control policy with one reference voltage, the output voltage transient response is shown in the following figure. To simplify the analysis, we assume that the load transient is synchronized with the switching cycle. This assumption can be removed easily in future work.

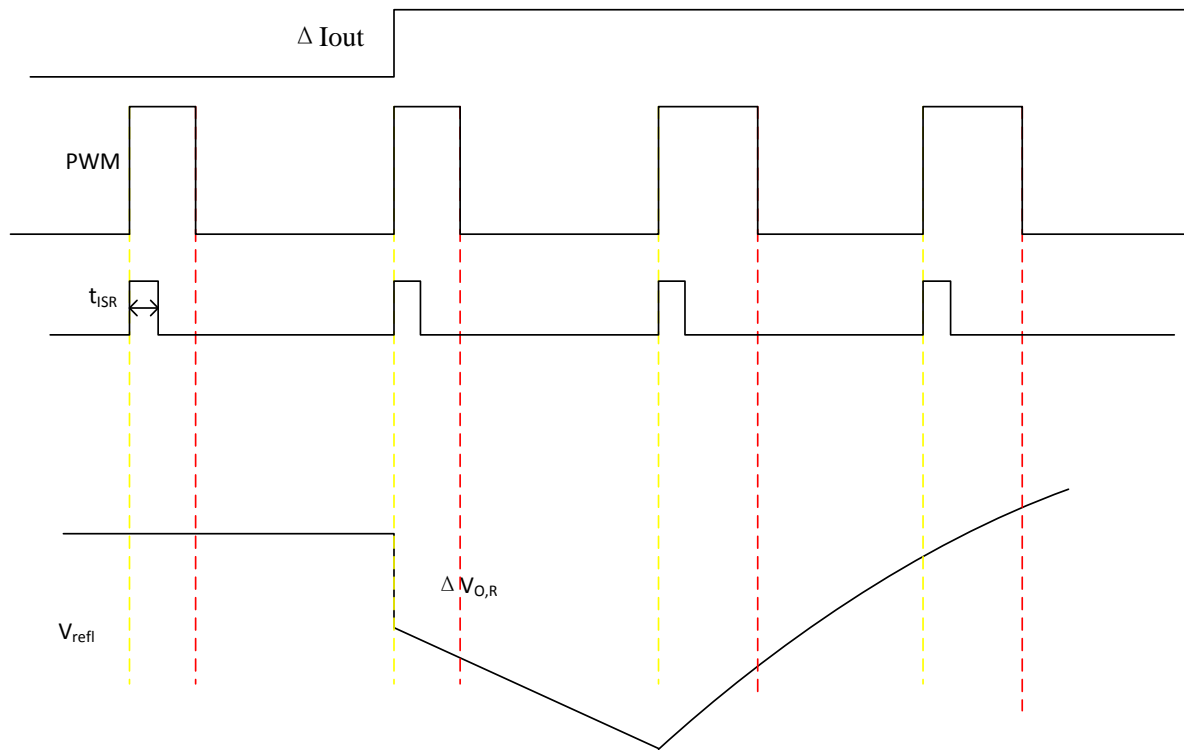


Figure 3-10 Transient response in time-triggered control mode

In this project, the event-triggered (also called dead-band) control is used. Because it needs some time for the output voltage to go out of the bounds, the timing analysis thus has some additional delay and special characteristics compared with normal control.

The analysis will be based on the situation that the output current has a sudden increase. There are three cases according to the relationship among the output voltage before the transient response (V_{ini}), the lower bound of the deadband (V_{refl}) and the voltage drop in the first phase of the transient response ($\Delta V_{O,R}$). In the first case, the absolute value of $V_{ini} - V_{refl}$ is larger than the first phase of voltage change, but the output voltage will still go out of the dead bands. In the second case is the absolute value of $V_{ini} - V_{refl}$ is no larger than the first

phase of voltage change. In the third case the transient response process never goes out of the deadband.

From Peterchev's and Redl's research [4], $\Delta V_{O,R}$ is proportional to the change of the current ΔI , which follows the equation below:

$$\Delta V_{O,R} = \Delta I R_C$$

Because ΔI results from a sudden load resistor change in my project, it follows the relationship below, assuming this is the optimal control system and that the load resistor has a sudden decrease which causes voltage output transient response.

$$\Delta I = I_{\text{after}} - I_{\text{before}}$$

$$I_{\text{after}} = V_o / R_{o,\text{after}}$$

$$I_{\text{before}} = V_o / R_{o,\text{before}}$$

3.4.1.1 Timing Analysis with Averaged Output Voltage

a. Case 1: $V_{\text{ini}} - V_{\text{refl}} > \Delta V_{O,R}$

The picture representing this case is shown in Figure 3-11.

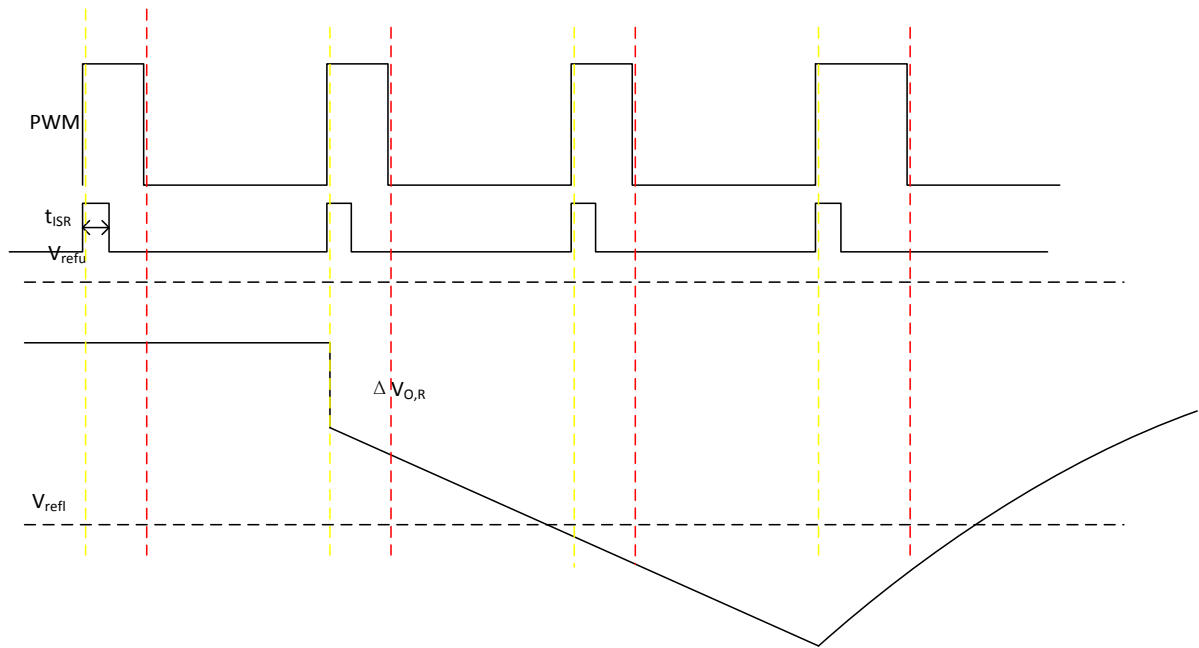


Figure 3-11 Transient response in dead-band control in case $V_{ini} - V_{refl} > \Delta V_{O,R}$

In this case, the ADC will delay for one period, because ADC only occurs at the beginning of the PWM period. The PWM duty cycle will be updated at the end of the same period where the ADC triggers an interrupt. The output voltage begins to rise at the next period.

b. Case 2: $V_{ini} - V_{refl} \leq \Delta V_{O,R}$

This case is shown in Figure 3-12.

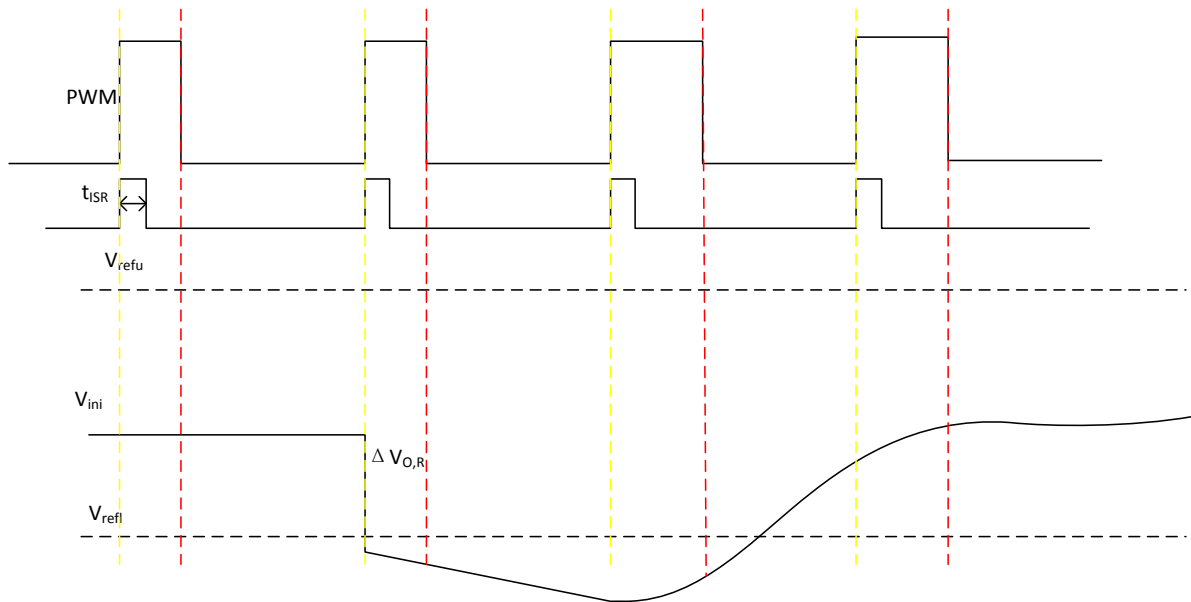


Figure 3-12 Transient response in dead-band control in case $V_{ini} - V_{refl} \leq \Delta V_{O,R}$

In this case, ADC will trigger an interrupt just in the PWM period where the voltage begins to change, because the voltage drops out of the band at the very beginning of the response. The voltage will rise at the next period.

c. Case 3: The output transient response doesn't fall out of the dead bands.

This case is shown in Figure 3-13.

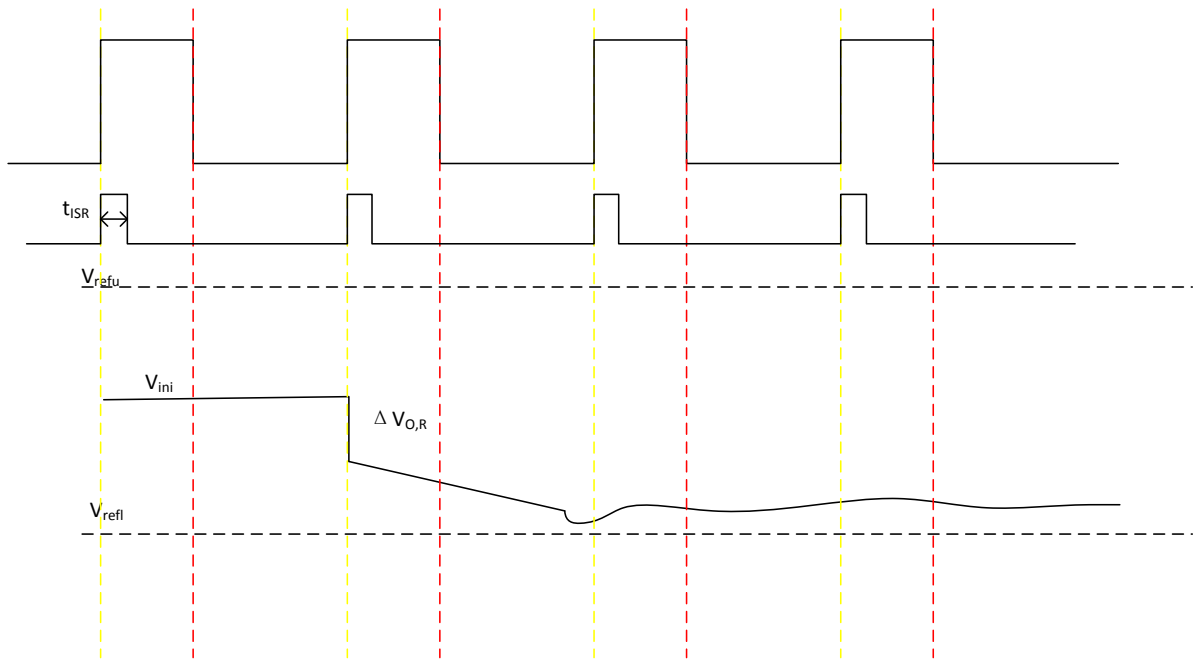


Figure 3-13 Transient response in dead-band control in never-fall-out case

In this case, the output voltage will perform like the open-loop voltage response. The time period from the beginning of the voltage change to the point when the voltage comes to another steady state. The settling time t_{settle} and the voltage change between the two steady states ΔV_{settle} can be estimated by the following equations:

$$t_{\text{settle}} = \frac{\pi\sqrt{LC_{\text{out}}}}{2} \pm T_{\text{sw}}$$

$$\Delta V_{\text{settle}} = -\Delta I_{\text{out}}(R_C + R_L)$$

3.4.1.2 Timing Analysis When Ripple Is Considered

The situations above are based on average output voltage. The real situation includes ripple caused by PWM signals. In Figure 3-14, the blue signal is the output voltage with the ripples. The black signal is the average output voltage. As shown in Figure 3-14, from the view of the averaged output voltage, it falls out of the lower band at the second sampling period. The

controller will work at the third period. If the ripple is considered, the real output will fall out of the dead-band earlier than the averaged output but the fall-out points are always in the same period. If the fall-out points of the real output voltage and the averaged output voltage are both later than ISR run period, the controller will still begin to work in the third period. If the fall-out point of the real output voltage locates within the ISR but the average output fall-out point doesn't, the controller will begin to work one period earlier.

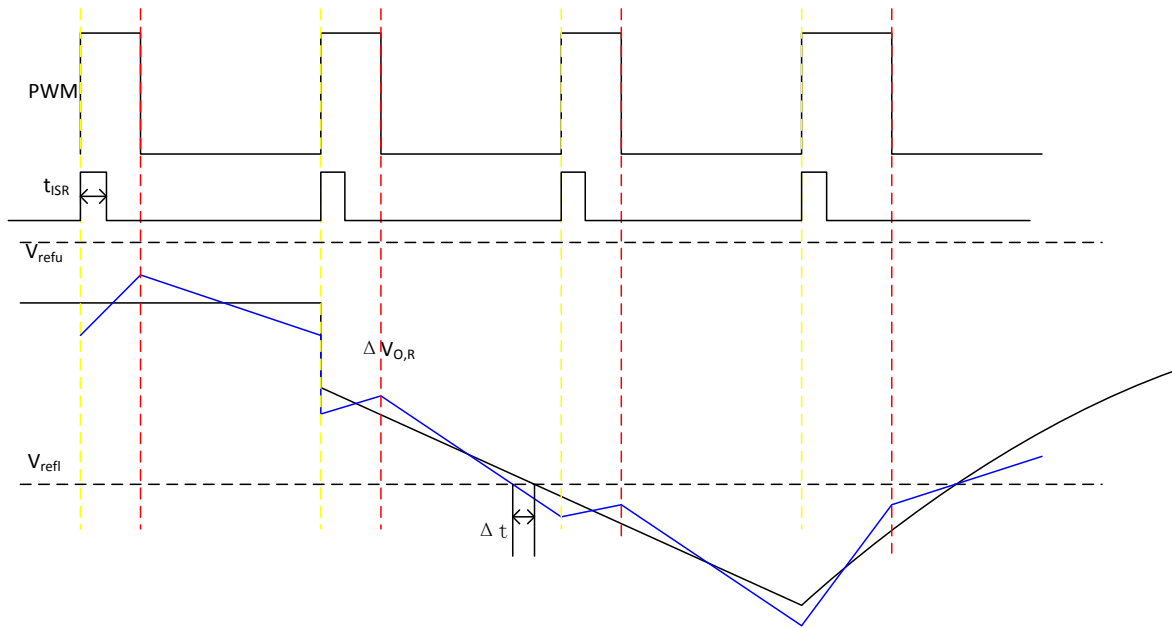


Figure 3-14 Transient response of dead-band control in real situation, first fall-out point after ISR

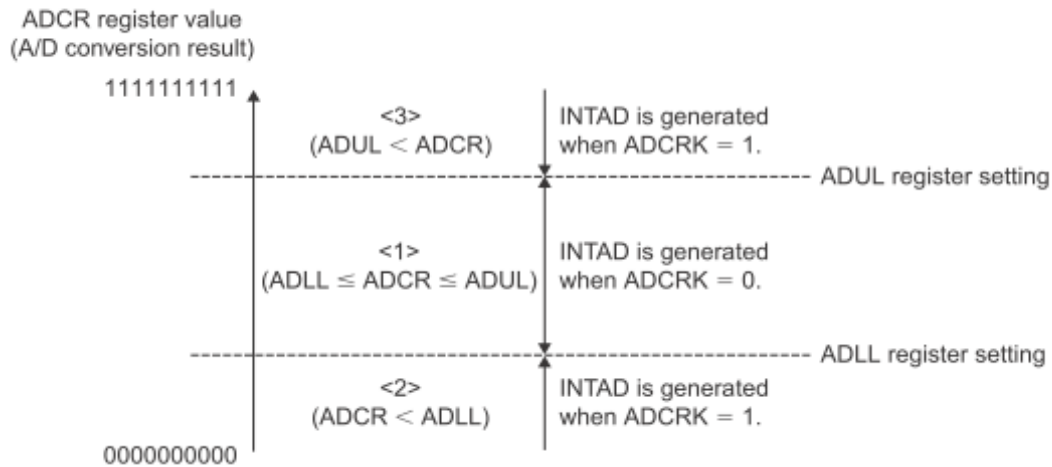


Figure 3-16 Indication graph of three A/D conversion modes

Figure 3-15 shows two A/D conversion interrupt trigger modes, selected by the control bit ADCRK. Mode 1 is that interrupt is triggered when A/D result is within a range; mode 2-3 is the interrupt is triggered when A/D result is out of a preset range. The decision is made by checking two 8-bit registers: ADUL and ADLL, which are the pre-written values and will be compared with the highest 8 bits of A/D conversion result ADCR. The register storing that highest 8 bits of ADCR is ADCRH. ADUL is the upper bound, ADLL is the lower one.

To implement event-based control, we use A/D converter to convert output voltage to digital value, set A/D conversion interrupt trigger mode in mode 2-3 and set ADUL/ADLL as the highest 8 bits of the value converted from the dead band value by the same rule as of the output voltage converting to ADCR. When ADCRH is larger than ADUL or smaller than ADLL, the interrupt is triggered and the control is implemented, i.e., the event-based control is implemented.

3.4.3 Two Reference Voltages Implementation

Because in the project, dead-band control is implemented, steady state error within the dead-band is acceptable. When the output voltage falls out of the dead-band, PI control will be

applied. The error from the middle of the bands to the real output is larger than the error from the closer dead-band edge to the output. If one reference voltage is applied, the error is potentially larger than the error calculated when two reference voltages are applied. Larger error will lead to a larger control signal, which can create a larger oscillation of the output signal. To avoid oscillation in steady state as much as possible, I apply two reference voltages in the project.

The difference of the results with one and two reference voltages is shown in Figure 3-16.

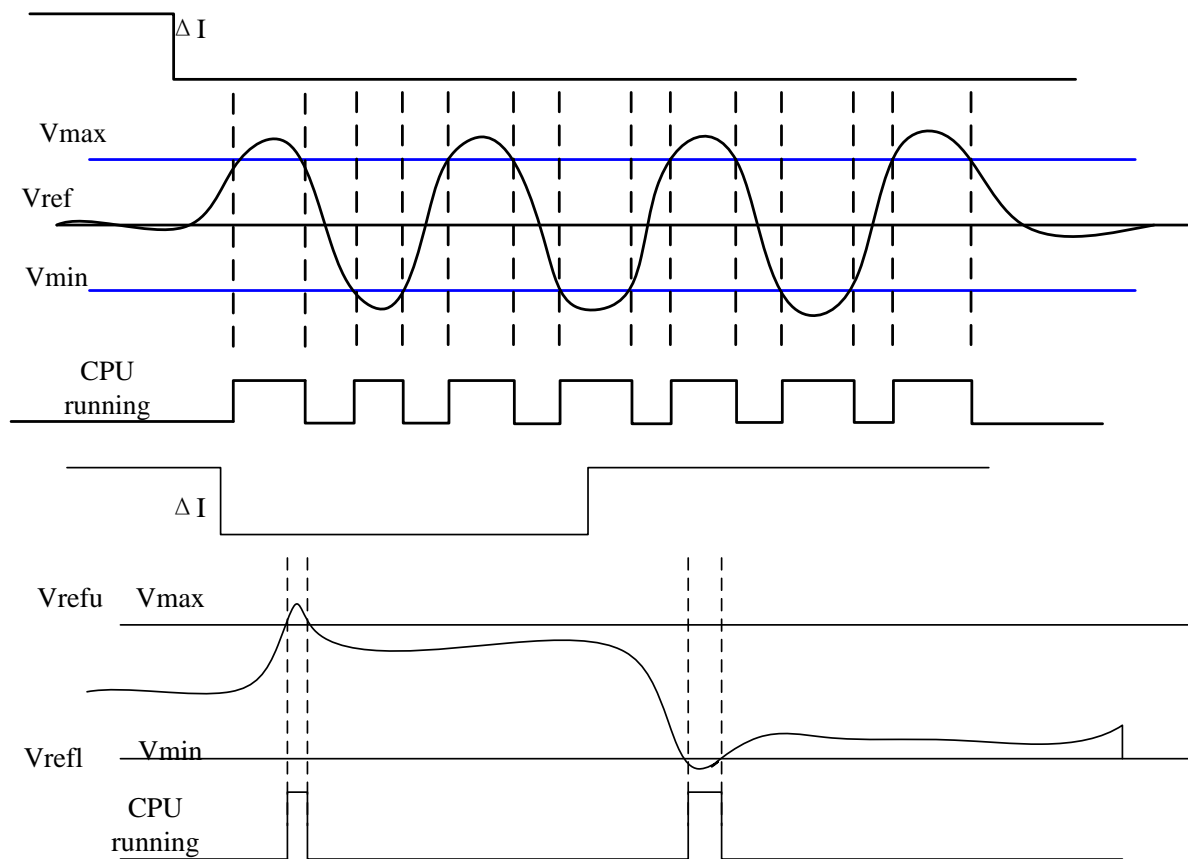


Figure 3-17 Difference between one-reference-voltage mode and two-reference-voltage mode

There are three interrupt sources related in this software design. The first interrupt source is a timer interrupt triggered every 1 microsecond, which uses channel 0 in Timer Array Unit 1. Entering this interrupt means the hardware counter counts down to zero and will be reset to the upper bound . The second interrupt source is a timer interrupt triggered every 20 millisecond, which uses channel 2 in Timer Array Unit 1. When entering this interrupt, CPU usage is calculated, afterwards the Cpu_Counter will be reset to zero. The third interrupt source is ADC interrupt. When the A/D conversion finishes, if the output voltage falls out of the bands, the system will enter this interrupt and implement PI control.

The hardware counter interrupt and the ADC interrupt have the highest priority. When the system enters ADC interrupt, we capture the hardware counter values at the beginning and the end of the ISR routine, and subtract the two values to get the ISR routine time. Summarize the ISR routine time together throughout 20 ms. The summation is stored in the variable Cpu_Counter , the CPU usage calculation process follows the following equation:

$$\text{cpu_usage} = \frac{\text{Cpu_Counter}}{20000}$$

Chapter 4 EXPERIMENTAL EVALUATION AND SIMULATION

4.1 Simulation

In the project, I use PLECS to simulate the closed loop control performance. The simulation circuit and connection is in Figure 4-1.

For the transfer function part, I use the following function:

$$\frac{D(z)}{E(z)} = \frac{0.033z - 0.03}{z - 1}$$

The simulation output is in Figure 4-2.

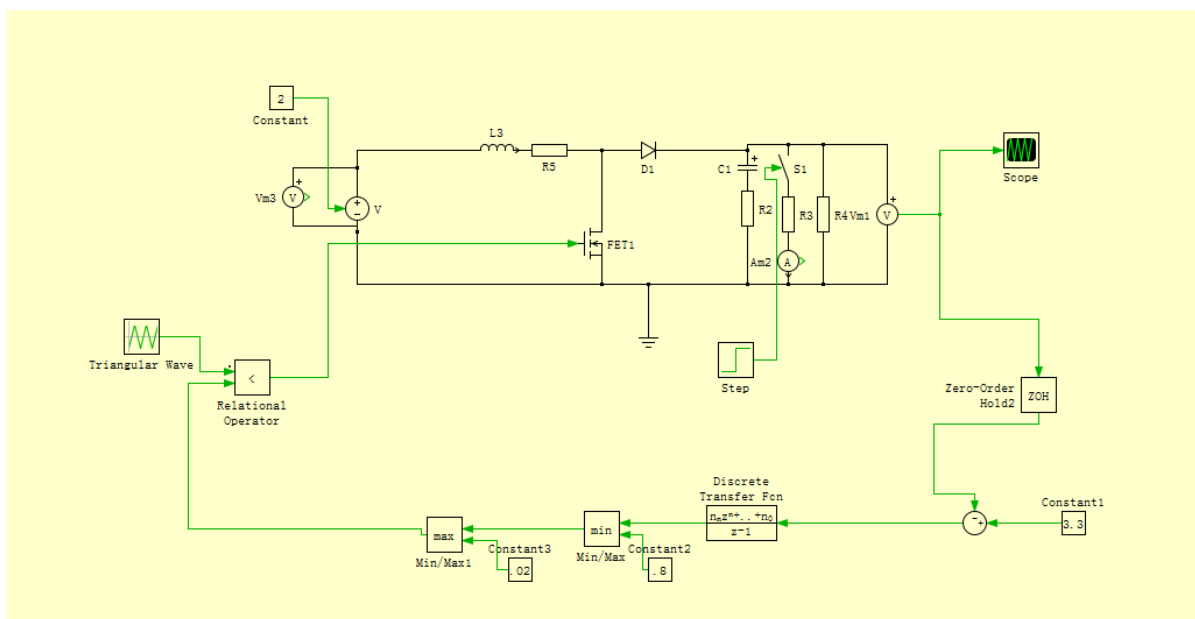


Figure 4-1 Simulation circuit graph with PLECS

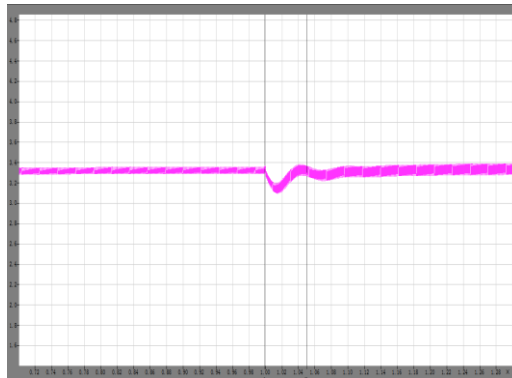


Figure 4-2 Simulation result

4.2 Time-Triggered Control and Dead-Band Control

In this part, we will look at and compare two types of control strategies: time-triggered control, which is implemented at the beginning of every sampling period; and dead-band control, in which control is only performed when the voltage goes out of the dead-band. After showing the waveform, we will see the difference of the CPU usage between them.

The CPU usage can be viewed and calculated in a direct way. In both control strategies, control will only be applied at the beginning of some sampling periods. If we use a digital output port bit, set it to 1 when the control is applied, reset it to 0 when control is finished, the waveform of this port bit will have the same length of the high level time in a specific situation. The method to estimate CPU usage is to count the number of control loop run times, multiply it by the control loop execution time, and divide the result by the measurement time period.

Next, we will look at the output waveforms and their corresponding CPU usage of three kinds of load: constant load, periodically changed load and a real Wi-Fi module load. In each kind of load case, we will show one sub case in detail, and put the waveforms of the other cases in to appendices.

In the end, we will come to summarize and analyze the experiment results, sort them into three graphs, showing the relationship between the output voltage and the size of the dead band in three types of control.

4.2.1 Test Load (Constant)

We first observe and compute the CPU usage of the constant load case in time-triggered control mode, dead-band control mode with one reference and dead-band control mode with two reference voltages. The constant control is a resistant load with the resistance of 47Ω . The desired output voltage is 3.3V.

4.2.1.1 Time-Triggered Control

In the time-trigger control, the output waveform with the control loop running is shown in Figure 4-3.

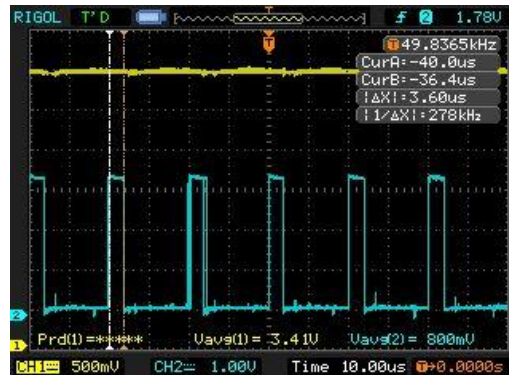


Figure 4-3 Constant load in time-triggered control: ISR indication and output

The control is triggered once every $20\mu\text{s}$. The control execution time is about $4\mu\text{s}$. So CPU usage is about $4/20 = 0.20$;

4.2.1.2 Dead-Band Control

4.2.1.2.1 Dead-Band Control with One Reference Voltage

We will look at the waveform of the first case. The rest of the waveforms are listed in Appendix C.

a. Dead-band: 3.397V to 3.185V

The output against ISR indication is shown in Figure 4-4.

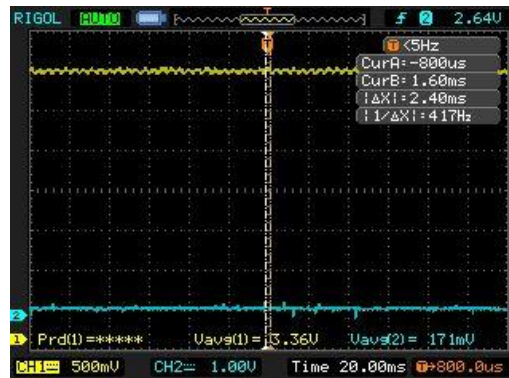


Figure 4-4 Constant load in dead-band control with one-reference-voltage mode in bands: 3.397V to 3.185V

Number of control times: 0;

CPU usage is 0.

b. Dead-band: 3.373V to 3.208V

Number of control times: 0;

CPU usage is 0.

c. Dead-band: 3.350V to 3.232V

Number of control times: 9;

Control execution time is $4.4\mu\text{ s}$;

CPU usage is 0.00198.

d. Dead-band: 3.326V to 3.255V

Number of control times: 32;

Control execution time is $4.4\mu\text{ s}$;

CPU usage is $32 * 4.4 / 20000 = 0.00704$.

4.2.1.2.2 Dead-Band Control with Two Reference Voltages

a. Dead-band: 3.397V to 3.185V

The output against ISR indication is shown in Figure 4-5.

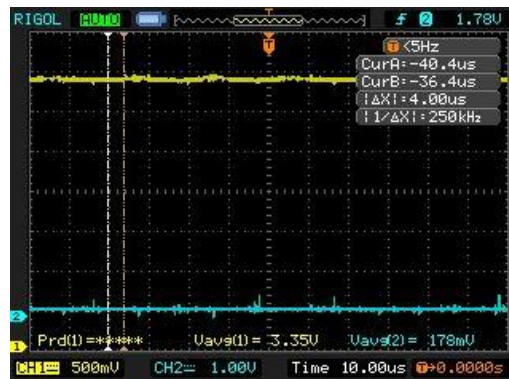


Figure 4-5 Constant load in dead-band control with two-reference-voltage mode in bands: 3.397V to 3.185V

Number of control times: 0;

CPU usage is 0.

b. Dead-band: 3.373V to 3.208V

Number of control times: 0;

CPU usage is 0.

c. Dead-band: 3.350V to 3.232V

Number of control times: 6;

Control execution time is 4.4 μ s;

CPU usage is 0.00132.

d. Dead-band: 3.326V to 3.255V

Number of control times: 40;

Control execution time is 4.4 μ s;

CPU usage is $40 \times 4.4 / 20000 = 0.0088$.

4.2.2 Test Load (Periodic)

4.2.2.1 Time-Triggered Control

In the sampling-triggered control, there is only one reference voltage and no dead band. The control frequency is the same as the sampling frequency. In this experiment, we set the sampling frequency as 50 kHz, and the reference voltage as 3.3V. There are two 47 Ω resistor loads. One is the fixed load which is always connected between the output voltage and the ground. The other is the switching load which will only be connected between the output voltage and the ground when the MOSFET is on.

The output voltage (the blue signal) against the load change (the yellow signal) has the waveform in Figure 4-6.



Figure 4-6 Periodical load in time-triggered control: transient response

The waveform indicating CPU usage is in Figure 4-7.

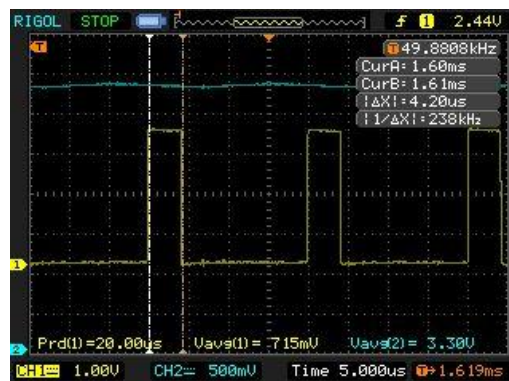


Figure 4-7 Periodical load in time-triggered control: ISR indication graph with output

We see that the CPU utilization is $4.2/20 = 21\%$.

4.2.2.2 Dead-Band Control

In the dead-band control, we have the same fixed resistor load and switching load. The following experiments will be divided into two groups: dead-band control with one reference voltage; and dead-band control with two reference voltages. To be succinct, I will only show one case in detail in each group, including the related waveform and CPU usage computation process. The waveforms of the rest cases are listed in Appendix. The CPU usage results are shown in the analysis part.

4.2.2.2.1 Dead-Band Control with One Reference Voltage

In this case, the reference voltage is set to 3.3V and the dead-band is set to different widths. We will observe the following four sub cases.

a. Dead-band: 3.397V to 3.185V

The waveform and CPU usage indication is shown in Figure 4-8.

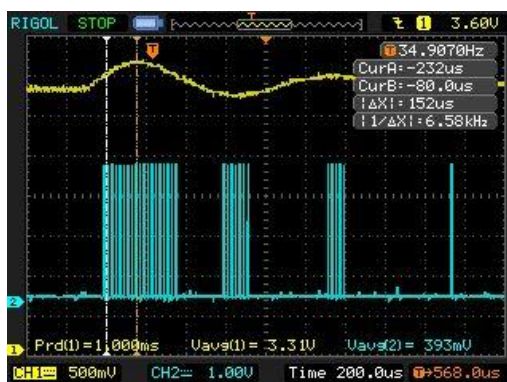


Figure 4-8 Periodical load in dead-band control with one-reference-voltage mode in bands: 3.397V to 3.185V

There are 39 control execution times in average during one load switching period. The Control execution time is about 4μ s. So CPU usage is about $39 * 4 / 20000 = 0.0078$.

b. Dead-band: 3.373V to 3.208V

Number of control times: 60;

Control execution time is 4μ s;

CPU usage is $60 * 4 / 20000 = 0.012$.

c. Dead-band: 3.350V to 3.232V

Number of control times: 130;

Control execution time is 4μ s;

CPU usage is $130 * 4 / 20000 = 0.026$.

d. Dead-band: 3.326V to 3.255V

Number of control times: 245;

Control execution time is 4μ s;

CPU usage is $245 * 4 / 20000 = 0.049$.

4.2.2.2.2 Dead-Band Control with Two Reference Voltages

In these cases, the reference voltage is set to be the closer edge of the dead-band. We will observe the waveforms and CPU usage of one case. The waveforms of the rest three cases are in Appendix B.

a. Dead bands are 3.397V to 3.185V

The output against ISR indication is shown in Figure 4-9.

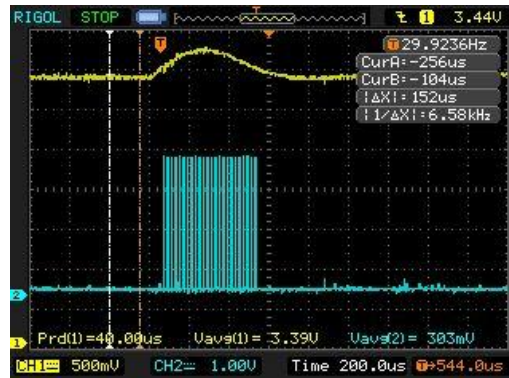


Figure 4-9 Periodical load in dead-band control with one-reference-voltage mode in bands: 3.397V to 3.185V

Number of control times: 24;

Control execution time is 4μ s;

CPU usage is $24 \cdot 4 / 20000 = 0.0048$.

b. Dead-band: 3.373V to 3.208V

Number of control times: 42;

Control execution time is 4μ s;

CPU usage is $42 \cdot 4 / 20000 = 0.0084$.

c. Dead-band: 3.350V to 3.232V

Number of control times: 96;

Control execution time is 4μ s;

CPU usage is $96 \cdot 4 / 20000 = 0.0192$.

d. Dead-band: 3.326V to 3.255V

Number of control times: 168;

Control execution time is 4μ s;

CPU usage is $168 \cdot 4 / 20000 = 0.0336$.

4.2.3 Wi-Fi Module as Load

After evaluating the performance of the dead-band control on a synthetic, periodic load, we wish to evaluate its performance on a real application with more complicated power requirements. We use the Gainspan GS1011MIPS Wi-Fi module, which is built into the RDK development board. This module operates at 3.3 V and draws a wide range of currents based on radio activity (receiving, transmitting, idle).

In this experiment, the boost converter powers the Wi-Fi module on the other G14 RDK, which is actively sending and receiving data. To sense the voltage change in accordance to the current change, a current sensing resistor placed in series between the voltage output pin of the boost converter, and voltage input pin of the Wi-Fi module. The graph is shown below.

When Wi-Fi module transmits signals, the input current of the Wi-Fi module changes with the signals. The input voltage of Wi-Fi module will change with the current change.

The following experiment results show the initial setup state of Wi-Fi module and the steady state after setting up. The first experiment is based on that the system uses two reference voltage dead-band control. The dead bands are 3.326V to 3.255V. The second sets of experiments show the CPU usage and the dead-band size with one or two reference voltages.

4.2.3.1 Input Voltage vs. Input Current.

General view of the initial state and steady state waveform is shown in Figure 4-10.



Figure 4-10 Wi-Fi module: initial state and steady state

The initial state after zooming in is shown in Figure 4-11.

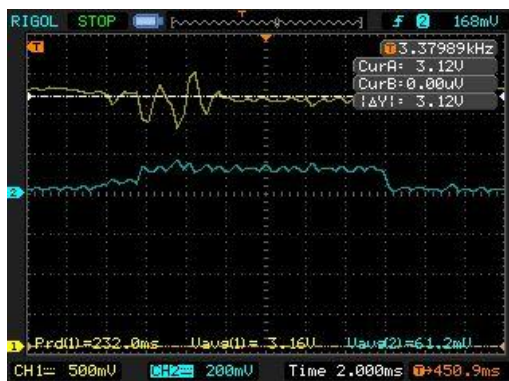


Figure 4-11 Wi-Fi module: initial state

The steady state after zooming in is shown in Figure 4-12.



Figure 4-12 Wi-Fi module: steady state

4.2.3.2 CPU Usage in Dead-Band Control with One or Two Reference Voltages

For the CPU usage, when dead-band size increases, the CPU usage has the following changes, which are shown from Figure 4-13 through Figure 4-20.

a. Dead-band: 3.326V to 3.255V

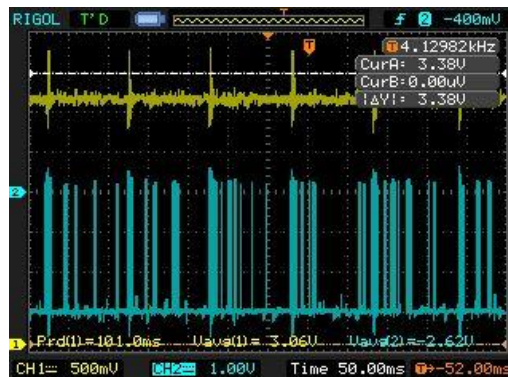


Figure 4-13 Wi-Fi module: one-reference-voltage mode with bands 3.326V to 3.255V

CPU usage for one reference voltage situation:0.06827

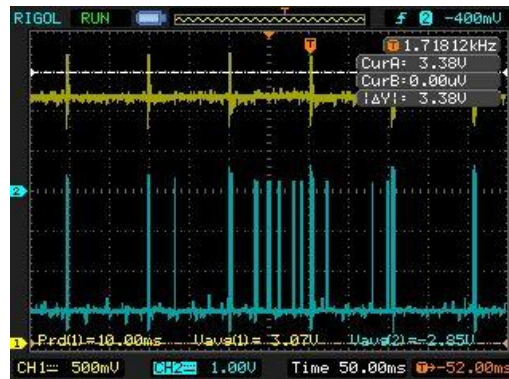


Figure 4-14 Wi-Fi module: two-reference-voltage mode with bands 3.326V to 3.255V

CPU usage for two reference voltages situation:0.06137

b. Dead-band: 3.350V to 3.232V



Figure 4-15 Wi-Fi module: one-reference-voltage mode with bands 3.350V to 3.232V

CPU usage for one reference voltage situation:0.06189

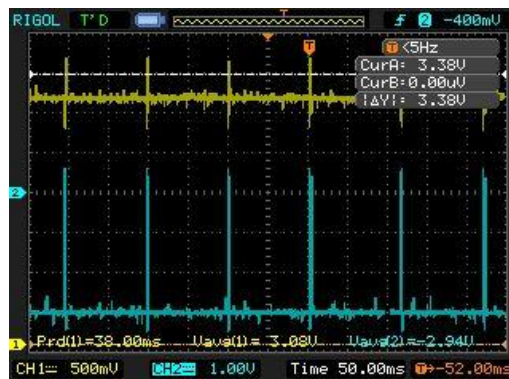


Figure 4-16 Wi-Fi module: two-reference-voltage mode with bands 3.350V to 3.232V

CPU usage for two reference voltages situation:0.05509

c. Dead-band: 3.373V to 3.208V

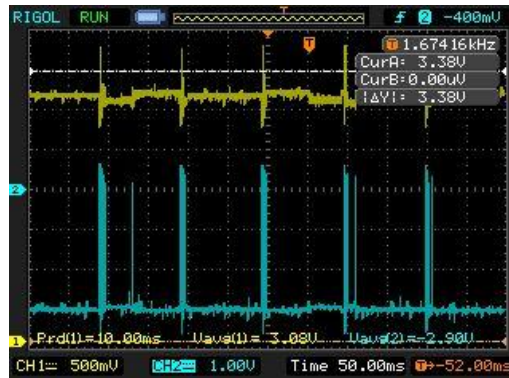


Figure 4-17 Wi-Fi module: one-reference-voltage mode with bands 3.373V to 3.208V

CPU usage for one reference voltage situation:0.05983



Figure 4-18 Wi-Fi module: two-reference-voltage mode with bands 3.373V to 3.208V

CPU usage for two reference voltages situation:0.05461

d. Dead-band: 3.397V to 3.185V



Figure 4-19 Wi-Fi module: one-reference-voltage mode with bands 3.397V to 3.185V

CPU usage for one reference voltage situation:0.05811



Figure 4-20 Wi-Fi module: two-reference-voltage mode with bands 3.397V to 3.185V

CPU usage for two reference voltages situation:0.05439

4.3 Analysis

In this section, we will plot the diagrams showing the CPU usage against the dead band size in different control modes for the above three loads.

4.3.1 Analysis for Constant Load

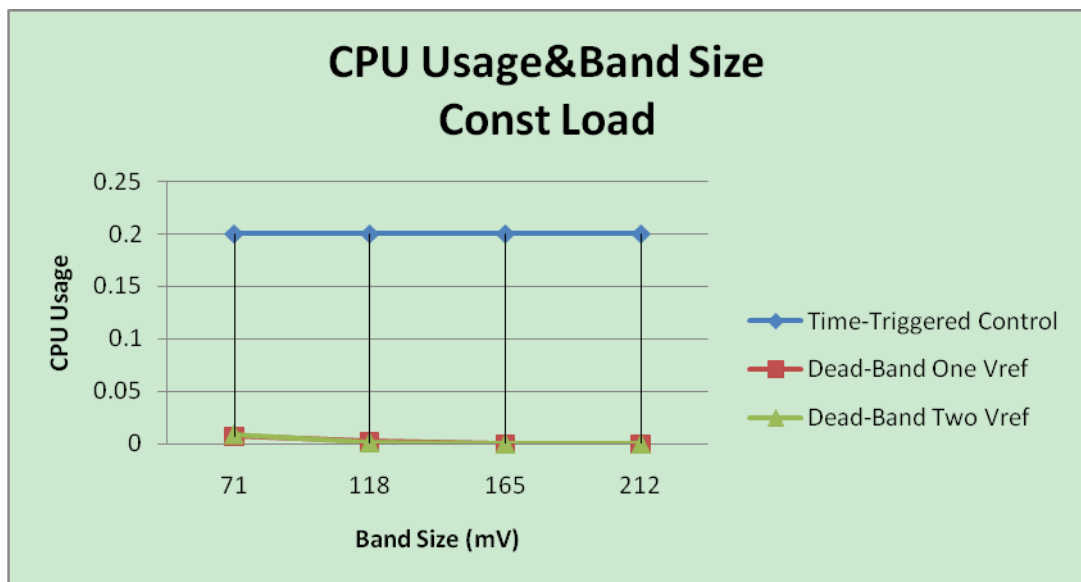


Figure 4-21 Analysis: CPU usage of constant load

From Figure 4-21, we can see the dead-band control reduces about 90% of CPU utilization compared with time-triggered control. Dead-band control with two reference voltages has even less utilization than the other two control modes.

4.3.2 Analysis for Periodic Load

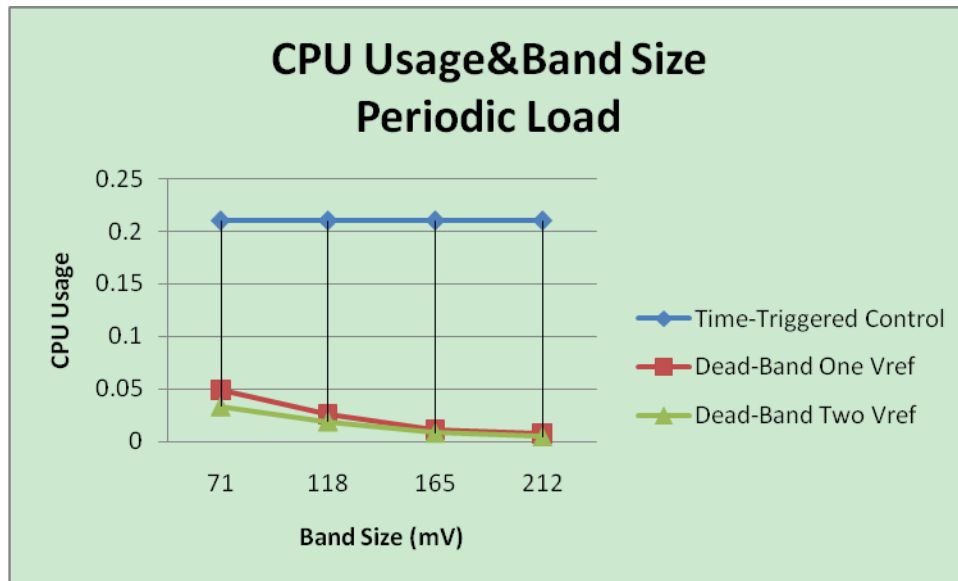


Figure 4-22 Analysis: CPU usage of periodical load

From Figure 4-22, we can see the dead-band control reduces 75% of CPU utilization compared with time-triggered control. Dead-band control with two reference voltages has even less utilization than the other two control modes.

4.3.3 Analysis for Wi-Fi Module

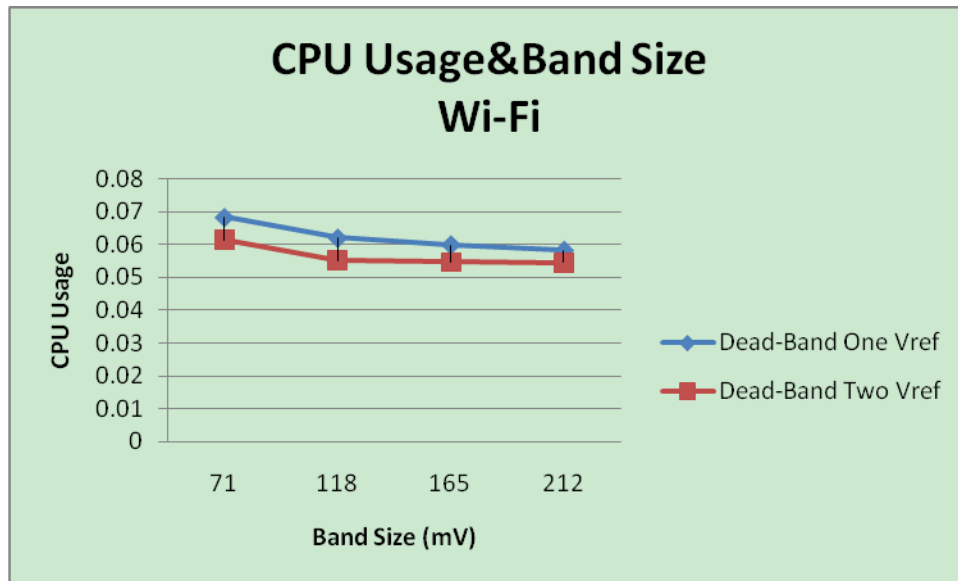


Figure 4-23 Analysis: CPU usage of Wi-Fi module

Because we would like to specifically test dead-band control in Wi-Fi module, we don't record the time-triggered control for it. From Figure 4-23, for the two dead-band control modes, the one with two reference voltages has about 14.28% of CPU utilization lower than the one with one reference voltages.

Chapter 5 CONCLUSION AND FUTURE WORK

5.1 Conclusion

In this thesis, I implemented dead-band control, compared the CPU usage between it and sampling-triggered control. I conclude that dead-band control will reduce the CPU usage significantly. The amount of reduction depends on the load transient size and frequency, dead-band size, PI control parameter setting and the number of reference voltages. The results are based on the overall integrations and relationships among the factors above. The factors have the following effects on CPU usage.

Increasing dead-band size will potentially result in reduction of CPU usage. However, if the system has poor PI parameters, the system will oscillate and goes out of the dead-band a lot, which increases CPU usage. This problem could be solved by using two reference voltages without changing the PI parameters.

If control signal is large, the output voltage will converge back to reference voltage(s) fast, which means the system has longer time to be in the dead-band. This will reduce CPU usage. But it may cause oscillations between two bands. As mentioned below, two reference voltages could solve this problem, but the system performance will decay and could have larger steady-state error.

If two reference voltages are used, the control signal is smaller compared with one reference voltage. If the system is not very stable with one reference voltage, two reference voltages could stabilize the system and keep the system within the dead-band longer, which reduces CPU usage. However, two reference voltages method may cause larger steady-state error, and makes the system more sensitive to noise.

5.2 Future work

The inductor used in the boost converter is $150\ \mu\text{H}$, which makes the system very insensitive to the control signal. To make the system react to the control signal change faster, a smaller inductor should be used.

Besides, the current system still has relatively large overshoot. To compensate this overshoot, feed forward control may be implemented in the future. What's more, feed forward control may also lead to a faster control response. [10]

For the code optimization part, the current fixed point number method is hard to understand to some extent. Which will cause potential maintain problem. To solve this problem, developing a small tool to convert float point number operation to fixed point number operation should be a good idea.

REFERENCES

- [1] Hauke, Brigitte, Texas Instruments, "Basic Calculation of a Boost Converter's Power Stage," Application Report SLVA372C, 2009.
- [2] Mitulkumar R. Dave and K. C. Dave, "Analysis of Boost Converter Using PI Control Algorithms," *International Journal of Engineering Trends and Technology*, vol. 3, no. 2, 2012.
- [3] Kiam Heong Ang and Gregory Chong, "PID Control System Analysis, Design, and Technology," *IEEE Transactions on Control Systems Technology*, vol. 13, no. 4, pp. 559-576, July 2005.
- [4] Angel Vladimirov Peterchev and Seth R. Sanders, "Digital Control of PWM Converters: Analysis and," Electrical Engineering and Computer Sciences University of California at Berkeley, Berkeley, Technical Report No. UCB/EECS-2006-146, 2006.
- [5] Karl-Erik Årz ́n, "A Simple Event-Based PID Controller," in *IFAC World Congress*, 1999.
- [6] Sylvain Durand and Nicolas Marchand, "An Event-Based PID Controller With Low Computational Cost," in *8th International Conference on Sampling Theory and Applications*, Marseille, 2010.
- [7] Sylvain Durand and Nicolas Marchand, "Further Results on Event-Based PID Controller," in *European Control Conference*, Budapest, 2009.
- [8] Nan Xing, Yujuan Lin, and Jinhui Zhang, "Some Improvements on Event-Based PID Controllers," in *the 32nd Chinese Control Conference*, Xi'an, 2013.
- [9] Robert W. Erickson and Dragan Maksimović, *Fundamentals of Power Electronics*, Second Edition ed. Boulder, Colorado, USA: Kluwer Academic Publishers, 2001.
- [10] Marian K. Kazimierczuk and Antonio Massarini, "Feedforward Control of DC-DC PWM Boost Converter," *IEEE Transactions on Circuits and Systems*, vol. 44, no. 2, pp. 143-148, February 1997.

APPENDICES

Appendix A. Waveforms of Dead-Band Control with One Reference Voltage for the Periodically Changed Load

a. Dead-band: 3.326V to 3.255V

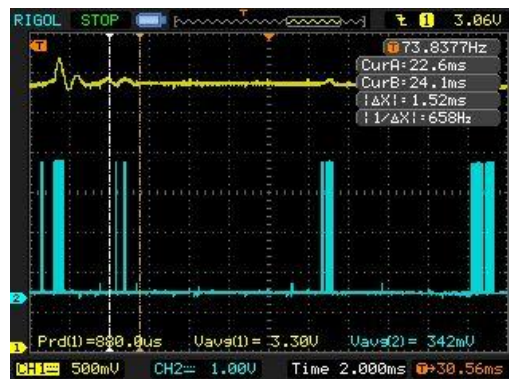


Figure 5-1 Periodical load in dead-band control: one-reference-voltage, dead bands 3.326V to 3.255V, zoom-out version

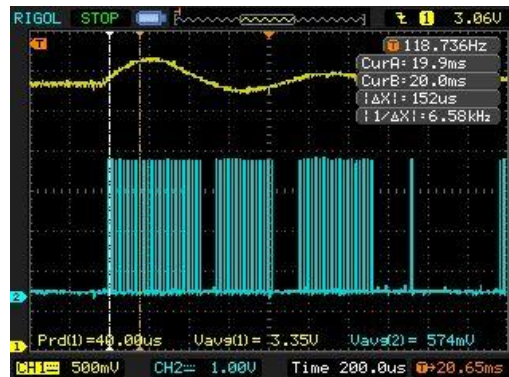


Figure 5-2 Periodical load in dead-band control: one-reference-voltage, dead bands 3.326V to 3.255V, zoom-in version

b. Dead-band: 3.350V to 3.232V

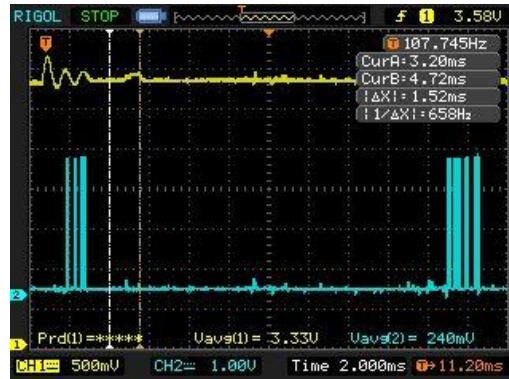


Figure 5-3 Periodical load in dead-band control: one-reference-voltage, dead bands 3.350V to 3.232V, zoom-out version

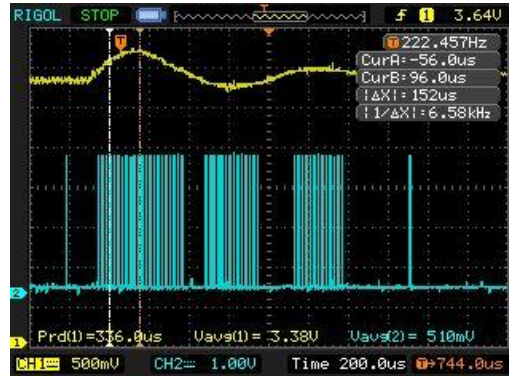


Figure 5-4 Periodical load in dead-band control: one-reference-voltage, dead bands 3.350V to 3.232V, zoom-in version

c. Dead-band: 3.373V to 3.208V

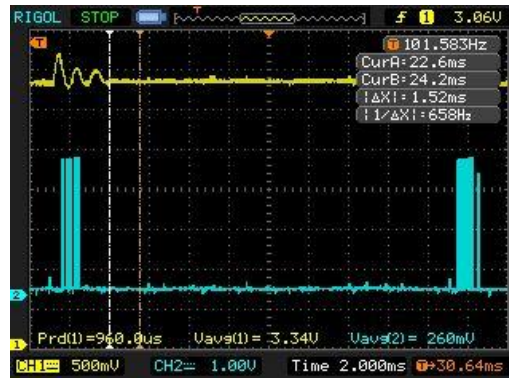


Figure 5-5 Periodical load in dead-band control: one-reference-voltage, dead bands 3.373V to 3.208V, zoom-out version

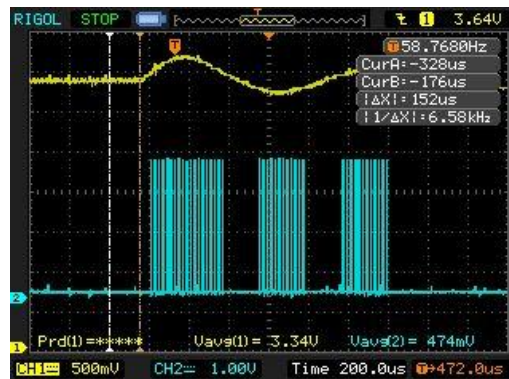


Figure 5-6 Periodical load in dead-band control: one-reference-voltage, dead bands 3.373V to 3.208V, zoom-in version

Appendix B. Waveforms of Dead-Band Control with Two Reference Voltages for the Periodically Changed Load

a. Dead-band: 3.326V to 3.255V



Figure 5-7 Periodical load in dead-band control: two-reference-voltage, dead bands 3.326V to 3.255V, zoom-out version

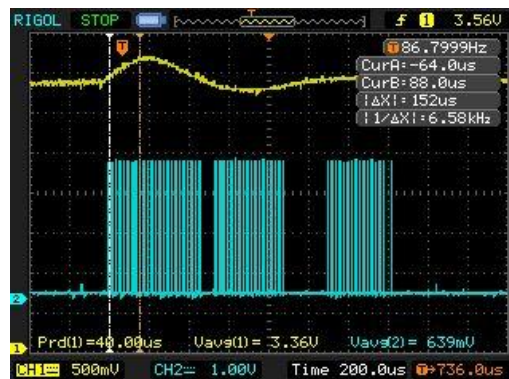


Figure 5-8 Periodical load in dead-band control: two-reference-voltage, dead bands 3.326V to 3.255V, zoom-in version

b. Dead-band: 3.350V to 3.232V



Figure 5-9 Periodical load in dead-band control: two-reference-voltage, dead bands 3.350V to 3.232V, zoom-out version

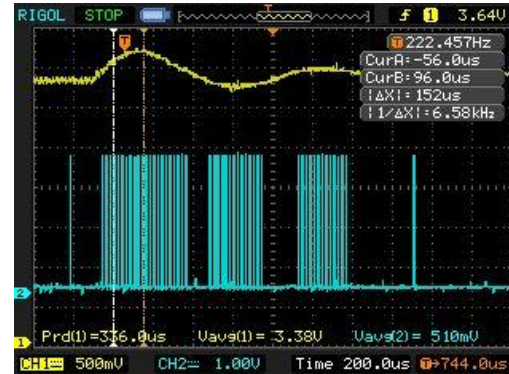


Figure 5-10 Periodical load in dead-band control: two-reference-voltage, dead bands 3.350V to 3.232V, zoom-in version

c. Dead-band: 3.373V to 3.208V

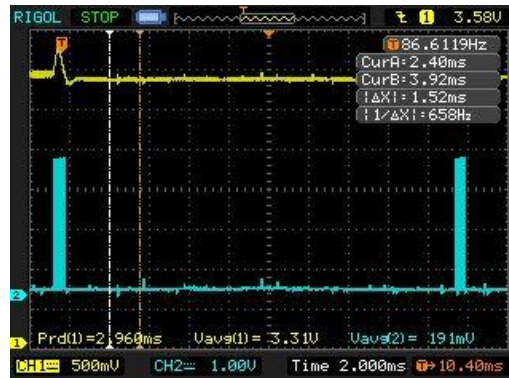


Figure 5-11 Periodical load in dead-band control: two-reference-voltage, dead bands 3.373V to 3.208V, zoom-out version

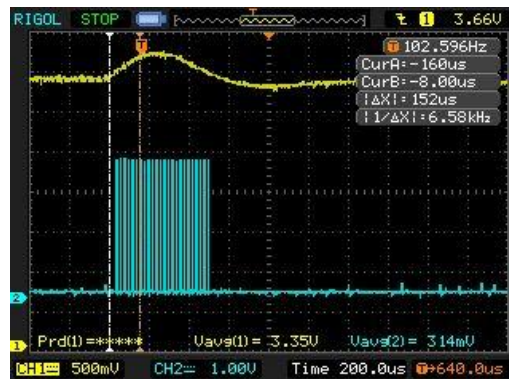


Figure 5-12 Periodical load in dead-band control: two-reference-voltage, dead bands 3.373V to 3.208V, zoom-in version

Appendix C. Waveforms of Dead-Band Control with One Reference Voltages for the
Constant Load

a. Dead-band: 3.373V to 3.208V

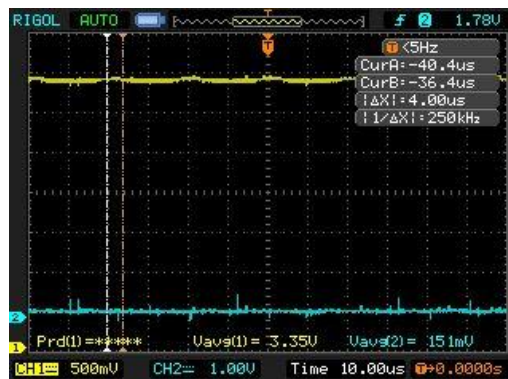


Figure 5-13 Constant load in dead-band control: one-reference-voltage, dead bands 3.373V to 3.208V

b. Dead-band: 3.350V to 3.232V

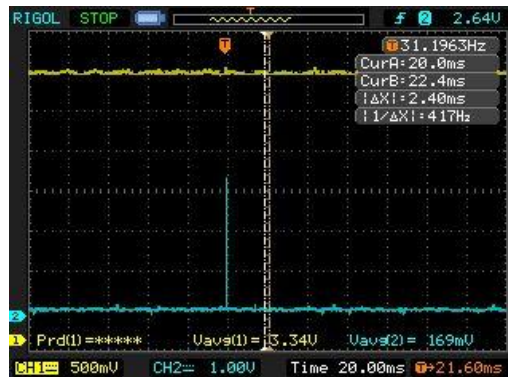


Figure 5-14 Constant load in dead-band control: one-reference-voltage, dead bands 3.350V to 3.232V

c. Dead-band: 3.326V to 3.255V

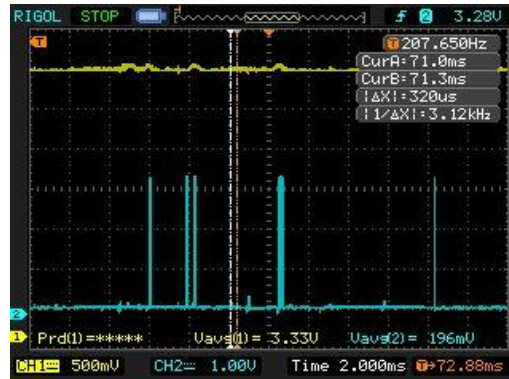


Figure 5-15 Constant load in dead-band control: one-reference-voltage, dead bands 3.326V to 3.255V

Appendix D. Waveforms of Dead-Band Control with Two Reference Voltages for the
Constant Load

a. Dead-band: 3.373V to 3.208V

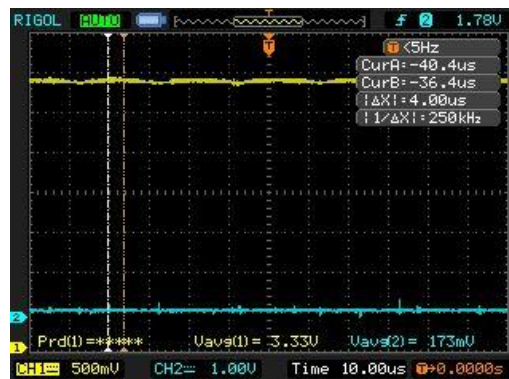


Figure 5-16 Constant load in dead-band control: two-reference-voltage, dead bands 3.373V to 3.208V

b. Dead-band: 3.350V to 3.232V

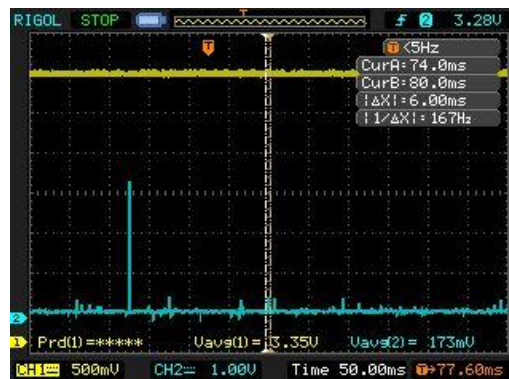


Figure 5-17 Constant load in dead-band control: two-reference-voltage, dead bands 3.350V to 3.232V

c. Dead-band: 3.326V to 3.255V

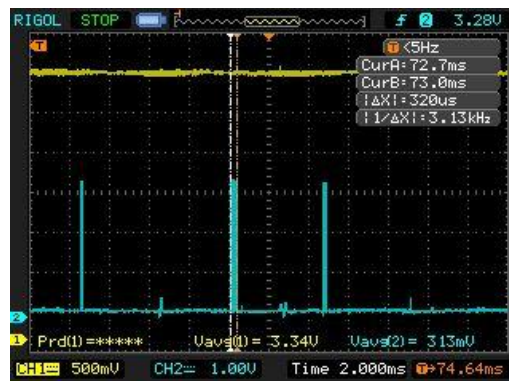


Figure 5-18 Constant load in dead-band control: two-reference-voltage, dead bands 3.326V to 3.255V

Appendix E. Object Code

```

126      __interrupt static void r_adc_interrupt(void)
        r_adc_interrupt:
127      {
128          /* Start user code. Do not edit comment generated here */
129          /* Upper byte of the ADCR register holds the ADC result */
130
131          //signed short delta_TDR01_temp;
132          P4_bit.no1 = 1; // pin 11 on MCU
133          \ 000000 711204          SET1      S:0xFFFF04.1          ;; 2 cycles
134          flag_overflow = 0;
135          \ 000003 F5....          CLRB      N:flag_overflow          ;; 1 cycle
136          \ 000006 61DF          SEL        RB1
137          Cpu_CounterT = TCR10;
138          \ 000008 AFC001          MOVW      AX, 0x1C0          ;; 1 cycle
139          \ 00000B BF....          MOVW      N:Cpu_CounterT, AX          ;; 1 cycle
140          //R_TAU1_Channel10_Start();
141          gADC_Result = ADCR;
142          \ 00000E AD1E          MOVW      AX, S:0xFFFF1E          ;; 1 cycle
143          \ 000010 BF....          MOVW      N:gADC_Result, AX          ;; 1 cycle
144          Voltage_FX = (gADC_Result>>8) + (gADC_Result>>9) +
(gADC_Result>>15);
145          \ 000013 16          MOVW      HL, AX          ;; 1 cycle
146          \ 000014 31FE          SHRW      AX, 0xF          ;; 1 cycle
147          \ 000016 37          XCHW      AX, HL          ;; 1 cycle
148          \ 000017 319E          SHRW      AX, 0x9          ;; 1 cycle
149          \ 000019 14          MOVW      DE, AX          ;; 1 cycle
150          \ 00001A AF....          MOVW      AX, N:gADC_Result          ;; 1 cycle
151          \ 00001D F0          CLRB      X          ;; 1 cycle
152          \ 00001E 08          XCH       A, X          ;; 1 cycle
153          \ 00001F 05          ADDW      AX, DE          ;; 1 cycle
154          \ 000020 07          ADDW      AX, HL          ;; 1 cycle
155          \ 000021 BF....          MOVW      N:Voltage_FX, AX          ;; 1 cycle
156          error_p=error_10_6;
157          \ 000024 AF....          MOVW      AX, N:error_10_6          ;; 1 cycle
158          \ 000027 BF....          MOVW      N:error_p, AX          ;; 1 cycle
159          /*****boost control part*****/
160          Mult_R = ((unsigned long)(TDR01_temp)) << 4; //2_14 to 12_20
161          \ 00002A AF....          MOVW      AX, N:TDR01_temp          ;; 1 cycle
162          \ 00002D 16          MOVW      HL, AX          ;; 1 cycle
163          \ 00002E 31CE          SHRW      AX, 0xC          ;; 1 cycle
164          \ 000030 12          MOVW      BC, AX          ;; 1 cycle
165          \ 000031 17          MOVW      AX, HL          ;; 1 cycle
166          \ 000032 314D          SHLW      AX, 0x4          ;; 1 cycle
167          \ 000034 BF....          MOVW      N:Mult_R, AX          ;; 1 cycle
168          \ 000037 13          MOVW      AX, BC          ;; 1 cycle
169          \ 000038 BF....          MOVW      N:Mult_R+2, AX          ;; 1 cycle
170          //first calculate error_p and K2_boost
171          if(error_flag == 1)
172          \ 00003B E6          ONEW      AX          ;; 1 cycle
173          \ 00003C 42....          CMPW      AX, N:error_flag          ;; 1 cycle
174          \ 00003F DB....          MOVW      BC, N:error_p          ;; 1 cycle
175          \ 000042 AF....          MOVW      AX, N:K2_boost          ;; 1 cycle
176          \ 000045 DF13          BNZ       ??Handle_Switch_0          ;; 4 cycles
177          \ 000047          ; ----- Block:
38 cycles
178          \ 000047          ; * Stack frame (at entry) *
179          \ 000047          ; Param size: 0
180          \ 000047          ; Auto size: 0
181          {
182

```

```

148      Mult_R += K2_boost * error_p;
\ 000047 CEFB01      MULHU                ;; 2 cycles
\ 00004A EB....      MOVW          DE, N:Mult_R        ;; 1 cycle
\ 00004D F7          CLRW          BC                ;; 1 cycle
\ 00004E 05          ADDW          AX, DE          ;; 1 cycle
\ 00004F 61D8        SKNC
\ 000051 A3          INCW          BC                ;; 5 cycles
\ 000052 FB....      MOVW          HL, N:Mult_R+2      ;; 1 cycle
\ 000055 33          XCHW          AX, BC                ;; 1 cycle
\ 000056 07          ADDW          AX, HL          ;; 1 cycle
\ 000057 33          XCHW          AX, BC                ;; 1 cycle
\ 000058 EF0E        BR          S:??Handle_Switch_1 ;; 3 cycles
\ 00005A
17 cycles ; ----- Block:
149      }
150      else
151      Mult_R -= K2_boost * error_p;
      ??Handle_Switch_0:
\ 00005A CEFB01      MULHU                ;; 2 cycles
\ 00005D 14          MOVW          DE, AX                ;; 1 cycle
\ 00005E AF....      MOVW          AX, N:Mult_R        ;; 1 cycle
\ 000061 DB....      MOVW          BC, N:Mult_R+2      ;; 1 cycle
\ 000064 25          SUBW          AX, DE          ;; 1 cycle
\ 000065 61D8        SKNC
\ 000067 B3          DECW          BC                ;; 5 cycles
\ 000068
11 cycles ; ----- Block:
      ??Handle_Switch_1:
\ 000068 BF....      MOVW          N:Mult_R, AX        ;; 1 cycle
\ 00006B 13          MOVW          AX, BC                ;; 1 cycle
\ 00006C BF....      MOVW          N:Mult_R+2, AX      ;; 1 cycle
152
153      if(One_vref){
\ 00006F D5....      CMP0          N:One_vref        ;; 1 cycle
\ 000072 DD5C        BZ          ??Handle_Switch_2    ;; 4 cycles
\ 000074
8 cycles ; ----- Block:
154      //then calculate error_10_6 and K1 boost
155      if(vref <= voltage_FX)
\ 000074 FB....      MOVW          HL, N:vref        ;; 1 cycle
\ 000077 AF....      MOVW          AX, N:voltage_FX   ;; 1 cycle
\ 00007A 47          CMPW          AX, HL          ;; 1 cycle
\ 00007B DC21        BC          ??Handle_Switch_3    ;; 4 cycles
\ 00007D
7 cycles ; ----- Block:
156      {
157      error_flag = 1;
\ 00007D E6          ONEW          AX                ;; 1 cycle
\ 00007E BF....      MOVW          N:error_flag, AX  ;; 1 cycle
158      error_10_6 = voltage_FX - Vref;
\ 000081 AF....      MOVW          AX, N:voltage_FX   ;; 1 cycle
\ 000084 22....      SUBW          AX, N:vref        ;; 1 cycle
\ 000087 BF....      MOVW          N:error_10_6, AX  ;; 1 cycle
159      Mult_R -= K1_boost * error_10_6;
\ 00008A 12          MOVW          BC, AX                ;; 1 cycle
\ 00008B AF....      MOVW          AX, N:K1_boost     ;; 1 cycle
\ 00008E CEFB01      MULHU                ;; 2 cycles
\ 000091 14          MOVW          DE, AX                ;; 1 cycle
\ 000092 AF....      MOVW          AX, N:Mult_R        ;; 1 cycle
\ 000095 DB....      MOVW          BC, N:Mult_R+2      ;; 1 cycle
\ 000098 25          SUBW          AX, DE          ;; 1 cycle
\ 000099 61D8        SKNC
\ 00009B B3          DECW          BC                ;; 5 cycles
\ 00009C EF2B        BR          S:??Handle_Switch_4    ;; 3 cycles

```

```

\      00009E          ; ----- Block:
21 cycles
160
161
\      00009E FB....
\      0000A1 AF....
\      0000A4 47
\      0000A5 DC29
\      0000A7
7 cycles
162
163
\      0000A7 F6
\      0000A8 BF....
164
\      0000AB AF....
\      0000AE 22....
\      0000B1 BF....
165
\      0000B4 12
\      0000B5 AF....
\      0000B8 CEFB01
\      0000BB EB....
\      0000BE F7
\      0000BF 05
\      0000C0 61D8
\      0000C2 A3
\      0000C3 FB....
\      0000C6 33
\      0000C7 07
\      0000C8 33
\      0000C9
21 cycles
\      0000C9 BF....
\      0000CC 13
\      0000CD BF....
\      0000D0
3 cycles
166
167
168
169
\      0000D0 D5....
\      0000D3 DD5C
\      0000D5
5 cycles
170
171
\      0000D5 FB....
\      0000D8 AF....
\      0000DB 47
\      0000DC DC21
\      0000DE
7 cycles
172
173
\      0000DE E6
\      0000DF BF....
174
\      0000E2 AF....
\      0000E5 22....
\      0000E8 BF....

; ----- Block:
}
else if(vref >= voltage_FX)
??Handle_Switch_3:
MOVW    HL, N:Voltage_FX    ;; 1 cycle
MOVW    AX, N:vref          ;; 1 cycle
CMPW    AX, HL              ;; 1 cycle
BC      ??Handle_Switch_2  ;; 4 cycles
; ----- Block:

{
error_flag = 0;
CLRW    AX                  ;; 1 cycle
MOVW    N:error_flag, AX   ;; 1 cycle
error_10_6 = vref - voltage_FX;
MOVW    AX, N:vref         ;; 1 cycle
SUBW    AX, N:Voltage_FX  ;; 1 cycle
MOVW    N:error_10_6, AX  ;; 1 cycle
Mult_R += K1_boost * error_10_6;
MOVW    BC, AX             ;; 1 cycle
MOVW    AX, N:K1_boost     ;; 1 cycle
MULHU   ;; 2 cycles
MOVW    DE, N:Mult_R       ;; 1 cycle
CLRW    BC                 ;; 1 cycle
ADDW    AX, DE             ;; 1 cycle
SKNC    ;; 1 cycle
INCW    BC                 ;; 5 cycles
MOVW    HL, N:Mult_R+2    ;; 1 cycle
XCHW    AX, BC            ;; 1 cycle
ADDW    AX, HL             ;; 1 cycle
XCHW    AX, BC            ;; 1 cycle
; ----- Block:

??Handle_Switch_4:
MOVW    N:Mult_R, AX       ;; 1 cycle
MOVW    AX, BC             ;; 1 cycle
MOVW    N:Mult_R+2, AX    ;; 1 cycle
; ----- Block:

}
}
if(Two_vref){
??Handle_Switch_2:
CMP0    N:Two_vref        ;; 1 cycle
BZ      ??Handle_Switch_5  ;; 4 cycles
; ----- Block:

//then calculate error_10_6 and K1 boosst
if(vrefu <= voltage_FX)
MOVW    HL, N:vrefu       ;; 1 cycle
MOVW    AX, N:Voltage_FX  ;; 1 cycle
CMPW    AX, HL            ;; 1 cycle
BC      ??Handle_Switch_6  ;; 4 cycles
; ----- Block:

{
error_flag = 1;
ONEW    AX                 ;; 1 cycle
MOVW    N:error_flag, AX  ;; 1 cycle
error_10_6 = voltage_FX - vrefu;
MOVW    AX, N:Voltage_FX  ;; 1 cycle
SUBW    AX, N:vrefu       ;; 1 cycle
MOVW    N:error_10_6, AX  ;; 1 cycle
}
}

```

```

175      Mult_R -= K1_boost * error_10_6;
\      0000EB 12      MOVW      BC, AX          ;; 1 cycle
\      0000EC AF....  MOVW      AX, N:K1_boost   ;; 1 cycle
\      0000EF CEFB01  MULHU     ;; 2 cycles
\      0000F2 14      MOVW      DE, AX          ;; 1 cycle
\      0000F3 AF....  MOVW      AX, N:Mult_R     ;; 1 cycle
\      0000F6 DB....  MOVW      BC, N:Mult_R+2  ;; 1 cycle
\      0000F9 25      SUBW      AX, DE          ;; 1 cycle
\      0000FA 61D8    SKNC
\      0000FC B3      DECW      BC              ;; 5 cycles
\      0000FD EF2B    BR        S:??Handle_Switch_7 ;; 3 cycles
\      0000FF
21 cycles ; ----- Block:
176      }
177      else if(vrefl >= voltage_FX)
      ??Handle_Switch_6:
\      0000FF FB....  MOVW      HL, N:Voltage_FX ;; 1 cycle
\      000102 AF....  MOVW      AX, N:vrefl     ;; 1 cycle
\      000105 47      CMPW      AX, HL          ;; 1 cycle
\      000106 DC29    BC        ??Handle_Switch_5 ;; 4 cycles
\      000108
7 cycles ; ----- Block:
178      {
179      error_flag = 0;
\      000108 F6      CLRW      AX              ;; 1 cycle
\      000109 BF....  MOVW      N:error_flag, AX ;; 1 cycle
180      error_10_6 = vrefl - Voltage_FX;
\      00010C AF....  MOVW      AX, N:vrefl     ;; 1 cycle
\      00010F 22....  SUBW      AX, N:Voltage_FX ;; 1 cycle
\      000112 BF....  MOVW      N:error_10_6, AX ;; 1 cycle
181      Mult_R += K1_boost * error_10_6;
\      000115 12      MOVW      BC, AX          ;; 1 cycle
\      000116 AF....  MOVW      AX, N:K1_boost   ;; 1 cycle
\      000119 CEFB01  MULHU     ;; 2 cycles
\      00011C EB....  MOVW      DE, N:Mult_R     ;; 1 cycle
\      00011F F7      CLRW      BC              ;; 1 cycle
\      000120 05      ADDW      AX, DE          ;; 1 cycle
\      000121 61D8    SKNC
\      000123 A3      INCW      BC              ;; 5 cycles
\      000124 FB....  MOVW      HL, N:Mult_R+2  ;; 1 cycle
\      000127 33      XCHW      AX, BC          ;; 1 cycle
\      000128 07      ADDW      AX, HL          ;; 1 cycle
\      000129 33      XCHW      AX, BC          ;; 1 cycle
\      00012A
21 cycles ; ----- Block:
\      00012A BF....  ??Handle_Switch_7:
\      00012D 13      MOVW      N:Mult_R, AX    ;; 1 cycle
\      00012E BF....  MOVW      AX, BC          ;; 1 cycle
\      000131
3 cycles ; ----- Block:
182      }
183      }
184      }
185      // normalize MAC accumulator from 12-20 down to 2-14
186      TDR01_temp = (unsigned short)(Mult_R >> 4);
\      000131 AF....  ??Handle_Switch_5:
\      000134 DB....  MOVW      AX, N:Mult_R     ;; 1 cycle
\      000137 314E    MOVW      BC, N:Mult_R+2  ;; 1 cycle
\      000139 33      SHRW      AX, 0x4         ;; 1 cycle
\      00013A 31CD    XCHW      AX, BC          ;; 1 cycle
\      00013C 03      SHLW      AX, 0xC         ;; 1 cycle
\      00013C 03      ADDW      AX, BC          ;; 1 cycle
187      /*****duty cycle restraints*****/
188      #if 1

```



```

\ 000179 BF.... MOVW N:Cpu_CounterB, AX ;; 1 cycle
204 P4_bit.no1 = 0; // pin 11 on MCU
\ 00017C 711304 CLR1 S:0xFFF04.1 ;; 2 cycles
205 Cpu_Counter += flag_overflow*319+Cpu_CounterT-Cpu_CounterB;
      ^
warning[Pa082]: undefined behavior: the order of volatile accesses is
                undefined in this statement
\ 00017F 8F.... MOV A, N:flag_overflow ;; 1 cycle
\ 000182 70 MOV X, A ;; 1 cycle
\ 000183 F1 CLRB A ;; 1 cycle
\ 000184 323F01 MOVW BC, #0x13F ;; 1 cycle
\ 000187 CEFB01 MULHU ;; 2 cycles
\ 00018A 02.... ADDW AX, N:Cpu_CounterT ;; 1 cycle
\ 00018D FB.... MOVW HL, N:Cpu_Counter+2 ;; 1 cycle
\ 000190 22.... SUBW AX, N:Cpu_CounterB ;; 1 cycle
\ 000193 EB.... MOVW DE, N:Cpu_Counter ;; 1 cycle
\ 000196 F7 CLRW BC ;; 1 cycle
\ 000197 05 ADDW AX, DE ;; 1 cycle
\ 000198 61D8 SKNC
\ 00019A A3 INCW BC ;; 5 cycles
\ 00019B 33 XCHW AX, BC ;; 1 cycle
\ 00019C 07 ADDW AX, HL ;; 1 cycle
\ 00019D 33 XCHW AX, BC ;; 1 cycle
\ 00019E BF.... MOVW N:Cpu_Counter, AX ;; 1 cycle
\ 0001A1 33 XCHW AX, BC ;; 1 cycle
\ 0001A2 BF.... MOVW N:Cpu_Counter+2, AX ;; 1 cycle
\ 0001A5 33 XCHW AX, BC ;; 1 cycle
206 flag_overflow = 0;
\ 0001A6 F5.... CLRB N:flag_overflow ;; 1 cycle
207 /* End user code. Do not edit comment generated here */
208 }
\ 0001A9 61FC RETI ;; 6 cycles
\ 0001AB ; ----- Block:
35 cycles
\ 0001AB ; ----- Total:
278 cycles

```