Autonomous system clock frequency controller for pico- and nano- satellites

Ian Kamajaya

2015

Ian Kamajaya. (2015). Autonomous system clock frequency controller for pico- and nano-satellites. Master's thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/62573>

<https://doi.org/10.32657/10356/62573>

AUTONOMOUS SYSTEM CLOCK FREQUENCY CONTROLLER FOR PICO- AND NANO- SATELLITES

IAN KAMAJAYA

**SCHOOL OF ELECTRICAL AND ELECTRONIC
ENGINEERING**

A dissertation submitted to the Nanyang Technological
University

In partial fulfillment of the requirement for the degree of

Master of Engineering

April 2015

Acknowledgments

*Glory be to God the Trinity,
who has made Himself known through His creations,
so that man may learn to love and to revere Him.*

*The heavens declare the glory of God;
the skies proclaim the work of His hands.
Day after day they pour forth speech;
night after night they display knowledge.
There is no speech or language
where their voice is not heard.
Their voice goes out into all the earth,
their words to the ends of the world. (Psalm 19: 1-4)*

*When I consider Your heavens,
the work of Your fingers,
the moon and the stars,
which You have set in place,
what is man that You are mindful of him,
the son of man that You care for him?
You made him a little lower than heavenly beings
and crowned him with glory and honor.
You made him ruler over the works of Your hands;
You put everything under his feet (Psalm 8: 3-6)*

Let everything that has breath praise the LORD. (Psalm 150: 6)

Acknowledgments

It is with great pleasure that I write the acknowledgments for my thesis to express my heartfelt thanks to those who have rendered support and assistance to the completion of my thesis in one way or another.

Firstly, I would like to thank A/P Low Kay Soon for giving me an opportunity to work in the Satellite Research Centre (SaRC) on the development of the on-board data handling (OBDH) system for satellites. The work he hands over to me provides me with great opportunity to learn and to refine my skills needed to complete my research thesis. His guidance and support throughout the past three years are also instrumental to the completion of my thesis.

Secondly, I would like to thank my three colleagues, Mr. Xie Shuang Long, Mr. Jan Hakenberg, and Mr. Lee Guo Xiong for their assistance in reviewing my work. I appreciate their time and efforts they have given me despite their busy schedule.

Next, I would like to thank all the other colleagues, managers, technical staffs, and Final Year Project (FYP) students in the SaRC, who have given me their support one way or another for the completion of my work. Without their help, the completion of my work would be impossible.

Last, but not least, I would like to express my sincerest gratitude to my girlfriend, Ms. Livia Sutrisno, who has been patiently supporting me throughout my work. Though her support is nothing technical (on the contrary, I even find it hard times to explain what I am doing in the project to her!), but her encouragement and patience has helped me a lot to continue to work on my research with a good spirit in spite of difficult times. You have my utmost gratitude, Livia!

Table of Contents

Acknowledgments	1
Table of Contents	3
List of Figures	6
List of Tables	7
Summary.....	8
1. Introduction.....	10
1.1. Background.....	10
1.1.1. Classification of Satellites	10
1.1.2. Power on Miniaturized Satellites	11
1.1.3. System Clock on Miniaturized Satellite's Applications.....	13
1.1.3.1. Fixed System Clock Frequency Method.....	14
1.1.3.2. Mode-Based System Clock Frequency Method.....	14
1.2. Objective.....	15
1.3. Contributions.....	16
1.4. Outline of Thesis	16
2. Literature Review.....	18
2.1. Introduction.....	18
2.2. Improving Input Power	18
2.2.1. Deployable Solar Panels	18
2.2.2. Optimization of Power Efficiency.....	19
2.2.3. Optimization of Orbit Path	19
2.3. Reducing Power Consumption of Satellite.....	20
2.4. Implementation Issues on Low Speed Microcontrollers.....	24
2.5. Proposed Autonomous System Clock Frequency Controller.....	25
2.6. Summary.....	25
3. System Modeling.....	27
3.1. Introduction.....	27
3.2. Task's Execution Time Model	27
3.2.1. Single Execution Time Model.....	29
3.2.2. Foreground Tasks' Execution Time Model.....	29
3.2.3. Background Tasks' Execution Time Model.....	30
3.2.3.1. Event-Dependent Background Tasks' Execution Time Model.....	30
3.2.3.2. Time-Dependent Background Tasks' Execution Time Model.....	31
3.2.3.3. Background Tasks' Worst Case Execution Time (WCET) Scenario	33
3.2.3.4. Current Background Tasks' Actual Execution Time Model.....	34
3.2.3.5. Left-Over Background Tasks' Actual Execution Time Model	35
3.2.4. Worst Case Execution Time (WCET) Model.....	35

3.2.4.1.	Total Execution Time Model without Leftover Background Tasks' Job	35
3.2.4.2.	System Clock Frequency Requirement Model without Leftover Background Tasks' Job	37
3.2.4.3.	WCET and System Clock Frequency Requirement Model with Leftover Background Tasks' Job	38
3.2.5.	Execution Time Model Error	38
3.2.5.1.	Hard-Deadline System	39
3.2.5.2.	Soft-Deadline System	39
3.2.6.	Execution Time Margin and Guaranteed System Time Performance	39
3.2.6.1.	Hard-Deadline System	39
3.2.6.2.	Soft-Deadline System	40
3.3.	Energy Consumption Model	40
3.4.	Conclusions	43
4.	Design and Implementation of an Autonomous System Clock Frequency Controller	44
4.1.	Introduction	44
4.2.	Design	44
4.2.1.	Flight Software	45
4.2.2.	Time Estimator	46
4.2.2.1.	Time Estimator Optimization	46
4.2.2.2.	Off-line Static Parameters Computation	47
4.2.2.3.	Conditional Truncation and Scaling	47
4.2.2.4.	Completely-Dependent Tasks Identifiers	47
4.2.3.	Energy Estimator	47
4.2.4.	Frequency Controller	48
4.2.5.	Summary of the Design Steps	49
4.3.	Implementation	49
4.3.1.	Determining System Clock Frequencies	50
4.3.2.	Energy Estimator	52
4.3.3.	Time Estimator	55
4.3.3.1.	Obtaining Time Estimator Parameters	56
4.3.3.2.	Optimization of the Time Estimator Parameters	58
4.4.	Conclusions	59
5.	Experimental Results	60
5.1.	Experimental Settings	60
5.2.	Benchmarking	62
5.3.	Time Performance	64
5.4.	Frequency Controller Responses	67
5.5.	Energy Consumption	69
6.	Conclusions and Future Works	75

6.1. Conclusions	75
6.2. Future Works.....	75
Bibliography.....	77
Appendix 1 – Peripheral Error Rates Analysis	81
<i>Universal Asynchronous Receiver Transmitter 0 (UART0)</i>	81
<i>Universal Asynchronous Receiver Transmitter 1 (UART1)</i>	81
<i>Serial Peripheral Interface 0 (SPI0)</i>	82
<i>Serial Management Bus (SMBus)</i>	83
<i>Analog to Digital Converter 0 (ADC0)</i>	83
<i>Timer 2 (TMR2)</i>	84
Appendix 2 – Time Estimator’s Parameters’ Model Error Extra Statistical and Graphical Results	85
<i>Extra Statistical Result</i>	85
<i>Graphical Results</i>	87
Foreground Tasks	87
Event-Dependent Background Tasks.....	92
Time-Dependent Background Tasks.....	94
Appendix 3 – Experiment’s Extra Statistical and Graphical Results	95
<i>Extra Statistical Results</i>	95
<i>Graphical Results</i>	96
Orbit 1	96
Orbit 2	98
Orbit 3	99
Orbit 4	101
Orbit 5	102

List of Figures

Figure 1-1: VELOX-P11, A 1U CubeSat Pico-Satellite Built by Nanyang Technological University (NTU).....	11
Figure 1-2: Delfi-C3, a Triple (3U) CubeSat Nano-Satellite developed by Delft University of Technology (TU Delft).....	12
Figure 1-3: QuakeSat, a Triple (3U) CubeSat Developed by Stanford University, QuakeFinder, and California Polytechnic State University.....	13
Figure 1-4: STUDSAT-1 (1U), India's first student-built Pico-Satellite.....	13
Figure 2-1: XSAS shown attached to 2U CubeSat in a) 1U stowed configuration and b) fully deployed configuration [17]	18
Figure 2-2: Screenshot of Jtrack Satellite Orbit [31]	21
Figure 4-1: System Clock Frequency Controller System Block Diagram	44
Figure 4-2: Time Estimator Block Diagram	46
Figure 4-3: Original vs 1st Order Least-Squares Fitting for System Clock vs Motherboard's Current	54
Figure 4-4: Error Graph and Error Graph Percentage for Motherboard's Current Model	54
Figure 5-1: System Time Performance Graph (First 5 Minutes) for Orbital Scenario 1	65
Figure 5-2: System Time Performance Graph (First 5 Minutes) for Orbital Scenario 5	65
Figure 5-3: Frequency Controller Response Graph (First 5 Minutes) for Orbital Scenario 1.....	68
Figure 5-4: Frequency Controller Response Graph (First 5 Minutes) for Orbital Scenario 5.....	68
Figure 5-5: The Comparison of the Motherboard's Average Current Drawn per Second for the First 5 Minutes of the Orbital Scenario 1 for Different Methods.....	69
Figure 5-6: The Comparison of the Motherboard's Average Current Drawn per Second for the First 5 Minutes of the Orbital Scenario 5 for Different Methods.....	70
Figure 5-7: Energy Consumption (in mWh) of the Proposed Controller Compared to the Fixed-Method, Mode-Based-Method, and Optimal System Clock Frequency in Various Orbital Scenarios.....	72
Figure 5-8: Energy Reduced (in %) of the Proposed Controller Compared to the Fixed-Method, Mode-Based-Method, and the Use of Optimal System Clock Frequency in Various Orbital Scenarios with respect to the Fixed-Method.....	72
Figure 5-9: Average Power Reduced (in mW) of the Proposed Controller Compared to the Fixed-Method, Mode-Based-Method, and the Use of Optimal System Clock Frequency in Various Orbital Scenarios with respect to the Fixed-Method	73

List of Tables

<i>Table 1-1: Satellite s' Classification Based on Their Mass [3]</i>	10
<i>Table 1-2: Existing Approaches on Microcontroller's System Clock Frequency in Miniaturized Satellite Applications</i>	15
<i>Table 2-1: Different Low Earth Orbit (LEO) Parameters [30]</i>	20
<i>Table 2-2: Different Beta Angle Range based on Orbital Inclination [30]</i>	20
<i>Table 2-3: Example of Characteristics of the System per Region [31]</i>	21
<i>Table 2-4: Summary of CPU's Energy Reduction Techniques</i>	24
<i>Table 3-1: Examples of Completely Dependent and Partially Dependent Tasks</i>	28
<i>Table 4-1: Peripheral Rate Requirements for VELOX-I Flight Software</i>	50
<i>Table 4-2: System Clock Frequency Level, System Clock Frequency, and System Clock Period</i>	51
<i>Table 4-3: Range of Peripheral Error Rates in VELOX-I Flight Software after the Implementation of Seven Levels of System Clock Frequencies</i>	51
<i>Table 4-4: Motherboard's Average Current at Different System Clock Frequency Level</i>	52
<i>Table 4-5: VELOX-I's Task List</i>	55
<i>Table 4-6: Motherboard's Tasks' Time Estimator's Coefficients</i>	57
<i>Table 4-7: Motherboard's Time Estimator's C Matrix</i>	58
<i>Table 5-1: Simulated Orbital Scenarios</i>	60
<i>Table 5-2: Simulated Tasks on Different Loads</i>	62
<i>Table 5-3: Comparison of Fixed Method, Mode Based Method, and the Proposed Controller</i>	63
<i>Table 5-4: Microcontroller System Clock Frequency Level Applied for the Various Orbital Scenarios by Fixed-Method, Mode-Based Method, and the Proposed System Clock Frequency Controller</i>	64
<i>Table 5-5: Proposed Controller System's Time Performance Statistics</i>	66
<i>Table 5-6: Energy/Power Experimental Data of the Proposed Controller Compared to the Fixed-Method, Mode-Based-Method, and the Use of Optimal System Clock Frequency in Various Orbital Scenarios</i>	71
<i>Table 5-7: Energy Reduction Percentage Difference between the Mode-Based Method, the Proposed Controller, and the Optimal System Clock Frequency</i>	73

Summary

Nano-satellites and pico-satellites are miniaturized satellites weighing less than 10 kg and 1 kg respectively. Over the last decade, there has been growing trend to build miniaturized satellites for research and technology demonstration. One of the major concerns in any satellite system is the power consumption. The problem is more apparent in the miniaturized satellites as their limited physical size restricts the area available for the solar panels. There have been various research works to resolve this power limitation issue. But one area remains to be more extensively researched is the optimization of micro-controller's power consumption.

To reduce the power consumption of a Central Processing Unit (CPU), techniques such as the Dynamic Voltage Scaling (DVS) and Dynamic Voltage and Frequency Scaling (DVFS) are popularly used in computer system. DVS and DVFS rely on special hardware components, namely, a programmable DC-DC switching voltage regulator and a high performance (≥ 600 MHz) processor (to ensure that the scheduling and pre-emption overheads in the algorithm are negligible) with a wide operating frequency range. While those components are available in most of the modern computers, they are not suitable in the miniaturized satellite applications, which are often controlled by low-speed (≤ 100 MHz) micro-controllers and having only a programmable clock generator to generate its system clock frequency.

Furthermore, satellite components are typically of older technology (by 20 to 30 years) than its earth-use counterpart due to the extremely expensive rocket cost and its scarce market. This is to say that the modern computer-level microprocessors, apart from being consuming higher energy level due to its higher system clock frequency (by law of physics), would likely have a long way to go to the space due to the satellite components' limited market.

Hence, this work fills the gap between the research in the computer science and the realization in the miniaturized satellites. By introducing an autonomous system clock frequency controller which only depends on a programmable system clock generator, this research work extends the research on power optimization of the high speed micro-processor to the low-speed micro-controllers, making it suitable for miniaturized satellite applications.

The controller introduced by this work is shown to be able to dynamically adjust the system clock frequency of an example nano-satellite, VELOX-I, according to its current tasks and under a user-specified time constraint, regardless of the satellite operating mode. It makes use of time estimator and energy estimator modules to find the optimal operating frequency under a given time constraint and instance. Tested under various orbital scenarios, the system using the controller is shown to consume less energy as compared to the same system that uses conventional methods, namely the *fixed system clock frequency method* and *mode-based system clock frequency method*, by an average of 42.44% and 19.137% respectively.

Putting it in the miniaturized satellite context, the reduction in energy consumption using the proposed controller could change the energy balance from negative to positive. For example such as VELOX-II, its input power budget is only 1050 mW. However, the energy consumption for the microcontrollers used in its four main subsystems (On-Board Data Handling, Power Supply System, Communication System, and Attitude Determination and Control System) is 1140 mW with fixed system clock frequency method.

It is possible to have later generation of the miniaturized satellites to use more advance CPU. However, since the energy consumption and energy obtained are tied to the laws of physics (higher frequency, higher energy consumption. Larger solar panel area, larger energy harvested), when the time comes, there will likely be smaller satellite class (like femto-satellites) where the algorithm developed would still be relevant to be applied. In this thesis, how the algorithm may benefit the current generation of miniaturized satellites (pico- and nano- satellites) is demonstrated.

1. Introduction

1.1. Background

Since the successful launch of the first artificial satellite, Sputnik 1, to the space in 1957, there has been significant progress in the satellite technology. For decades, the trend has been to build larger satellites that can carry equipment which are ever more impressive.

Starting with a beach-ball sized Sputnik 1, in four decades, humans had built the International Space Station (ISS) whose size and weight are $72.8 \times 108.5 \times 20 \text{ m}^3$ and 450,000 kg respectively – making ISS the largest artificial satellite that has ever come into existence.

Over the last decade, however, the reverse trend of building smaller satellites has also become a subject of research interest [1]. In contrast to large satellites, small satellites provide opportunities to enable missions that larger satellite could not accomplish. For example, using constellations for low data rate communications, performing formations to gather data from multiple points, or in-orbit inspection of larger satellites [2].

1.1.1. Classification of Satellites

Satellite is typically classified based on its mass, which is proportional to the launch cost of the satellite. Table 1-1 shows the satellites' classification based on the mass [3]

Table 1-1: Satellite s' Classification Based on Their Mass [3]

No	Class	Mass Range
1	Large Satellite	More than 1000 kg
1	Medium Satellite	500 kg to 1000 kg
2	Mini Satellite	100 kg to 500 kg
3	Micro Satellite	10 kg to 100 kg
4	Nano Satellite	1 kg to 10 kg
5	Pico Satellite	0.1 kg to 1 kg
6	Femto Satellite	Less than 0.1 kg

From Table 1-1, pico-satellite belongs to a class of small satellites with a mass ranging from 0.1 kg to 1.33 kg, while a nano-satellite is a small satellite with a mass ranging from 1.33 kg to 20 kg. CubeSats, or cube satellites, are small satellites with a standardized size, weight, and shape. 1U (one unit) CubeSat is a cube-shaped satellite with 10 cm x 10 cm x 10 cm volume and 1 to 1.33 kg mass [4]. The CubeSat standard is primarily designed for pico- and nano-satellites classes. To distinguish pico- and nano-satellite classes from the rests, the term *miniaturized satellites* will be used in the remaining of this thesis referring to them.

Figure 1-1: VELOX-P11, A 1U CubeSat Pico-Satellite Built by Nanyang Technological University (NTU)

Figure 1-1 shows an example of a pico-satellite named VELOX-P11 following the CubeSat standard built by the Satellite Research Centre (SaRC) in Nanyang Technological University (NTU). The pico-satellite was successfully launched on 21st November 2013 by the Dnepr rocket from Yasny, Russia [5].

1.1.2. Power on Miniaturized Satellites

One of the major concerns in any satellite system is the power balance [6]. The problem is more apparent in the miniaturized satellites as their extremely limited physical size restricts the area available for the solar panels.

Figure 1-2: Delfi-C3, a Triple (3U) CubeSat Nano-Satellite developed by Delft University of Technology (TU Delft)

Typically the available input power of the 1U, 2U, and 3U CubeSats' range from 1 to 2.5 Watts, 2 to 5 Watts, and 7 to 20 Watts, respectively [7-11]. In some cases, the available input power may significantly vary. For example, CP1 is a 1U pico-satellite that has an input power budget of 726 mW to support its full operation [12]. Another CubeSat named Delfi-C3 (shown in Figure 1-2), a 3U CubeSat batteryless nano-satellite, only has a guaranteed input power of 2.4 W for its operation [13].

In response to the extreme power limitation, two common approaches have been used in the satellite community. One is to improve the input power and the other is to reduce the power usage without reducing the satellite performance. There were some research works aimed to improve the input power to the satellites through deployable solar panels [14-18], optimization of power gain [19-29], and optimization of the satellite's orbit path [30]. Moreover, there were also some research works aimed to reduce the use of energy in various aspects of the satellite such as its flying region [31], communication link [32], as well as computational algorithm [33, 34].

In the computer science, there are also some works in reducing energy consumption for the general high-speed microprocessors (typically ≥ 600 MHz) [35-45]. Popular techniques include Dynamic Voltage Scaling (DVS) [38, 44], Dynamic Voltage and Frequency Scaling (DVFS) [37, 38, 44, 46, 47], Dynamic Power Management (DPM) [35, 43], Feedback Control Scheduling (FCS) [36], as well as power-aware task scheduling [36, 39-42, 44, 46-48].

1.1.3. System Clock on Miniaturized Satellite's Applications

Figure 1-3: QuakeSat, a Triple (3U) CubeSat Developed by Stanford University, QuakeFinder, and California Polytechnic State University

Figure 1-4: STUDSAT-1 (1U), India's first student-built Pico-Satellite

The energy consumed by the microcontrollers in miniaturized satellites cannot be overlooked. It consumes significant amount of the overall energy budget. QuakeSat shown in Figure 1-3 is a 3U nano-satellite with an average input power budget of 12.6 W. However, 2.5 W power budget is used by its main CPU which runs at 66 MHz all the time [49]. That amount of power is slightly higher than 2.4 W input power budget for the full operation of Delfi-C3 [13]. It is more than twice of the power budget of STUDSAT, India's first student 1U CubeSat (Figure 1-4), which is 1053 mW [9]. Another satellite, VELOX-P11, has an input power budget of 1050 mW. But it requires 640 mW of its power to keep its 3 out of 4 main microcontrollers (On-Board Computer subsystem, Power Supply Subsystem, and Communication subsystem) running all the time. When the satellite turns on its transmitter (consumes 700-800 mW) or its Attitude and Determination Control Subsystem (consumes 500 mW), its power budget will become

negative. Thus, the energy consumed by the microcontrollers in miniaturized satellites can be very significant.

1.1.3.1. *Fixed System Clock Frequency Method*

Today, many microcontrollers could support a wide range of system clock frequencies [50-65]. As the energy consumed by a microcontroller is proportional to its system clock frequency, it is desired to operate the clock at a lowest possible rate.

The most common approach to use a microcontroller in the miniaturized satellites is to fix one system clock frequency for the application [12, 66-69]. This approach is called *fixed system clock frequency method*. It is the simplest approach as one does not need to formulate and adjust various peripherals' clock settings of the microcontroller. This approach is non-autonomous since it cannot change its system clock frequency based on the state of the satellite.

The major drawback of this approach is its tendency to overclock the system requirements, which leads to inefficient power use. In miniaturized satellite applications, where 1 W of power could be the entire power budget of the system, the power inefficiency introduced by this mode possess substantial risks. Examples of overclock situation could be found in QuakeSat and VELOX-P11 initial development. QuakeSat reduced its microcontroller's system clock frequency from 100 MHz to 66 MHz to reduce its power consumption in the later phase of the development [49]. Similarly, VELOX-P11, a 1U pico-satellite reduced its on-board computer system clock frequency from 99.5328 MHz to 22.1184 MHz and power supply system's operating frequency from 49 MHz to 28 MHz to achieve positive power balance requirement (while having its Attitude Determination and Control Subsystem turned off most of the time).

1.1.3.2. *Mode-Based System Clock Frequency Method*

Another approach to choose the system clock frequency is to pre-determine it based on operation modes [11, 70, 71]. This is called *mode-based system clock frequency method*. This method partially optimizes the energy consumption. General implementation of this method allows the microcontroller to be operated at slower system clock frequency while the system is idling but reverted to normal system clock frequency while the system is active.

This method is commonly used in consumer electronics, but less popularly used in the miniaturized satellite applications. In consumer electronics, there is typically a standardized real-time operating system which allows the user to select the operating modes that use less power (such as Sleep Mode or Hibernate Mode for Windows PC) upon user selection or automatic detection. In the miniaturized satellites which are mostly micro-controller based, it lacks of a standard real-time operating system to allow the “user” to choose the “operating modes” that their hardware could execute. Even the notion of “operating modes” themselves are not standardized, but highly application dependent. This significant effort required to implement the *mode-based method* makes it a less popular choice.

Table 1-2 summaries of the existing approaches on using the microcontroller’s system clock frequencies in miniaturized satellite applications.

Table 1-2: Existing Approaches on Microcontroller’s System Clock Frequency in Miniaturized Satellite Applications

No	Method	Strengths	Weaknesses
1	Fixed Method [12, 66-69]	<ul style="list-style-type: none"> • Easy to implement 	<ul style="list-style-type: none"> • Tend to overclock the system requirements • Non-optimized • Non-autonomous
2	Mode-based Method [11, 70, 71]	<ul style="list-style-type: none"> • Partially optimized • Partially autonomous 	<ul style="list-style-type: none"> • Highly application dependent • Not optimal solution • Hard to determine the system clock frequencies of the modes

1.2. Objective

In this research, an autonomous, energy efficient, task-based approach to control the microcontroller's system clock frequency in miniaturized satellite applications will be investigated. The research exploits a typical microcontroller inherent design which provides a wide range of system clock frequencies to create an autonomous system clock frequency controller. The controller will minimize the energy consumption and maintain the user-defined system time performance based on the estimated state variables and the present satellite state.

In addition to achieve energy efficient design, the developed system is also aimed for autonomy. It allows the system to intelligibly respond to the tasks' dynamics with the optimal operating frequency while maintaining system stability through fixed peripheral rates.

In this research, a specific nano-satellite named VELOX-I, is chosen for a case study. The satellite is built by NTU SaRC and is successfully launched on 30th June 2014 [5]. To measure its energy efficiency, the proposed system is benchmarked with two existing approaches, i.e. the fixed operating frequency method and the mode-based operating frequency method.

1.3. Contributions

The main contributions of this research are as follows:

- Study the effects of the microcontroller's system clock frequency on the energy consumption in low power miniaturized satellite applications.
- Develop a suitable time estimator model for microcontroller-based miniaturized satellite applications.
- Design an autonomous system clock frequency controller to operate the microcontroller in its optimal system clock frequency based on the tasks' dynamics and system time requirements. The controller minimizes the microcontroller's energy consumption while maintains a guaranteed system time performance.
- Evaluate the proposed system clock frequency controller's energy and time performance with an experimental setup.

1.4. Outline of Thesis

This thesis is divided into seven chapters. Chapter 2 first presents a literature review on works to resolve the power limitation issue of miniaturized satellites. It then reviews the various research works in computer science to reduce energy consumption using high-speed microprocessors. Chapter 3 presents mathematical models to estimate the execution time and the energy consumption based on the microcontroller's system clock frequency. Chapter 4 explains the proposed autonomous system clock frequency controller by making use of the mathematical models presented in chapter 3. Chapter 5 shows the implementation of the mathematical model and the proposed autonomous system clock frequency controller

elaborated in chapters 3 and 4 respectively using a representative nano-satellite, VELOX-I. The experimental setup and results using the proposed controller are presented in chapter 6. Chapter 7 concludes the research work and recommendations for future work.

2. Literature Review

2.1. Introduction

In this chapter, various reported works to overcome the extreme power limitation on miniaturized-satellites will be presented. There are two main approaches. One is to improve the input power and the other is to reduce the power consumption. On the reduction of the power consumption, various works in the computer science to reduce the energy consumption of high-speed microprocessors will also be presented.

2.2. Improving Input Power

2.2.1. Deployable Solar Panels

One approach to increase the input power of a miniaturized satellite is through deployable solar panels. The main idea is to increase the solar array area in the miniaturized satellite.

Figure 2-1: XSAS shown attached to 2U CubeSat in a) 1U stowed configuration and b) fully deployed configuration [17]

Figure 2-1 shows the eXtendable Solar Array System (XSAS), a project from the University of Michigan. A 1U configuration XSAS could provide more than 20 watt-hours average power, nearly triple the average of 7 watt-hours power generation capability of a 3U CubeSat [17]. It also shows that its 1.5U configuration could provide up to five times the amount of power of

standard CubeSat configurations with the same size [16]. Another research [72] has also shown similar result with 20-30 Watts on-orbit average power through deployable solar panels, and a recent research [18], has shown a maximum power of 50.4 W obtained on a 3U CubeSat.

This method depends on the reliability of mechanical design to ensure successful deployment of the solar panels. Its main drawback is the increase of the weight.

2.2.2. Optimization of Power Efficiency

In this research area, the objective is to improve the power efficiency through improving output power from the solar array. In [19], a combined constant current and constant voltage battery charging is used for maximum power point tracker (MPPT) solar array control, resulting in 85% to 91% power efficiency. In [26], a Fourth-Order Buck Converter is used as an alternative to a typical buck converter with an input L-C filter. Implemented with a popular Perturb and Observe (P & O) Algorithm for MPPT [21, 27] in dsPIC30F6010, this work shows up to 98% MPPT efficiency in its test. Other works in this area include Current Feedback Based MPPT Controller [22], MPPT method using Taylor mean value theorem [23], MPPT control for photovoltaic system using adaptive neuro-fuzzy “ANFIS” [24], Microcontroller based intelligent DC/DC converter for MPPT [25], and MPPT employing sliding mode control [28].

The power improvement obtained by the research in this area is varying. Typically, the MPPT efficiency ranges from 80-89%. Under certain conditions, however, the measured MPPT efficiency can be higher than 90% [19, 25, 26, 28].

Unlike the deployable solar panels, this implementation requires little addition to the weight and the volume of the satellite.

2.2.3. Optimization of Orbit Path

The main idea in this research is to maximize the solar energy captured on the satellites based on orbit path selection.

Table 2-1: Different Low Earth Orbit (LEO) Parameters [30]

Altitude (km)	Satellites velocity (m/s)	Orbital period (h:min:sec)	Time fraction in shadow	Eclipse time (min:sec)	Illumination time (min:sec)
160	7.8087	1:27:41	42.94	38:39	50:02
500	7.6132	1:34:36	37.79	36:45	59:51
1000	7.3507	1:45:07	33.23	35:56	70:11
1500	7.1136	1:55:58	30.03	35:50	81:08
1700	7.025	2:00:25	28.97	35:53	86:32

Table 2-2: Different Beta Angle Range based on Orbital Inclination [30]

Orbital inclination i°	30	60	90	150
β angle range	[-52, 52]	[-80, 80]	[-86, 86]	[-51, 51]

Table 2-1 and Table 2-2 [30] show how different orbit paths produce different sun-exposure/eclipse duration and different orbit (beta) angle (the minimum angle between the satellite's orbit plane and solar vector). These differences would result in different solar energy harvested. By being able to choose the orbit path, the amount of solar energy input to the satellite could also be increased. The amount of power gain using this approach depends on the orbit path chosen.

As this idea does not depend on mechanical, electronics, and algorithm parts of the satellite, implementation of this idea costs nothing to the weight and size of the satellite. However, this approach assumes that the orbit path can be chosen. Often, miniaturized satellites are the piggybacks to bigger satellites launch and therefore its orbit path has been fixed by the main passenger.

2.3. Reducing Power Consumption of Satellite

The work in [31] exploits regularity of a satellite's orbit path to reduce the power consumption in a satellite. By introducing periodic energy recharge model for a deterministic satellite's orbit, it can partition the orbit into several independent regions with different characteristics (such as type of computation, importance, performance requirements, and energy consumption). Based on the given characteristics and energy recharge model developed, optimal voltages for the regions can be determined such that overall performance is maximized within the energy budget in the period.

Figure 2-2: Screenshot of Jtrack Satellite Orbit [31]

Table 2-3: Example of Characteristics of the System per Region [31]

Region	R_1	R_2	R_3	R_4	R_5	R_6	R_7
$d_i(\text{min.})$	26.1	11.6	11.6	8.7	14.5	11.6	20.3
λ_i	0.4	1	0.2	0.4	1.2	0.4	0.2
$\mu_{i,\text{ref}}$	1	1.5	1	1	1.8	1	1
w_i	1	1.5	0.9	0.7	1.2	0.7	0.7
$e_{i,\text{ref}}(\text{mJ})$	2.1	2.5	1.8	2.1	2.5	2.1	1.8
B	100	50	80	100	70	100	80
Λ_i	0.1	0.02	0.1	0.05	0.01	0.05	0.1

Figure 2-2 shows an example of a satellite's orbit path with various satellite system characteristics based on different regions. By using the given characteristics and energy recharge model developed per region, optimal voltages for the regions can be determined such that overall performance is maximized within the energy budget in the period [31]. This approach assumes that the Dynamic Voltage Scaling (DVS) is available in the satellite system and that a satellite's workload in different regions would be different.

In satellite communications, inefficient algorithm causes longer transmission time. Efficient algorithm for signal decoding in satellite communication is studied in [32-34]. In [33], a transformed Viterbi Algorithm to decode digital signal is introduced to reduce the number of data address toggle as well as more efficient indexing in the microcontroller algorithm. The power reduction depends on the microcontroller's architecture and instruction sets to carry out the algorithm, as well as the algorithm themselves.

Apart from the satellite-specific research areas, there are various works in computer science to reduce energy consumption [35-44, 46-48]. The popular techniques include Dynamic

Voltage Scaling/Dynamic Voltage and Frequency Scaling (DVS/DVFS) [37, 38, 44, 46, 47, 73], Dynamic Power Management (DPM) [35, 43], Feedback Control Scheduling (FCS) [36], as well as power-aware task scheduling [36, 39-42, 44, 46, 48]. These techniques are discussed in the sequel.

DVS/DVFS technique slows down the underutilized resources by decreasing their operating voltages, or their operating frequencies, or both. It relies on special hardware, such as a programmable DC-DC switching voltage regulator or a programmable clock generator, and a high performance processor with wide operating frequency range. The energy reduction advantage is considerably high, due to the quadratic relationship between supply voltage and energy consumption as well as the polynomial relationship between frequency and energy consumption, as shown in some models [36, 38, 39, 74].

DVS algorithm aims to follow computational load of the processors closely by feedback mechanism which computes the amount of idle time of the processors over a period of time. However, there are significant issues when implemented in real-time as tasks' deadlines cannot be traded-off with energy-saving [73].

The DVS algorithm assumes the use of a high performance processor. This ensures that scheduling and pre-emption overheads in the DVS algorithm are negligible. Without a high performance processor, the DVS algorithm cannot be implemented. Hence, it is not a good choice for implementation with low-speed processors. DVS also cannot be applied if the supply voltage to a micro-processor is fixed.

DPM refers to a selectively shut-off of idle system components. It is a design methodology for dynamically reconfiguring systems to provide the requested services and performance levels with a minimum number of active components or a minimum load on such component microprocessor [43]. It does not scale down the voltage or the frequency of the system. As compared to the DVS/DVFS technique, DPM has the advantage of not requiring special hardware. For multi-devices system where the system components other than the processors are major source of energy consumption, DPM is preferred over DVS/DVFS due to its simpler implementation.

FCS technique [36] introduces a software feedback loop that dynamically control software resource allocation based on the system availability, system requirements, as well as system constraints. One of the FCS techniques, called the dynamic slack management FCS, introduces resource allocation method that reduces the overall energy consumption of the system by producing manageable slack time to accomplish given tasks under energy-constrained scenario. FCS algorithm only requires the software algorithm to monitor the system availability and to control the software resource allocation. Unlike DVS/DVFS or DPM, it does not deal directly with the power consumption issue. It assumes that if software resources could be allocated more efficiently, then the power consumption could be reduced by a different technique.

Power-aware design does not necessarily imply minimization of power or energy. Power-aware system is one which the system modifies its behavior based on current power/energy availability [45]. A task scheduling technique estimates a process' resource requirements based on the process' model or the task scheduler's knowledge of the process' workload, or worst-case execution times (WCET), to allocate microprocessor resources. Power-aware task scheduling technique puts significant consideration on power/energy aspect of the system in its scheduling [36, 39-42, 44]. Its implementation may not follow power reduction techniques (DVS/DVFS and DPM) which depend solely on time (or computational) performance and computational resources. Instead, power-aware scheduling design takes power/energy input feedback for its scheduling consideration. Hence, the power-aware scheduling depends on the availability of the hardware which could provide power/energy input feedback for its use, which may not necessarily be present in every system.

Table 2-4 summarizes the CPU's energy reduction techniques, its advantages and disadvantages. From the table it is observed that there are several energy efficient designs but may not fit well in the miniaturized satellite applications, whose main processor is typically a low-speed (≤ 100 MHz) micro-controller.

Table 2-4: Summary of CPU's Energy Reduction Techniques

No	Technique	Advantages	Disadvantages
1	DVS/DVFS	<ul style="list-style-type: none"> • Does not depend on implementation of other techniques • Minimize the power consumption of a processor • Suitable choice for a system whose processor consumes power the most 	<ul style="list-style-type: none"> • Requires DC-DC switching voltage regulator • Requires high-speed processors • Possess risk due to changing of system clock frequency. Especially true for implementation of traditional DVS algorithms in time-critical system
2	DPM	<ul style="list-style-type: none"> • Simple to implement • Does not possess risk due to changing of system clock frequency • Suitable choice for multi-device systems whose activities are controlled by a single (main) controller • Does not depend on implementation of other techniques 	<ul style="list-style-type: none"> • Does not reduce the power usage of a processor • Not suitable for a system whose processor consumes power the most
3	FCS	<ul style="list-style-type: none"> • Efficient use of software resources • Independent of special hardware 	<ul style="list-style-type: none"> • Cannot reduce the energy consumption by itself • Depend on implementation of other techniques
4	Power-Aware Scheduling	<ul style="list-style-type: none"> • Taking power into design consideration • Flexible implementation, does not strictly require special hardware (except for providing power/energy use feedback) 	<ul style="list-style-type: none"> • Does not aim to minimize power • Requires power/energy use feedback • Depend on implementation of other techniques

2.4. Implementation Issues on Low Speed Microcontrollers

Among the works presented in section 2.3, DVS/DVFS technique is the technique which is designed to minimize the power consumption of a processor through dynamic scaling. However, it has major drawback to implement it in the miniaturized satellite applications due to its special hardware requirement of DC-DC switching voltage regulator and high-speed processors.

Modern microcontrollers can support a wide range of system clock frequencies [50-65]. Texas Instruments (TI) low power MSP430F525x microcontroller series, for instance, is installed with an internal oscillator which operates in a typical 9.4 kHz frequency and a high frequency crystal that supports up to 32 MHz [63]. Another example is Silicon Labs C8051F120 microcontroller. It supports 32 kHz to 100 MHz range of system clock frequency [64, 75].

To implement the configurable system clock frequency feature, the common technique is to use a frequency divider and a phase-lock loop (PLL) circuit. Those circuits allow the microcontroller to generate base system clock that differs from the original oscillator frequencies. The effective system clock is further fed to various peripherals in the microcontroller to generate their individual clock rate.

2.5. Proposed Autonomous System Clock Frequency Controller

Given the common trait of wide-range of operating frequencies among low-speed COTS microcontrollers, it is possible to implement Dynamic Frequency Scaling (DFS) on them. This work proposes an autonomous, system clock frequency controller to optimize the power usage of microcontrollers in miniaturized satellite applications. It makes use of the typically underused wide operating frequency range in common COTS microcontrollers. One of the major concerns posed by applying changeable system clock frequency in a microcontroller is the control system's stability, which is affected by the sampling rate. The system clock of a microcontroller does not directly affect the sampling rate of the system, but its peripheral does. To ensure the stability of a system while altering its microcontroller's system clock is to maintain its peripheral clock rates.

2.6. Summary

In this chapter, a survey on the power limitation issue on pico- and nano-satellite classes and various works in the computer science to reduce high-speed microprocessors energy consumption have been presented. The survey reveals that, there is a lack in research of configurable system clock frequencies in the low-speed microcontrollers to reduce energy consumption in miniaturized satellite applications.

In the subsequent chapters, a new approach is proposed to solve the power limitation issue by using system clock frequencies inherently designed in most of the COTS microcontrollers.

3. System Modeling

3.1. *Introduction*

In chapter 1, it is shown that there exists an extreme limitation of power in the pico- and nano-satellites. In chapter 2, it is shown that there had been various works related to miniaturized satellites to tackle the extreme power limitation issue. In the same chapter, it is shown that there had been works done aiming to reduce the power consumption for the high-speed microprocessors.

In this research, the power constraint in the miniaturized satellite is overcome by introducing a modified DFS technique, with suitable time and energy estimator models where peripherals components other than the microprocessor are present and taken into consideration by the models.

In this research work, a guaranteed system time performance while minimizing energy consumption for pico- and nano- Satellite applications is proposed.

3.2. *Task's Execution Time Model*

The system clock frequency directly affects the microcontroller's processing speed. Doubling the system clock frequency will result in doubling the number of instructions per second (IPS) and halving the time needed to execute the same instructions.

Therefore, in the scheduling algorithm of DVS/DVFS, it is generally assumed that for every given real-time task in the RTOS is independent from other tasks, the worst case execution time (WCET) of a task will increase in the same proportion as the inverse of the processing speed. If a task is not independent from all other tasks, however, the worst case execution time (WCET) of a task will not necessarily increase in the same proportion.

In a microcontroller's application, the dependency among tasks are common, especially among foreground and background tasks. This is because of the microcontroller's inherent design where the application usually depends significantly on the peripherals.

In a typical application using the microcontroller, some tasks are only affected by the microcontroller's processing speed and some are not. For instance, a task like sorting microcontroller's internal memory depends only on the microcontroller's speed. But another task such as sending data from the microcontroller to a personal computer (PC) through serial port does not only depend on the microcontroller's speed, but also serial communication peripheral rate. Therefore doubling the system clock frequency will not necessarily result in halving a microcontroller's task's execution time.

Table 3-1 lists some examples of the tasks which are either completely dependent or partially dependent on the microcontroller's speed.

Table 3-1: Examples of Completely Dependent and Partially Dependent Tasks

Completely Dependent	Partially Dependent
<ul style="list-style-type: none"> • Sorting algorithm • Data encryption and decryption • Data type conversion (ex: calendar to time) • Mathematical equation • Error detection algorithm • Error correction algorithm • Task scheduler • Internal counter • Digital signal/image data processing algorithm • Data parsing • Orbit propagator algorithm • Hardware multiple and accumulate (MAC) calculation 	<ul style="list-style-type: none"> • Data reading from or writing to external memory (SD card, EEPROM, external RAM, network drive) • Debug printing to PC • Inter-device communication through any computer bus (such as Inter-Integrated Circuit (I2C), Serial Peripheral Interface (SPI), or Controller Area Network (CAN)) • Data uplink and downlink • Analog-to-Digital (ADC) and Digital-to-Analog data reading • Reading from Inertia Measurement Unit (IMU) • External timer device read/write • Programmable Counter Array (PCA) timer read/write • Port pin switching

Hence, the modelling of execution time in a microcontroller's task should consider the presence of the partially dependent tasks.

3.2.1. Single Execution Time Model

Let f_s be a positive real-valued variable representing the system clock frequency of the microcontroller. Define

$$T_s = \frac{1}{f_s} \quad (3-1)$$

as the system clock period of the microcontroller.

Let t_e be the foreground task's execution time, a task's single (one time) execution time can be modeled as a linear function of the system clock period

$$t_e = a_1 T_s + a_2 \quad (3-2)$$

where a_1 and a_2 are constants. a_1 represents the execution time of the task's element that is dependent on microcontroller's speed, while a_2 represents the maximum execution time of the task's element (code segment) that is independent from the microcontroller's speed

3.2.2. Foreground Tasks' Execution Time Model

For software with n_f number of foreground tasks whose state (either running or stopped) is controlled by binary parameter b_f at any given time, the relationship between system clock period T_s and foreground tasks' total execution time $t_{f_{tot}}$ can be written as

$$t_{f_{tot}} = \sum_{i=1}^{n_f} b_{f_i} t_{ef_i} = \sum_{i=1}^{n_f} b_{f_i} (a_{1f_i} T_s + a_{2f_i}) \quad (3-3)$$

where $t_{f_{tot}}$ is the foreground tasks' total execution time, b_{f_i} is the i^{th} foreground task binary parameter - b_{f_i} is 1 if i^{th} foreground task is running and 0 if it is not running. t_{ef_i} is the i^{th} foreground task's execution time, n_f is the number of foreground tasks, a_{1f_i} and a_{2f_i} are the i^{th} foreground task's constants.

In vector form, (3-3) can be rewritten as

$$t_{f_{tot}} = \mathbf{b}_f(\mathbf{a}_{1f}T_s + \mathbf{a}_{2f}) \quad (3-4)$$

where, $\mathbf{b}_f$ is $1 \times n_f$ row vector, $\mathbf{a}_{1f}$ and $\mathbf{a}_{2f}$ are $n_f \times 1$ column vectors.

3.2.3. Background Tasks' Execution Time Model

Background tasks in a microcontroller's application refer to its peripheral interrupt service routines (ISR). There are two types of background tasks: One is event-dependent (non-time dependent) and the other is time-dependent.

3.2.3.1. Event-Dependent Background Tasks' Execution Time Model

An event-dependent background task does not depend on the microcontroller's timer to run. It normally occurs due to foreground tasks activities which trigger the interrupts. One notable example is Universal Asynchronous Receiver/Transmitter (UART) ISR which is used in microcontroller's for debugging.

Given that a single time execution of an event-dependent background task can be formulated as

$$t_{be} = a_{1be}T_s + a_{2be} \quad (3-5)$$

For software with n_f foreground tasks whose state (either running or stopped) is controlled by a binary parameter b_f at any given time and n_{be} event-dependent background tasks, we define c_{ij} as the number of times i^{th} foreground task incurs j^{th} event-dependent background task's execution when it is running such that the relationship between system clock period T_s and the software's event-dependent background tasks' total execution time $t_{be_{tot}}$ can be expressed as

$$t_{be_{tot}} = \sum_{j=1}^{n_{be}} \sum_{i=1}^{n_f} b_{fi} c_{ij} (a_{1be_j}T_s + a_{2be_j}) \quad (3-6)$$

where b_{fi} is the i^{th} foreground task binary parameter - b_{fi} is 1 if i^{th} foreground task is running and 0 if it is not. c_{ij} is the number of times i^{th} foreground task incurs j^{th} event-dependent

background task's execution. n_f and n_{be} are the number of foreground tasks and event-dependent background tasks in the software respectively. a_{1be_j} and a_{2be_j} are the j^{th} event-dependent background task's constants.

In matrix form, the equation (3-6) can be rewritten as

$$t_{be_{tot}} = \mathbf{b}_f \mathbf{C} (\mathbf{a}_{1be} T_s + \mathbf{a}_{2be}) \quad (3-7)$$

where $\mathbf{b}_f$ is $1 \times n_f$ row vector, $\mathbf{C}$ is $n_f \times n_{be}$ matrix, $\mathbf{a}_{1be}$ and $\mathbf{a}_{2be}$ are $n_{be} \times 1$ column vector.

3.2.3.2. Time-Dependent Background Tasks' Execution Time Model

A time-dependent background task depends on the microcontroller's timer alone to run. It occurs periodically. One common example of a time-dependent background task is the timer used as sampling period generator.

Given that a single time execution of a time-dependent background task can be formulated as

$$t_{bt} = a_{1bt} T_s + a_{2bt} \quad (3-8)$$

For software with n_{bt} time-dependent background tasks whose state (either running or stopped) is controlled by binary parameter b_b at any given time, and where T_{btp_k} , $t_{f_{tot}}$, and $t_{be_{tot}}$ are the execution period of the k^{th} time-dependent background task, the software's foreground tasks' total execution time, and the software's event-dependent background tasks' total execution time respectively, the relationship between system clock period T_s and the software's time-dependent background tasks' total execution time $t_{bt_{tot}}$ can be formulated as

$$t_{bt_{tot}} = \sum_{k=1}^{n_{bt}} b_{b_k} \left(\frac{t_{f_{tot}} + t_{be_{tot}} + t_{bt_{tot}}}{T_{btp_k}} \right) (a_{1bt_k} T_s + a_{2bt_k}) \quad (3-9)$$

where

- $t_{bt_{tot}}$, $t_{f_{tot}}$, and $t_{be_{tot}}$, are the software's time-dependent background tasks', foreground tasks', and event-dependent background tasks' total execution time respectively.

- b_{b_k} is the k^{th} time-dependent background task binary parameter - b_{b_k} is 1 if k^{th} time-dependent task is running and 0 if the foreground task is not running.
- n_{bt} are the number of time-dependent background tasks in the software
- T_s and T_{btp_k} are the system clock period and the k^{th} time-dependent background task's period respectively, and
- a_{1bt_k} and a_{2bt_k} are the k^{th} time-dependent background task's constants

Let $b_{bt_k} = \frac{b_{b_k}}{T_{btp_k}}$, and substitute equation (3-4) and (3-7) for $t_{f_{tot}}$ and $t_{be_{tot}}$ respectively, (3-9)

can be rewritten in matrix form as

$$t_{bt_{tot}} = \frac{\mathbf{b}_{bt}(\mathbf{a}_{1bt}T_s + \mathbf{a}_{2bt})}{1 - \mathbf{b}_{bt}(\mathbf{a}_{1bt}T_s + \mathbf{a}_{2bt})} \mathbf{b}_f\{(\mathbf{a}_{1f}T_s + \mathbf{a}_{2f}) + \mathbf{C}(\mathbf{a}_{1be}T_s + \mathbf{a}_{2be})\} \quad (3-10)$$

where

- $\mathbf{b}_{bt}$ is $1 \times n_{bt}$ row vector
- $\mathbf{a}_{1bt}$ and $\mathbf{a}_{2bt}$ are $n_{bt} \times 1$ column vector

From (3-4) and (3-7) we have,

$$t_{f_{tot}} + t_{be_{tot}} = \mathbf{b}_f\{(\mathbf{a}_{1f}T_s + \mathbf{a}_{2f}) + \mathbf{C}(\mathbf{a}_{1be}T_s + \mathbf{a}_{2be})\} \quad (3-11)$$

From (3-10) and (3-11), the execution time of the time-dependent background tasks can be seen as a fraction of the summation of the execution time of foreground tasks and event-dependent background tasks. Where the fraction $q_{t_{bt}}$ can be written as

$$q_{t_{bt}} = \frac{\mathbf{b}_{bt}(\mathbf{a}_{1bt}T_s + \mathbf{a}_{2bt})}{1 - \mathbf{b}_{bt}(\mathbf{a}_{1bt}T_s + \mathbf{a}_{2bt})} \quad (3-12)$$

Hence, we can also express the time-dependent background tasks' total execution time $t_{bt_{tot}}$ as

$$t_{bt_{tot}} = q_{t_{bt}}(t_{f_{tot}} + t_{be_{tot}}) \quad (3-13)$$

From (3-12), it is noted that if $\mathbf{b}_{bt}(\mathbf{a}_{1bt}T_s + \mathbf{a}_{2bt}) \geq 1$, then the time expressed in (3-10) and (3-13) will be invalid, since (3-10) will be either infinity or negative. Such case may happen if software only runs collection of background tasks without any foreground tasks or event-dependent background tasks, or the time-dependent background tasks' total execution time is too long.

Therefore, a software design must satisfy the following condition:

$$0 \leq \mathbf{b}_{bt}(\mathbf{a}_{1bt}T_s + \mathbf{a}_{2bt}) < 1 \quad (3-14)$$

As $\mathbf{b}_{bt}(\mathbf{a}_{1bt}T_s + \mathbf{a}_{2bt}) \rightarrow 1$, the software's time-dependent background tasks' portion is getting bigger, leaving shorter amount of time for foreground tasks' and event-dependent background tasks' execution. Therefore, a good software design will keep the execution time of time-dependent background tasks' as low as possible as compared to the total execution time of the software, that is,

$$\mathbf{b}_{bt}(\mathbf{a}_{1bt}T_s + \mathbf{a}_{2bt}) \rightarrow 0 \quad (3-15)$$

such that

$$q_{t_{bt}}, t_{bt_{tot}} \rightarrow 0 \quad (3-16)$$

3.2.3.3. Background Tasks' Worst Case Execution Time (WCET) Scenario

The equations (3-7) and (3-10) reflect the background tasks' worst case execution time (WCET) scenario in the current loop. Suppose in the l^{th} loop, a foreground task T_{fi} triggers c_{ij} number of calls to an event-dependent background task T_{bej} . The event-dependent background task T_{bej} would be executed, in addition to its remaining jobs, as many as c_{ij} number of times due to T_{fi} 's call. The c_{ij} number of execution, however, does not necessarily occur all together in the l^{th} loop.

Suppose in the l^{th} loop T_{be} has no remaining jobs to start with and is not called by other active foreground tasks in the l^{th} loop. And suppose that it is only triggered to be called c_{ij} times by T_{fi} but is only executed $c_{ij-l} < c_{ij}$ times. In the $(l+1)^{th}$ loop, if the remaining background

tasks' jobs not been cancelled by the starting of the subsequent loop, even without any additional call made in the $(l + 1)^{th}$ loop to T_{be_j} , it would still be executed $c_{ij_{l+1}} \leq c_{ij} - c_{ij_l}$ number of times to accomplish its left-over jobs given by T_{f_i} in the l^{th} loop.

Hence, the expression $\mathcal{C}(\mathbf{a}_{1be}T_s + \mathbf{a}_{2be})$ controlled by binary parameters $\mathbf{b}_f$ in (3-7) and (3-10) represents the WCET scenario of the event-dependent background tasks which creates WCET scenario for the time-dependent background tasks as well.

3.2.3.4. Current Background Tasks' Actual Execution Time Model

Now, suppose the $1 \times n_{be}$ row vector $\mathbf{c}_{j_{act}} = [c_{1_{act}}, c_{2_{act}}, c_{3_{act}}, \dots, c_{n_{be_{act}}}]$ represent the set of the actual number of event-dependent background tasks executed by the microcontroller at the end of l^{th} loop, then the actual total execution time taken by all the event-dependent background tasks in the l^{th} loop $t_{be_{act}}$ can be formulated as

$$t_{be_{act}} = \mathbf{c}_{j_{act}}(\mathbf{a}_{1be}T_s + \mathbf{a}_{2be}) \quad (3-17)$$

where

- $\mathbf{c}_{j_{act}}$ is $1 \times n_{be}$ row vector which contains of the collection of actual number of times the event-dependent background tasks are called in the current loop $[c_{1_{act}}, c_{2_{act}}, c_{3_{act}}, \dots, c_{n_{be_{act}}}]$.
- $t_{be_{act}}$ is the actual total execution time of the event-dependent background tasks in the current loop

Consequently, the actual time-dependent background tasks execution time $t_{bt_{act}}$ at the end of l^{th} loop caused by $t_{be_{act}}$ can be modeled as

$$t_{bt_{act}} = q_{t_{bt}}\{t_{f_{tot}} + t_{be_{act}}\} \quad (3-18)$$

Therefore, the actual background tasks' execution time at the end of l^{th} loop can be expressed as

$$t_{b_{act}} = t_{be_{act}} + t_{bt_{act}} = q_{t_{bt}}t_{f_{tot}} + (1 + q_{t_{bt}})t_{be_{act}} \quad (3-19)$$

where, $t_{b_{act}}$ is the actual total execution time of the background tasks in the current loop

3.2.3.5. Left-Over Background Tasks' Actual Execution Time Model

At the start of l^{th} loop, the $1 \times n_{be}$ row vector $\mathbf{c}_{j_{left}} = [c_{1_{left}}, c_{2_{left}}, c_{3_{left}}, \dots, c_{n_{be_{left}}}]$ represent the set of left-over number of event-dependent background tasks to be executed, then the additional execution time required by the left-over event-dependent background tasks in the l^{th} loop $t_{be_{left}}$ can be formulated as

$$t_{be_{left}} = \mathbf{c}_{j_{left}}(\mathbf{a}_{1be}T_s + \mathbf{a}_{2be}) \quad (3-20)$$

and the consequent additional time-dependent background tasks execution time $t_{bt_{left}}$ required at the start of l^{th} loop caused by $t_{be_{left}}$ can be modeled as

$$t_{bt_{left}} = q_{t_{bt}} t_{be_{left}} \quad (3-21)$$

Therefore, from (3-20) and (3-21), the additional background tasks' execution time margin $t_{b_{margin}}$ required at the start of l^{th} loop can be expressed as

$$t_{b_{margin}} = t_{be_{left}} + t_{bt_{left}} = (1 + q_{t_{bt}})t_{be_{left}} = \frac{\mathbf{c}_{j_{left}}(\mathbf{a}_{1be}T_s + \mathbf{a}_{2be})}{1 - \mathbf{b}_{bt}(\mathbf{a}_{1bt}T_s + \mathbf{a}_{2bt})} \quad (3-22)$$

3.2.4. Worst Case Execution Time (WCET) Model

3.2.4.1. Total Execution Time Model without Leftover Background Tasks' Job

Suppose there is no leftover background task in the l^{th} loop, the total execution time is given by

$$t_{tot} = t_{f_{tot}} + t_{be_{tot}} + t_{bt_{tot}} \quad (3-23)$$

Substitute (3-4), (3-7), and (3-10) into (3-23), the total execution time becomes

$$t_{tot} = \frac{\mathbf{b}_f \{(\mathbf{a}_{1f}T_s + \mathbf{a}_{2f}) + \mathbf{C}(\mathbf{a}_{1be}T_s + \mathbf{a}_{2be})\}}{1 - \mathbf{b}_{bt}(\mathbf{a}_{1bt}T_s + \mathbf{a}_{2bt})} \quad (3-24)$$

By expanding and rearranging (3-24), the relationship between t_{tot} and T_s can be written as

$$t_{tot} = \frac{p_1 T_s + p_2}{p_3 - p_4 T_s} \quad (3-25)$$

where

$$p_1 = \mathbf{b}_f(\mathbf{a}_{1f} + \mathbf{C}\mathbf{a}_{1be}) \quad (3-26)$$

$$p_2 = \mathbf{b}_f(\mathbf{a}_{2f} + \mathbf{C}\mathbf{a}_{2be}) \quad (3-27)$$

$$p_3 = 1 - \mathbf{b}_{bt}\mathbf{a}_{2bt} \quad (3-28)$$

$$p_4 = \mathbf{b}_{bt}\mathbf{a}_{1bt} \quad (3-29)$$

Here, it is to be noted that since every element in $\mathbf{a}_{1bt}$, $\mathbf{a}_{2bt}$, $\mathbf{a}_{1f}$, $\mathbf{a}_{2f}$, $\mathbf{a}_{1be}$, $\mathbf{a}_{2be}$, $\mathbf{b}_{bt}$, $\mathbf{b}_f$ vectors and $\mathbf{C}$ matrix is non-negative, from (3-26)-(3-29), we can conclude that p_1 , p_2 , and p_4 are non-negative while p_3 is non-negative if $\mathbf{b}_{bt}\mathbf{a}_{2bt} \leq 1$. However, based on (3-14), we also derive that p_3 must be positive. Mathematically, we write it as,

$$\begin{aligned} p_1, p_2, p_4 &\geq 0 \\ p_3 &> 0 \end{aligned} \quad (3-30)$$

Given any foreground task, p_1 and p_2 cannot be both zero. Thus,

$$p_1 + p_2 > 0 \quad (3-31)$$

In the event that there is no time-dependent background task,

$$p_4 = 0, p_3 = 1 \quad (3-32)$$

If time-dependent background tasks are present, then,

$$p_4 > 0, p_3 < 1 \quad (3-33)$$

3.2.4.2. System Clock Frequency Requirement Model without Leftover Background Tasks'

Job

From (3-25), the system clock period T_s can be expressed in terms of total execution time t_{tot} as

$$T_s = \frac{p_3 t_{tot} - p_2}{p_4 t_{tot} + p_1} \quad (3-34)$$

Using the definition in (3-1), (3-34) can be rewritten as

$$f_s = \frac{p_4 t_{tot} + p_1}{p_3 t_{tot} - p_2} \quad (3-35)$$

Since $f_s, p_3 > 0$ and $p_1, p_2, p_4 \geq 0$, from (3-35) we may derive the minimum time requirement constraint $t_{req_{min}} > 0$ such that $p_3 t_{req_{min}} - p_2 > 0$ as,

$$t_{req_{min}} > \frac{p_2}{p_3} \quad (3-36)$$

Given a time requirement $t_{req_{now}}$ the software has to complete its given tasks controlled by $\mathbf{b}_f$ and $\mathbf{b}_{bt}$ vectors in the current loop where $t_{req_{min}} \leq t_{req_{now}} \leq t_{tot}$, we can derive system clock frequency requirement $f_{req_{now}}$ from (3-35) as

$$f_{req_{now}} = \frac{p_4 t_{req_{now}} + p_1}{p_3 t_{req_{now}} - p_2} \geq f_s \quad \text{If } \frac{p_4}{p_3} \leq f_s, \text{ or} \quad (3-37)$$

$$f_{req_{now}} = \frac{p_4 t_{req_{now}} + p_1}{p_3 t_{req_{now}} - p_2} < f_s \quad \text{If } \frac{p_4}{p_3} > f_s \quad (3-38)$$

If there is no time-dependent background task, based on (3-32), (3-37) should be used to find $f_{req_{now}}$ as $\frac{p_4}{p_3} = 0 < f_s$ is true. If any background task is present, then from (3-30), (3-31), (3-33), and (3-35), we may derive

$$\begin{aligned}
p_1 p_3 + p_2 p_4 &> 0 \\
p_1 p_3 &> -p_2 p_4 \\
p_1 p_3 + p_3 p_4 t_{tot} &> -p_2 p_4 + p_3 p_4 t_{tot} \\
p_4 (p_3 t_{tot} - p_2) &< p_3 (p_4 t_{tot} + p_1) \\
\frac{p_4}{p_3} &< \frac{p_4 t_{tot} + p_1}{p_3 t_{tot} - p_2} = f_s
\end{aligned} \tag{3-39}$$

From (3-39), we may conclude that out of expressions (3-37) and (3-38), only expression (3-37) will be used in all scenarios. Also, since $f_s > 0$ by definition, we may also conclude that $\frac{p_4}{p_3} = f_s$ is impossible in any condition. We may further simplify expression (3-37) to find the system clock frequency requirement in the current loop (excluding unfinished jobs – that is, left over background tasks' job) $f_{req_{now}}$ as,

$$f_{req_{now}} = \frac{p_4 t_{req_{now}} + p_1}{p_3 t_{req_{now}} - p_2} \geq f_s \tag{3-40}$$

3.2.4.3. WCET and System Clock Frequency Requirement Model with Leftover Background Tasks' Job

Taking into consideration the additional time margin required at the start of the loop due to the left over background tasks' job $t_{b_{margin}}$ as shown in (3-40), and taking the maximum value of $t_{req_{now}}$ which is t_{tot} (as $t_{req_{now}} \leq t_{tot}$), the worst case execution time t_{WCET} can be modelled as

$$t_{WCET} = t_{tot} + t_{b_{margin}} \tag{3-41}$$

Using (3-35), the corresponding system clock frequency requirement f_{req} is

$$f_{req} = \frac{p_4 t_{WCET} + p_1}{p_3 t_{WCET} - p_2} \geq f_{req_{now}} \geq f_s \tag{3-42}$$

3.2.5. Execution Time Model Error

From (3-23) and (3-37), the actual execution time model $t_{tot_{act}}$ can be obtained by replacing $t_{be_{tot}} + t_{bt_{tot}}$ with $t_{b_{act}} = t_{be_{act}} + t_{bt_{act}}$

$$t_{tot_{act}} = t_{f_{tot}} + t_{b_{act}} = (1 + q_{t_{bt}})(t_{f_{tot}} + t_{be_{act}}) \quad (3-43)$$

Combining (3-43) with (3-12) and (3-17), (3-43) can be expanded as

$$t_{tot_{act}} = \frac{b_f(a_{1f}T_s + a_{2f}) + c_{j_{act}}(a_{1be}T_s + a_{2be})}{1 - b_{bt}(a_{1bt}T_s + a_{2bt})} \quad (3-44)$$

3.2.5.1. Hard-Deadline System

For a system with hard-deadline nature, the evaluation of the system time performance is measured in one loop or one cycle. Suppose the real execution time of the software in one loop is given by t_s , one can formulate the time model error e_{tm} for a hard-deadline system as,

$$e_{tm} = |t_{tot_{act}} - t_s| \quad (3-45)$$

3.2.5.2. Soft-Deadline System

For a system with soft-deadline nature, the evaluation of the system time performance is measured from summation or average of multiple loops (or cycles) rather than one. Hence, given n_s number of samples one can formulate time model error over multiple loops E_{tm} for a system with soft-deadline nature as,

$$E_{tm} = \left| \sum_{i=1}^{i=n_s} t_{tot_{act}i} - \sum_{i=1}^{i=n_s} t_{s_i} \right| \quad (3-46)$$

3.2.6. Execution Time Margin and Guaranteed System Time Performance

3.2.6.1. Hard-Deadline System

For a hard-deadline system, given the WCET t_{WCET} as expressed in the equation (3-41) and the real execution time t_s , at every instance,

$$t_s \leq t_{WCET} \quad (3-47)$$

must be true.

We can define the execution time margin m_{WCET} to indicate how close the actual execution time from the WCET scenario as,

$$m_{WCET} = t_{WCET} - t_s \quad (3-48)$$

Therefore, in order for a system to have guaranteed system time performance as conditioned in (3-47), m_{WCET} must always be non-negative.

$$m_{WCET} \geq 0 \quad (3-49)$$

3.2.6.2. Soft-Deadline System

For a soft-deadline system, given the WCET t_{WCET} as expressed in the equation (3-41), the real execution time t_s , and n_s samples,

$$\sum_{i=1}^{i=n_s} t_{s_i} \leq \sum_{i=1}^{i=n_s} t_{WCET_i} \quad (3-50)$$

must be true at every instance.

Given a constant t_{WCET} over n_s samples, we can define the summation of execution time margin M_{WCET} to indicate how close the actual execution time over a number of samples from the WCET scenario as,

$$M_{WCET} = n_s t_{WCET} - \sum_{i=1}^{i=n_s} t_{s_i} \quad (3-51)$$

For a system to have guaranteed system time performance as conditioned in (3-50), M_{WCET} must always be non-negative.

$$M_{WCET} \geq 0 \quad (3-52)$$

3.3. Energy Consumption Model

The system clock frequency and power/energy consumption relationship might vary from device to device. There had been various models proposed to estimate the power/energy consumption of microprocessor [36, 38, 39, 45, 74]. As there is no single model to describe the system clock frequency and power/energy consumption relationship model for every device, one may use empirical approach to find the suitable model for the device. One common characteristics is as the system clock frequency increases, so does the power/energy consumption.

Hence, system that has a fixed regulated voltage level, its current can be modeled as a continuous monotonically increasing function of the system clock frequency,

$$I_s = g(f_s) \quad (3-53)$$

Let V_s be the voltage level of the system, $[t_{ch0}, t_{ch1}]$, I_{ch} , and $E_{ch} = \int_{t_{ch0}}^{t_{ch1}} V_s I_{ch} dt$ are the time interval, the current, and the energy consumed to change the system clock frequency respectively, the amount of the energy increment/reduction incurred by changing the original system clock frequency, f_{sorig} , to the system clock frequency target, f_{starg} , for time interval $[t_0, t_1]$ is given by,

$$\Delta E = \int_{t_0}^{t_1} V_s \Delta I_s dt + E_{ch} = \int_{t_0}^{t_1} V_s \left(g(f_{starg}) - g(f_{sorig}) \right) dt + \int_{t_{ch0}}^{t_{ch1}} V_s I_{ch} dt \quad (3-54)$$

Let $\Delta t = (t_1 - t_0)$, assuming V_s , $g(f_{starg})$, and $g(f_{sorig})$ to be independent of time, by averaging I_{ch} over time interval $[t_{ch0}, t_{ch1}] = \Delta t_{ch}$, (3-54) can be simplified as,

$$\Delta E = V_s \left(g(f_{starg}) - g(f_{sorig}) \right) \Delta t + V_s \overline{I_{ch}} \Delta t_{ch} \quad (3-55)$$

where

$$E_{ch} = V_s \overline{I_{ch}} \Delta t_{ch} \quad (3-56)$$

The energy consumed by the system in the time interval $[t_0, t_1]$ is not increased if $\Delta E \leq 0$. To satisfy it, we have a time constraint,

$$\begin{aligned}
\Delta E &= V_s \left(g(f_{starg}) - g(f_{sorig}) \right) \Delta t + V_s \overline{I_{ch}} \Delta t_{ch} \leq 0 \\
\Delta t \left(g(f_{starg}) - g(f_{sorig}) \right) &\leq -\overline{I_{ch}} \Delta t_{ch} \\
\Delta t \left(g(f_{sorig}) - g(f_{starg}) \right) &\geq \overline{I_{ch}} \Delta t_{ch}
\end{aligned} \tag{3-57}$$

Since for energy reduction $g(f_{sorig}) - g(f_{starg}) > 0$, therefore the time constraint takes form of,

$$\Delta t \geq \frac{\overline{I_{ch}} \Delta t_{ch}}{g(f_{sorig}) - g(f_{starg})} \tag{3-58}$$

There exists a minimum amount of time interval $\Delta t_{min} = \frac{\overline{I_{ch}} \Delta t_{ch}}{g(f_{sorig}) - g(f_{starg})}$ where lowering the system clock frequency to f_{starg} will consume less energy than staying in a higher system clock frequency f_{sorig} .

From (3-58), if the execution time interval Δt_e is known, we have,

$$\begin{aligned}
g(f_{sorig}) - g(f_{starg}) &\geq \frac{\overline{I_{ch}} \Delta t_{ch}}{\Delta t_e} \\
g(f_{starg}) &\leq g(f_{sorig}) - \frac{\overline{I_{ch}} \Delta t_{ch}}{\Delta t_e}
\end{aligned} \tag{3-59}$$

For $f_{sorig} > f_{starg}$, it necessarily implies $g(f_{sorig}) > g(f_{starg})$ as $g(f_s)$ is a monotonically increasing function. Hence, if $g'(x)$ exists, from (3-53) and from monotonically increasing function properties, we can conclude that there exists the maximum system clock frequency target f_{starg_max} such that $g(f_{starg_max}) = g(f_{sorig}) - \frac{\overline{I_{ch}} \Delta t_{ch}}{\Delta t_e}$ allows a system to switch its system clock to a lower frequency and consuming non-increasing amount of energy. That is,

$$f_{starg_max} = g' \left(g(f_{sorig}) - \frac{\overline{I_{ch}} \Delta t_{ch}}{\Delta t_e} \right) \tag{3-60}$$

3.4. *Conclusions*

This chapter has provided mathematical foundation for later discussion on the design as well as on the performance measurement of the proposed controller. It has been shown that the execution time for a system with peripheral components present can be modelled by developing mathematical concepts of foreground, event-dependent (or foreground-task dependent), and time-dependent background tasks. The model can then be used to determine the worst-case execution time of a system in one loop with left-over tasks taken into consideration. The mathematical expressions to measure the system time performance has been shown for system time performance evaluation of hard-deadline and soft-deadline systems. It has also been shown that, for a system which possess an energy cost for changing its system clock frequency, an energy estimator model can be modeled to find the maximum target frequency allowed for positive energy gain. In the next chapter, the design and implementation of an autonomous system clock frequency controller will be presented.

4. Design and Implementation of an Autonomous System Clock Frequency Controller

4.1. Introduction

In chapter 3, the mathematical model used to design the proposed controller is presented. The chapter has established the execution time-system clock frequency relationship for the flight software, the system time performance measurement model, as well as the energy consumption model. In this chapter, the realization of an autonomous system clock frequency controller is presented.

4.2. Design

To realize the autonomous system clock frequency controller system based on the modeling presented in chapter 3, the proposed system is divided into three main components: the time estimator, the energy estimator, and the frequency controller. Figure 4-1 shows the block diagram representation of the proposed controller.

Figure 4-1: System Clock Frequency Controller System Block Diagram

The details are explained in the following sections.

4.2.1. Flight Software

As shown in (3-41), the calculation of the worst case execution time (t_{WCET}) estimation of the software requires two elements t_{tot} and $t_{bmargin}$. To obtain t_{tot} , as shown in (3-24), $\mathbf{b}_f$, $\mathbf{b}_{bt}$, and f_s which represent the foreground tasks' states, the time-dependent background tasks' states, and the actual system clock frequency respectively, must be obtained from the flight software. Then, as shown in (3-22), to obtain $t_{bmargin}$, $\mathbf{c}_{jleft}$, which is the left-over event-dependent background tasks' number of execution must be also be known from the flight software. Thus, by providing $\mathbf{b}_f$, $\mathbf{b}_{bt}$, f_s , and $\mathbf{c}_{jleft}$ parameters to the time estimator, t_{WCET} can be calculated.

Those parameters, however, are not sufficient for the time estimator to calculate the estimation and the actual execution time difference. To monitor the error between the time estimation and the actual execution time, $t_{tot_{act}}$ and t_s must be compared, just as shown in (3-45) and (3-46). But t_s is a flight software parameter obtained after loop execution. Also, to calculate $t_{tot_{act}}$, which is the estimated total execution time, an additional parameter, $\mathbf{c}_{jact}$, which is the actual event-dependent background tasks' number of execution (also known only after the loop execution), must also be provided by the software.

To complete the feedback parameters for both the t_{WCET} and the time estimation error calculations, $\mathbf{b}_f$, $\mathbf{b}_{bt}$, f_s , t_s , $\mathbf{c}_{jact}$, and $\mathbf{c}_{jleft}$ should all be provided by the flight software as feedback to the time estimator as shown in Figure 4-1.

Likewise, the flight software should provide two of its state elements $f_s = f_{sorig}$ and Δt_e , which represent the original system clock frequency and the execution time interval of the system clock frequency controller respectively, to the energy estimator to calculate the maximum target frequency as shown in (3-60). Thus, in Figure 4-1, we can see how the two parameters are fed-back from the flight software to the energy estimator.

The flight software receives the desired system clock frequency f_{sr} from the frequency controller which will be used to change its current system clock frequency to the desired level.

The desired system clock frequency f_{sr} is calculated by the frequency controller and is explained in the later section.

4.2.2. Time Estimator

Figure 4-2: Time Estimator Block Diagram

Figure 4-2 shows the time estimator block diagram. The most important task of the time estimator is to calculate t_{WCET} which will be used to determine f_{req} as expressed in (3-42). The f_{req} will further be passed to the frequency controller to calculate the desired frequency f_{sr} for the flight software to operate. In order to calculate t_{WCET} , apart from the flight software state elements b_f , b_{bt} , f_s , and c_{j_left} , the time estimator parameters a_{1bt} , a_{2bt} , a_{1f} , a_{2f} , a_{1be} , a_{2be} , and C must all be first measured off-line. Once those required parameters are known, t_{WCET} can be calculated based on (3-41) and f_{req} based on (3-42). The two elements of t_{WCET} , t_{tot} and t_{b_margin} , can be obtained from (3-25) to (3-29) and (3-22) respectively. a_{1bt} , a_{2bt} , a_{1f} , a_{2f} , a_{1be} , a_{2be} parameters are to be obtained from experimental results (sampling the running time of the tasks), while C matrix can be obtained by the programmer's specifications, as each element in the C matrix will be known from the system design.

To calculate the time performance parameters E_{tm} , M_{WCET} as shown in (3-44), (3-46), and (3-52), t_{tot_act} , t_s , and c_{j_act} inputs from the software will be used.

4.2.2.1. Time Estimator Optimization

The time estimator is the most computationally intensive part of the algorithm. To implement it in the low-speed microcontroller, optimization is desired. There are at least three areas available for its optimization to reduce its computational load, namely by:

1. Off-line static parameters computation
2. Conditional truncation and scaling, and
3. Completely-dependent tasks identifiers

4.2.2.2. Off-line Static Parameters Computation

Since $\mathbf{a}_{1bt}$, $\mathbf{a}_{2bt}$, $\mathbf{a}_{1f}$, $\mathbf{a}_{2f}$, $\mathbf{a}_{1be}$, $\mathbf{a}_{2be}$ vectors and $\mathbf{C}$ matrix parameters can all be computed before run-time (off-line), the major computations of p_1 , p_2 , p_3 , and p_4 in the equations (3-26)-(3-29) which involve those parameters, can also be computed off-line. The pre-computed values could then be stored in look-up tables for quick referencing during on-line execution. This leaves only $\mathbf{b}_f$ and $\mathbf{b}_{bt}$ binary parameters in the equations to be dynamically changed on run-time, based on flight software feedback, reducing computational load in the time estimator greatly.

4.2.2.3. Conditional Truncation and Scaling

In the 8-bit word-size microcontroller, floating point data-type calculation is often computationally intensive and therefore should be avoided whenever possible. If the parameters in the $\mathbf{a}_{1bt}$, $\mathbf{a}_{2bt}$, $\mathbf{a}_{1f}$, $\mathbf{a}_{2f}$, $\mathbf{a}_{1be}$, $\mathbf{a}_{2be}$ vectors are of the similar scale, then the parameters can be first calculated in single (or double) floating point precision off-line and then truncated and scaled (represented) in less computationally intensive data type (such as int8 or int16) during run-time.

4.2.2.4. Completely-Dependent Tasks Identifiers

Some tasks in the flight software are completely dependent on the processor's speed, implying their independent-from-processor-speed parameters' (a_{2bt_i} , a_{2f_i} , and a_{2be_i} parameters) values to be zero. Such tasks can be identified off-line and put under separated category from the rests for computational optimization. Those identified and categorized tasks can then be calculated separately (and more efficiently) from the partially dependent tasks during run-time to speed up the computational speed of the time estimator.

4.2.3. Energy Estimator

The primary task of the energy estimator is to measure what is the maximum target frequency f_{starg_max} which would be run within a time interval Δt_e that the change in system clock frequency would result in positive energy gain. Obviously, positive energy gain can only be

attained if the system clock frequency is lowered. Thus, the frequency controller should only target the system clock frequency which is lower than f_{starg_max} .

To find f_{starg_max} , the current-system clock frequency model $I_s = g(f_s)$ in (3-53) must be first established based on the empirical data through experimental measurements. Then, the energy consumed to change the system clock frequency E_{ch} in (3-56) is also to be measured through experiments. Afterwards, one can directly apply the formula in (3-60) to get the f_{starg_max} . It is to be noted that $g'(f_s)$ must be necessarily exist to apply the formula. Once calculated, f_{starg_max} is fed to the frequency controller.

4.2.4. Frequency Controller

Based on the inputs from the time and energy estimators, the frequency controller is to feed the flight software with the desired system clock frequency f_{sr} periodically (that is, every Δt_e). To determine f_{sr} , the frequency controller gets three inputs: the system clock frequency requirement f_{req} from the time estimator, the maximum system clock frequency target f_{starg_max} from the energy estimator, and the actual current system clock frequency f_s from the flight software (directly or through the energy estimator).

We define a set of system clock frequency F_s with n_g number of elements such that $F_s = \{f_{s_1}, f_{s_2}, \dots, f_{s_{n_g}}\}$. The frequency controller compares the system frequency requirement f_{req} against the actual current system clock frequency f_s . In the event that the system is underclocked, $f_{req} > f_s$, the controller chooses the optimal system clock frequency level r where $\{r | 1 \leq r \leq n_g, r \in \mathbb{N}\}$ such that $f_{s_r} \in F_s$ satisfies,

$$f_{s_r} \geq f_{req} > f_{s_{r-1}} \quad \text{If } 1 < r \leq n_g, \text{ or} \quad (4-1)$$

$$f_{s_r} \geq f_{req} \quad \text{If } r = 1 \quad (4-2)$$

For the case that the system is overclocked, $f_{req} < f_s$, the frequency controller choose the optimal system clock frequency level r such that $f_{s_r} \in F_s$ satisfies

$$f_{s_r} \geq f_{req} > f_{s_{r-1}} \quad \text{If } 1 < r \leq n_g \text{ and } f_{s_{targ_max}} \geq f_{req}, \text{ or} \quad (4-3)$$

$$f_{s_r} \geq f_{req} \quad \text{If } r = 1 \text{ and } f_{s_{targ_max}} \geq f_{req}, \text{ or} \quad (4-4)$$

$$f_{s_r} = f_s \quad \text{If } f_{s_{targ_max}} < f_{req} \quad (4-5)$$

to be applied to the flight software. No change in the system clock frequency level should be applied if $f_{req} = f_s$.

4.2.5. Summary of the Design Steps

In summary, to design the proposed system, one requires to:

- 1) Define multiple system clock frequency levels in a microcontroller. The system clock frequency levels define should not make the microcontroller's peripheral rates change and must not introduce non-acceptable error percentage to the rates.
- 2) Design a flight software which provide the state parameters: $b_f, b_{bt}, f_s, t_s, c_{j_{act}}, c_{j_{left}},$ and Δt_e
- 3) Obtain the time estimator parameters $a_{1bt}, a_{2bt}, a_{1f}, a_{2f}, a_{1be}, a_{2be}$ from the off-line experiments on the flight software tasks and C matrix from the system design. Makes all kind of optimization on these (for instance, through defining $p_1, p_2, p_3,$ and p_4 and provide its look-up table) whenever possible.
- 4) Obtain current-system clock frequency relationship $I_s = g(f_s)$ and the energy consumption for changing the system clock frequency E_{ch} .
- 5) Design the frequency controller whose logic is as described in section 4.2.4.

4.3. Implementation

The proposed autonomous system clock frequency controller presented in section 4.2 has been implemented and tested using the VELOX-I nano-satellite's flat-satellite as a representative case study. In this work, the implementation is realized only in the microcontroller of the VELOX-I's motherboard, leaving the microcontrollers of the other subsystems having fixed system clock frequencies.

4.3.1. Determining System Clock Frequencies

Table 4-1 shows the peripherals used for VELOX-I Flight Software, its desired rate, and its maximum allowed error percentage of the peripherals.

Table 4-1: Peripheral Rate Requirements for VELOX-I Flight Software

No	Peripheral	Desired Rate	Maximum Allowed Error Percentage
1	Universal Asynchronous Receiver Transmitter 0 (UART0)	115,200 baud	6.0000%
2	Universal Asynchronous Receiver Transmitter 1 (UART1)	115,200 baud	6.0000%
3	Serial Peripheral Interface 0 (SPI0)	500,000 bps	3.0000%
4	Serial Management Bus (SMBus)	400,000 Hz	10.0000%
5	Analog to Digital Converter 0 (ADC0)	750,000 Hz	5.0000%
6	Timer 2 (TMR2)	1,000 Hz	0.5000%

The system clock frequency levels are determined based on the available operating frequencies which provide stable peripheral rates to the system to avoid instability due to the changing of the sampling frequency of the system. In other word, as the system clock frequency changed, the peripherals must still be functional with acceptable performance level.

By using of the available range of system clock frequency in the C8051F120 microcontroller installed on the VELOX-I nano-satellite's motherboard, the system clock is divided into seven levels of system clock frequencies, i.e. $n_g = 7$ as shown in Table 4-2.

Table 4-2: System Clock Frequency Level, System Clock Frequency, and System Clock Period

Level	System Clock Frequency/ Period Symbols	System Clock Frequency (MHz)	System Clock Period (μs)
7	f_{s_7}/T_{s_7}	98.000	0.0102041
6	f_{s_6}/T_{s_6}	79.625	0.0125589
5	f_{s_5}/T_{s_5}	67.375	0.0148423
4	f_{s_4}/T_{s_4}	49.000	0.0204082
3	f_{s_3}/T_{s_3}	36.750	0.0272109
2	f_{s_2}/T_{s_2}	24.500	0.0408163
1	f_{s_1}/T_{s_1}	12.250	0.0816327

A more detailed analysis showing the microcontroller's peripheral rates' error due to changing of the system clock frequencies is shown in the Appendix 1. The minimum and maximum values of the system clock frequencies in Table 4-2 are chosen based on microcontroller's operating frequency limits using its internal oscillator. The resulting range of peripheral error rates after implementing the seven levels of system clock frequencies in Table 4-2 is shown in Table 4-3.

Table 4-3: Range of Peripheral Error Rates in VELOX-I Flight Software after the Implementation of Seven Levels of System Clock Frequencies

No	Peripheral	Minimum Error Percentage	Maximum Error Percentage	Maximum Allowed Error Percentage
1	Universal Asynchronous Receiver Transmitter 0 (UART0)	0.3092%	5.0564%	6.0000%
2	Universal Asynchronous Receiver Transmitter 1 (UART1)	1.5369%	2.8555%	6.0000%
3	Serial Peripheral Interface 0 (SPI0)	0.0000%	2.0833%	3.0000%
4	Serial Management Bus (SMBus)	0.4688%	4.4034%	10.0000%
5	Analog to Digital Converter 0 (ADC0)	0.1572%	2.0833%	5.0000%
6	Timer 2 (TMR2)	0.0041%	0.0163%	0.5000%

From Table 4-3, the implementation of the seven levels of system clock frequencies is acceptable for VELOX-I flight software system as the maximum error percentage of all the peripheral rates incurred from the implementation fall below the maximum allowed error percentage. Therefore the seven levels of system clock frequencies shown in Table 4-2 can be safely applied for the system.

As system clock frequency is changing from time to time, it is possible that the signal glitch may become an issue if the hardware is not chosen properly (that is, if the hardware cannot withstand changing of system clock frequency cause by the controller *on-line*). The hardware choice issue, however, is beyond the scope of this thesis. In this thesis, it is assumed that the hardware chosen is free from the signal glitch. The hardware chosen for the demonstration of the proposed controller in this thesis, C8051F120 microcontroller, is known to be robust to the towards signal glitch based on its long history of use.

4.3.2. Energy Estimator

To obtain the relationship between each system clock frequency level in Table 4-2 and the average current drawn, the motherboard's average current is measured at 5.0 V System Voltage. The measurement results are presented in Table 4-4,

Table 4-4: Motherboard's Average Current at Different System Clock Frequency Level

Level	System Clock Frequency (MHz)	System Voltage (V)	Motherboard's Average Current (mA)
7	98.000	5.0	99
6	79.625	5.0	84
5	67.375	5.0	73
4	49.000	5.0	57
3	36.750	5.0	47
2	24.500	5.0	36
1	12.250	5.0	22

The average current (I_s) is modeled as a function of System Clock frequency (f_s) as,

$$I_s = g(f_s) \quad (3-53)$$

The experimental results in Table 4-4 is shown in Figure 4-3. From the empirical result, the relationship between f_s and I_s can be approximated by a linear equation as

$$g(f_s) = q_1 f_s + q_2 \quad (4-6)$$

Applying first order least square fittings to the data sets, and let S be the error between the actual and the model value, such that

$$S = \sum_{i=1}^n (I_{si} - (q_1 f_{si} + q_2))^2 \quad (4-7)$$

where $n = 7$ is the number of the system clock frequency level, I_{si} is the i^{th} motherboard's actual current reading in mA, f_{si} is the i^{th} actual system clock frequency of the motherboard in MHz, q_1 and q_2 are the constants that minimize S .

The solution for q_1 and q_2 are given by

$$q_1 = \frac{n \sum f_{si} I_{si} - \sum f_{si} \sum I_{si}}{n \sum f_{si}^2 - (\sum f_{si})^2} \quad (4-8)$$

$$q_2 = \frac{1}{n} \left(\sum I_{si} - q_1 \sum f_{si} \right) \quad (4-9)$$

Using empirical data in Table 4-4 to solve q_1 and q_2 , we obtain

$$q_1 = 0.884738, q_2 = 13.265537 \quad (4-10)$$

Thus, the linear model (4-6) can be approximated as

$$I_s = 0.884738 f_s + 13.265537 \quad (4-11)$$

The inverse of the linear model can be obtained from (4-11) as

$$g'(I_s) = \frac{I_s - 13.265537}{0.884738} \quad (4-12)$$

Since $g'(x)$ exists, f_{starg_max} expressed in (3-60) can be found.

The real measurement graph compared to the linear model is shown in Figure 4-3,

Figure 4-3: Original vs 1st Order Least-Squares Fitting for System Clock vs Motherboard's Current

The error graph between the original and the 1st order least square fitting model is shown in Figure 4-4,

Figure 4-4: Error Graph and Error Graph Percentage for Motherboard's Current Model

From Figure 4-4, the absolute error of the 1st order least square fitting for the motherboard's current model for different system clock frequencies are at most 2.1036mA, with the average error of 0.8781mA and average error percentage of 2.466%.

The energy consumed by the microcontroller to change its system clock frequency is formulated as (3-56) in chapter 3 i.e.

$$E_{ch} = V_s \overline{I_{ch}} \Delta t_{ch} \quad (3-56)$$

In VELOX-I application, $E_{ch} = 0.0Ws$. It implies that for this application, the microcontroller change of system clock frequency is not restricted by energy consumption cost.

4.3.3. Time Estimator

The flight software of VELOX-I is soft-deadline. In the software, there are $n_f = 18$ foreground tasks, $n_{be} = 4$ event-dependent background tasks, and $n_{bt} = 1$ time-dependent background task as listed in Table 4-5,

Table 4-5: VELOX-I's Task List

No	Task's Name	Task's Type & Period	No	Task's Name	Task's Type & Period
1	Second Updates	Foreground Period: 1000ms	13	Zigbee mission handler	Foreground Period: 1373ms
2	Meta Handler	Foreground Period: 1ms	14	GPS mission handler	Foreground Period: 1243ms
3	Get PWRS subsystem housekeeping	Foreground Period: 503ms	15	Real-Time Clock (RTC) time synchronization	Foreground Period: 6009ms
4	Get ADCS subsystem housekeeping	Foreground Period: 3907ms	16	Powering physics payload mission handler	Foreground Period: 1631ms
5	Get COMM subsystem housekeeping	Foreground Period: 1473ms	17	Debugging data handler	Foreground Period: 107ms
6	Get xdata segments	Foreground Period: 283ms	18	Physics payload mission handler	Foreground Period: 1000ms

7	Wired Uplink handler	Foreground Period: 47ms	19	System Management Bus (SMBus) driver	Background Event- Dependent
8	Radio Uplink handler	Foreground Period: 827ms	20	Serial Peripheral Interface (SPI) driver	Background Event- Dependent
9	Command handler	Foreground Period: 137ms	21	UART0 driver	Background Event- Dependent
10	Schedule handler	Foreground Period: 1021ms	22	UART1 driver	Background Event- Dependent
11	Downlink handler	Foreground Period: 139ms	23	Timer 2 interrupt handler	Background Time- Dependent Period: 1ms
12	Imaging mission handler	Foreground Period: 967ms			

4.3.3.1. Obtaining Time Estimator Parameters

To obtain the time estimator parameters, the flight software is run on different level of system clock frequencies, an hour in each level. For each level, every foreground and time-dependent background tasks is run according to their specified period. While the tasks are being run, the maximum execution time taken by each task within a second (for a task may be run more than once every second) is sampled. The samples are then used to determine the time estimator parameters.

Following the soft-deadline nature of the motherboard's software, the average (instead of the maximum) of the 3,600 data samples are calculated for each task in each system clock frequency level. The averaged data are then modeled by using the least-square first order fitting method ((4-7)-(4-9)). The coefficients of the task's time estimator model are obtained as shown in Table 4-6

Table 4-6: Motherboard's Tasks' Time Estimator's Coefficients

<u>Foreground Tasks' Time Estimator's Coefficients</u>							
<u>Task Code & Symbol</u>		a_{1f}	a_{2f}	<u>Task Code & Symbol</u>		a_{1f}	a_{2f}
Second Upd	T_{f1}	7.658797	0.078057	Sched	T_{f10}	3.449174	0.005371
Meta	T_{f2}	8.104276	0.036177	Downlink	T_{f11}	9.624846	0.000000
PwrsHk	T_{f3}	5.692927	0.000000	Camera	T_{f12}	6.109117	0.000000
AdcsHk	T_{f4}	2.174681	0.000000	Xbee	T_{f13}	3.896999	0.000000
CommHk	T_{f5}	5.549421	0.000000	GPS	T_{f14}	5.429027	0.000000
XdataGet	T_{f6}	10.018988	0.000000	Rtc	T_{f15}	1.282428	0.000324
WireUp	T_{f7}	2.492916	0.002402	PwrCqt	T_{f16}	0.458663	0.001258
RadioUp	T_{f8}	8.564837	0.000000	Debug	T_{f17}	162.970609	0.175587
Cmd	T_{f9}	2.079918	0.001805	Cqt	T_{f18}	85.084379	0.098832
<u>Event-Dependent Background Tasks' Time Estimator's Coefficients</u>							
<u>Task Code & Symbol</u>		a_{1be}	a_{2be}	<u>Task Code & Symbol</u>		a_{1be}	a_{2be}
SMBus	T_{be1}	0.125209	0.000000	UART0	T_{be3}	0.131878	0.000077
SPIO	T_{be2}	0.247977	0.000000	UART1	T_{be4}	0.119508	0.000229
<u>Time-Dependent Background Tasks' Time Estimator's Coefficients</u>							
<u>Task Code & Symbol</u>		a_{1bt}	a_{2bt}	$T_{btp}(ms)$			
Timer2	T_{bt1}	0.110174	0.000000	1.000000			

The time estimator parameters' model error and graphical results are shown in more detailed in Appendix 2 of this thesis.

The time estimator parameters in the $\mathbf{C}$ matrix shown in (3-6) and (3-7) can obtained based on the system design. Each element in $\mathbf{C}$ matrix is the maximum amount of call can be made by each software foreground task to each of the event-dependent background task in one loop. Table 4-7 shows the $\mathbf{C}$ matrix used for the motherboard's time estimator's parameters.

Table 4-7: Motherboard's Time Estimator's C Matrix

<i>C matrix</i>					
<i><u>Task Code & Symbol</u></i>		SMBus	SPIO	UART0	UART1
		T_{be1}	T_{be2}	T_{be3}	T_{be4}
Second Upd	T_{f1}	0	650	0	0
Meta	T_{f2}	0	0	0	0
PwrsHk	T_{f3}	50	0	0	0
AdcsHk	T_{f4}	42	0	0	0
CommHk	T_{f5}	16	0	0	0
XdataGet	T_{f6}	124	0	0	0
WireUp	T_{f7}	0	0	0	0
RadioUp	T_{f8}	12	0	0	0
Cmd	T_{f9}	0	0	0	0
Sched	T_{f10}	0	0	0	0
Downlink	T_{f11}	86	0	0	0
Camera	T_{f12}	114	0	0	0
Xbee	T_{f13}	36	0	0	0
GPS	T_{f14}	110	0	0	0
Rtc	T_{f15}	20	0	0	0
PwrCqt	T_{f16}	10	0	0	0
Debug	T_{f17}	0	0	0	247
Cqt	T_{f18}	0	0	225	0

4.3.3.2. Optimization of the Time Estimator Parameters

It can be seen from Table 4-6 that the system has significant portion of completely dependent tasks (12 out of 21) but fairly wide range of parameter values (0.000077 to 162.970909). Thus, the time estimator parameters can be optimized by *Off-line Static Parameter Computation* (see section 4.2.2.2) and *Completely-Dependent Tasks Identifiers* (see section 4.2.2.3) methods. To optimize the time estimator using *Truncation and Scaling* (see section 4.2.2.4) might not be so desirable due to its fairly wide range of parameter values, unless resolutions are compromised. Therefore, *Off-line Static Parameter Computation* and *Completely-Dependent Tasks Identifiers* optimization are implemented.

4.4. *Conclusions*

In this chapter, the time estimator and the energy estimator modules based on mathematical model presented in chapter 3 have been used to design a system clock frequency controller. A representative case based on VELOX-I nano-satellite's motherboard has been used for illustration. In the next chapter, the experimental results of the implementation will be shown.

5. Experimental Results

5.1. Experimental Settings

To test the performance of the proposed system clock frequency controller, five simulated orbital scenarios shown in Table 5-1 are used. Each orbital scenario is run for 5,700 seconds (95 minutes), simulating the typical orbital period for Low Earth Orbit (LEO) satellites.

Table 5-1: Simulated Orbital Scenarios

Duration (second)								System	
Orbit No	Scenario Name	Load Level						Clock Frequency Level	
		1	2	3	4	5	6	Min	Max
1	Power Safe-hold	2,100	3,000	600				2	4
2	Normal Operation		1,500	3,600	600			3	5
3	Normal Operation + Ground pass			4,800	900			4	5
4	Mission + Ground pass			3,600	1,200	900		4	6
5	Multiple Missions + Ground pass			2,700	1,200	900	900	4	7

In a typical miniaturized satellite operation, four modes are usually present: Safe-Hold, Normal, Ground Pass, and Mission. In the Safe-Hold mode, the satellite is recharging its batteries and minimizing its power consumption. In this mode, only the necessary components are turned on. Consequently, only some tasks are enabled in this mode, leaving the micro-controller with the least load level. In the Normal mode, the satellite performs its normal house-keeping operation. In this mode, if the energy balance is maintained, the satellite always gets positive energy gain in its orbit. This is the mode where the satellite will operate the most. It typically consist of sun-tracking during the non-eclipse period and standard housekeeping data collection. In this mode, the satellite also sends beacon: a strong but short-messaged signal (usually in the form of Morse code) which tells the minimum information about the satellite's condition. The Ground Pass mode occurs when the satellite establishes contact with the

ground station. In this mode, the satellite will receive the uplink command from and send downlink data to the ground station. The ground pass which allows the ground station to make contact with the satellite typically last for 10-14 minutes depends on the satellite's orbit path. The satellite may pass the ground station contact range without being able to establish communication if the elevation angle of the satellite (with respect to the ground station) is too low (typically less than 10 degrees). The mission mode is when the satellite is required to carry out its "mission", that is to operate the payloads of the satellite. VELOX-I satellite is designed to support three payloads, camera (to capture image), GPS (to get GPS data), and Zigbee Wireless Sensor Networks (Zigbee WSN, to test inter-satellite communication in space). Executing the missions of a satellite typically consume the most power and thus must be carefully planned before the actual execution as not to waste the satellite resources.

To create different orbital scenarios in relation to the describe modes, a set of tasks which produce different load levels are made for different orbital scenarios. The different loads of the satellite are then further represented by different sets of enabled/disabled tasks. It should be noted that while the tasks are *enabled*, they need not be *activated* all the time. Disabling a task means forbidding it to be activated. But enabling a task means letting it to be activated when its activation conditions are met (such as due to events or periodical call). Table 5-2 shows different task settings on different loads.

By making the enabled tasks running according to its schedule and specifying system time performance requirement t_{req} , the tasks' dynamics will result in different frequency requirements f_{req} for each given set of tasks in a given instance. It will then determine different minimum and maximum system clock frequency level requirements as shown in Table 5-1.

Table 5-2: Simulated Tasks on Different Loads

Task Name	Task Symbol	Enabled (1) / Disabled (0)					
		Load 1	Load 2	Load 3	Load 4	Load 5	Load 6
Second Upd	Tf1	1	1	1	1	1	1
Meta	Tf2	1	1	1	1	1	1
PwrsHk	Tf3	1	1	1	1	1	1
AdcsHk	Tf4	0	1	1	1	1	1
CommHk	Tf5	0	1	1	1	1	1
XdataGet	Tf6	1	1	1	1	1	1
WireUp	Tf7	0	1	1	1	1	1
RadioUp	Tf8	0	0	1	1	1	1
Cmd	Tf9	0	0	0	1	1	1
Sched	Tf10	0	0	0	1	1	1
Downlink	Tf11	0	0	1	1	1	1
Camera	Tf12	0	0	0	0	0	1
Xbee	Tf13	0	0	0	0	1	1
GPS	Tf14	0	0	0	0	0	1
Rtc	Tf15	1	1	1	1	1	1
PwrCqt	Tf16	0	0	0	0	1	1
Debug	Tf17	1	1	1	1	1	1
Cqt	Tf18	0	0	0	0	1	1

For the experiment, $t_{req} = 8.2$ milliseconds is used. The input is chosen to provide wide range of frequency requirements in the simulation. The number of samples n_s is set as 10, following the housekeeping period requirement of VELOX-I, which is 10 seconds. Ten samples in 10 seconds gives the proposed controller sampling interval $\Delta t_e = 1$. The time performance parameters for soft-deadline system, E_{tm} (3-50) and M_{WCET} (3-52), are therefore to be calculated based on the last $n_s = 10$ samples' values.

5.2. Benchmarking

As presented in chapter 2, there are two common approaches to program the system clock of a microcontroller. One is a *fixed operating frequency method* and the other is a *mode-based operating frequency method*. In the *fixed method*, one chooses a system clock frequency and works with it. In the *mode-based method* one chooses a system clock frequency based on its current mode.

Table 5-3 summaries a comparison of the *fixed operating frequency method*, *mode-based operating frequency method*, and the proposed controller in various aspects.

Table 5-3: Comparison of Fixed Method, Mode Based Method, and the Proposed Controller

No	Aspect	Fixed Method	Mode Based Method	Proposed Controller
1	Energy use	Non-optimized	Partially optimized	Optimized
2	Design complexity	Low, choose one maximum frequency	Medium, choose multiple frequencies for different modes	High, create the controller
3	Overclocking risk	Present	Present	Not present
4	Autonomy	Non-autonomous	Partially- autonomous	Autonomous

For a fair comparison, the three methods are compared under the same requirements and scenarios. In the *fixed operating frequency method*, the maximum system clock frequency required among all the scenarios (i.e. system clock frequency level 7) would be applied to the satellite in all orbital scenarios. In the *mode-based operating frequency method*, the minimum system clock frequency required in each orbital scenario to satisfy the time requirement will be applied. In the *proposed controller*, the system clock frequency level used is not fixed based on the orbital scenario. Rather, it will adjust its system clock frequency automatically based on the tasks' dynamics.

The current drawn for a level of system clock frequency is known to be nearly constant. Before performing the experiment for benchmarking with the two conventional methods, the microcontroller was first subjected to different levels of system clock frequency while its average current consumption is measured for each fixed system clock frequency. The measurement result is shown in Table 4-4. The controller is set such that it does not change the tasks' periods or the tasks' settings (which ones are enabled and which ones are not) under all orbital scenarios. This way, the controller will not introduce additional computational load when compared to the original settings. Rather, the controller will only be taking the "free time" of the microcontroller (which would otherwise be spent in "idle" by the other two conventional methods) to do its job rather than introducing additional load. By doing so, and using the result shown in Table 4-4, the power consumption of the two conventional methods

can then calculated together with the proposed controller using Table 4-4 in a single experiment.

Table 5-4 summarizes the system clock frequency levels used by the three approaches in different orbital scenarios. To provide a standard reference for energy measurement, the average current drawn per second for different methods in all the scenarios shown in Table 5-4 are measured using the same system current–system clock frequency relationship shown in Table 4-4.

Table 5-4: Microcontroller System Clock Frequency Level Applied for the Various Orbital Scenarios by Fixed-Method, Mode-Based Method, and the Proposed System Clock Frequency Controller

Orbit No	Scenario Name	System Clock Frequency Level		
		Fixed Method	Mode-Based Method	Proposed Controller
1	Power Safe-hold	7	4	Varying
2	Normal Operation	7	5	Varying
3	Normal Operation + Ground pass	7	5	Varying
4	Mission + Ground pass	7	6	Varying
5	Multiple Missions + Ground pass	7	7	Varying

5.3. Time Performance

The flight software is run for 95 minutes per orbital scenario as described in Table 5-1. Maximum execution time of the software in one loop (t_s) and the predicted execution time (t_{tot}) by the time estimator are sampled every second. The time model error (E_{tm}) and the execution time margin (M_{WCET}) are also calculated by using (3-46) and (3-51) for the last $n_s = 10$ samples.

Figure 5-1: System Time Performance Graph (First 5 Minutes) for Orbital Scenario 1

Figure 5-2: System Time Performance Graph (First 5 Minutes) for Orbital Scenario 5

Figure 5-1 and Figure 5-2 show the system time performance of the proposed controller in the first 5 minutes of the orbital scenario 1 (the least heavy load) and the orbital scenario 5 (the heaviest load) respectively. It can be seen from the figures that the summation of the actual execution time $\sum_{i=1}^{i=n_s} t_{si}$ over the last $n_s = 10$ samples show almost periodic-like graph due to periodical nature of the system.

It can be observed from the figures that the time model error (E_{tm}) is smaller (1.52 ms on average) when the system is run under orbital scenario 1 than when the system is run under orbital scenario 5 (4.00 ms on average). It can also be observed that the execution time margin (M_{WCET}) over the last $n_s = 10$ samples in the experiment on the orbital scenario 1 (range: 12.00 ms to 33.00 ms) is more stable than in the experiment on the orbital scenario 5 (range: 13.00 ms to 56.00). This observation is consistent with the nature of the orbital scenario 1 as

compared to the orbital scenario 5, in which the earlier is enabling less amount of tasks than the latter as shown in Table 5-2.

The graphs showing the system time performance dynamics of other orbital scenarios can be found in Appendix 3.

Table 5-5 shows the complete time performance statistical data obtained from the experiment for all orbits

Table 5-5: Proposed Controller System's Time Performance Statistics

Time Performance Statistics (ms)									
Orbit No	Scenario Name	$\sum_{i=1}^{i=n_s} t_{si}$				E_{tm}			
		Min	Mean	Max	Stdev	Min	Mean	Max	Stdev
1	Power Safe-hold	49.00	62.11	70.00	5.18	0.00	1.52	6.31	1.13
2	Normal Operation	30.00	52.26	71.00	8.65	0.00	3.40	15.41	2.76
3	Normal Operation + Ground pass	30.00	48.83	69.00	9.18	0.00	3.41	15.25	2.85
4	Mission + Ground pass	30.00	47.02	68.00	8.47	0.00	3.44	17.64	2.72
5	Multiple Missions + Ground pass	26.00	44.36	69.00	9.33	0.00	4.00	15.55	2.97
Orbit No	Scenario Name	$\sum_{i=1}^{i=n_s} t_{toti} - \sum_{i=1}^{i=n_s} t_{si}$				M_{WCET}			
		Min	Mean	Max	Stdev	Min	Mean	Max	Stdev
1	Power Safe-hold	7.53	15.17	26.74	3.67	12.00	19.89	33.00	NA
2	Normal Operation	-0.58	22.47	49.22	10.14	11.00	29.74	52.00	NA
3	Normal Operation + Ground pass	4.23	25.19	43.57	10.12	13.00	33.17	52.00	NA
4	Mission + Ground pass	2.26	26.94	44.01	9.99	14.00	34.98	52.00	NA
5	Multiple Missions + Ground pass	2.26	28.58	46.53	9.51	13.00	37.64	56.00	NA

Table 5-5 shows the most important time performance parameters, E_{tm} and M_{WCET} . The experimental results show that the mean values of E_{tm} range from 1.52 ms to 4.00 ms in various orbital scenarios (or 2.447% to 9.025% with respect to the mean of $\sum_{i=1}^{i=n_s} t_{s_i}$ in the respective orbital scenarios). The minimum value of $E_{tm} \rightarrow 0$ is shown all orbital scenario tests, while the maximum value of $E_{tm} = 17.64$ (or 37.051% with respect to the mean of $\sum_{i=1}^{i=n_s} t_{s_i}$ in the same scenario) is shown in the fourth orbital scenario test.

The M_{WCET} values from the experimental results are positive in all scenarios. The minimum value of $M_{WCET} = 11$ ms occurs in the second orbital scenario and the maximum value of $M_{WCET} = 56$ ms occurs in the fifth orbital scenario. Therefore, the proposed approach satisfies the system time requirement $n_s m_{WCET} = 82$ ms with a minimum margin of 11 ms in all given scenarios at all time. The mean values of M_{WCET} range 19.89 ms to 37.64 ms (or 32.031% to 84.862% with respect to the mean of $\sum_{i=1}^{i=n_s} t_{s_i}$ in the respective orbital scenarios and 24.256% to 45.902% with respect to $n_s m_{WCET}$) showing, on average, how close the actual execution time from the anticipated WCET modeled by the proposed controller system is.

More statistical and the experimental results are shown in Appendix 3.

5.4. Frequency Controller Responses

To monitor the frequency controller response as the flight software is run under various orbital scenario as described in Table 5-1, the calculated system clock frequency requirement (f_{req}), the actual system clock frequency (f_{sorig}), the maximum target frequency which incur negative energy gain ($f_{starg_{max}}$), and the desired/target system clock frequency (f_{sr}) are sampled every second.

Figure 5-3 and Figure 5-4 show the frequency controller response for the first 5 minutes in the orbital scenario 1 and the orbital scenario 5 respectively.

Figure 5-3: Frequency Controller Response Graph (First 5 Minutes) for Orbital Scenario 1

Figure 5-4: Frequency Controller Response Graph (First 5 Minutes) for Orbital Scenario 5

From the figures, we can observe that $f_{starg_max} \leq f_{req} \leq f_{sr}$ is maintained all the time. Moreover, f_{sorig} follows f_{sr} closely as f_{sr} changes. These show how the designed frequency controller response to the task dynamics while satisfying user's time requirement and optimizing minimizing the energy use at the same time. As shown in section 4.2.4, while the system is underclock, as earlier, the controller would increase the system clock frequency. On the other hand, when it is overclock, it would decrease the system clock frequency while maintaining positive energy gain ($f_{starg_max} \leq f_{req} \leq f_{sr}$). Thus, we can see from the experimental results that the frequency controller in the system work as desired.

5.5. Energy Consumption

To monitor the energy usage of the VELOX-I's motherboard with the proposed controller and to benchmark it with the other two methods, the motherboard's average current drawn ($I_{controller}$) is sampled at every second. For the other two conventional methods, the current drawn by *fixed operating frequency method* (I_{fixed}) and *mode-based operating frequency method* (I_{mode_based}) are calculated by using Table 4-4 and their respective system clock frequency per orbital scenario as shown in Table 5-4. Using the actual frequency requirement (f_{req}), the minimum system current ($I_{optimal}$) is also calculated to provide comparison between the proposed controller and the actual optimal solution.

Figure 5-5 and Figure 5-6 show the current drawn by the system using various methods for the first 5 minutes in the orbital scenario 1 and the orbital scenario 5 respectively.

Figure 5-5: The Comparison of the Motherboard's Average Current Drawn per Second for the First 5 Minutes of the Orbital Scenario 1 for Different Methods

Figure 5-6: The Comparison of the Motherboard's Average Current Drawn per Second for the First 5 Minutes of the Orbital Scenario 5 for Different Methods

From the figures, it is observed that I_{fixed} is maintained in the presented orbital scenarios shown (results for orbital scenario 2, 3, and 4 are presented in appendix 3, whose results tally with the figures) and I_{mode_based} is maintained within an orbital scenario, while $I_{controller}$ responds to the task dynamics within an orbital scenario. It is also observed that while I_{fixed} and I_{mode_based} differ in orbital scenario 1, there is no different in orbital scenario 5. $I_{controller}$ responds to the task dynamics in both scenario. This shows the difference between the *mode-based operating frequency method* which is partially-optimized within its available operating frequency levels and the proposed controller which is fully optimized within its available operating frequency levels. The result of $I_{controller}$ is closer to $I_{optimal}$ orbital scenario 1 than in orbital scenario 5. This shows how task dynamics and the available operating frequency levels affect the optimization of the proposed controller as compared to the most optimal solution, which assumes availability of continuous system clock frequency levels.

The overall energy consumption for all orbital scenarios obtained from the experiment is shown in Table 5-6, while its graphical representations are shown in Figure 5-7 to Figure 5-9.

Table 5-6: Energy/Power Experimental Data of the Proposed Controller Compared to the Fixed-Method, Mode-Based-Method, and the Use of Optimal System Clock Frequency in Various Orbital Scenarios

	Energy Consumption (mWh)				Energy Reduced (mWh) with respect to Fixed-Method			
	Fixed	Mode Based	Controlled	Optimal	Fixed	Mode Based	Controlled	Optimal
Orbit 1	791.94	447.43	338.12	321.58	0.00	344.50	453.82	470.36
Orbit 2	791.81	570.10	433.80	402.78	0.00	221.71	358.01	389.03
Orbit 3	791.67	570.00	463.33	429.01	0.00	221.67	328.33	362.66
Orbit 4	791.12	656.63	491.84	452.30	0.00	134.49	299.28	338.82
Orbit 5	790.97	790.97	550.82	494.54	0.00	0.00	240.15	296.43
Average	791.50	607.03	455.58	420.04	0.00	184.47	335.92	371.46
	Energy Reduced Percentage (%) with respect to Fixed-Method				Average Power Reduced (mW) with respect to Fixed-Method			
	Fixed	Mode Based	Controlled	Optimal	Fixed	Mode Based	Controlled	Optimal
Orbit 1	0.00%	43.50%	57.31%	59.39%	0.00	217.58	286.63	297.07
Orbit 2	0.00%	28.00%	45.21%	49.13%	0.00	140.02	226.11	245.70
Orbit 3	0.00%	28.00%	41.47%	45.81%	0.00	140.00	207.37	229.05
Orbit 4	0.00%	17.00%	37.83%	42.83%	0.00	84.94	189.02	213.99
Orbit 5	0.00%	0.00%	30.36%	37.48%	0.00	0.00	151.67	187.22
Average	0.00%	23.30%	42.44%	46.93%	0.00	116.51	212.16	234.61

Figure 5-7: Energy Consumption (in mWh) of the Proposed Controller Compared to the Fixed-Method, Mode-Based-Method, and Optimal System Clock Frequency in Various Orbital Scenarios

Figure 5-8: Energy Reduced (in %) of the Proposed Controller Compared to the Fixed-Method, Mode-Based-Method, and the Use of Optimal System Clock Frequency in Various Orbital Scenarios with respect to the Fixed-Method

Figure 5-9: Average Power Reduced (in mW) of the Proposed Controller Compared to the Fixed-Method, Mode-Based-Method, and the Use of Optimal System Clock Frequency in Various Orbital Scenarios with respect to the Fixed-Method

Table 5-6 and Figure 5-7 to Figure 5-9 show that the proposed method uses energy more efficiently to perform the same tasks than the conventional methods in all orbital scenarios. Compared to the fixed-method, the proposed system consumes 30.36% to 57.31% less energy under the various orbital scenarios, while the mode-based method consumes 0.00% to 43.5% less energy. On average, the proposed system consumes 42.44% and 23.30% less energy than the fixed-method and mode-based method respectively. Table 5-7 summarizes the energy reduced percentage difference for various methods

Table 5-7: Energy Reduction Percentage Difference between the Mode-Based Method, the Proposed Controller, and the Optimal System Clock Frequency

Energy Reduction Percentage Difference (%)			
Method 1	Optimal	Optimal	Proposed Controller
Method 2	Mode Based	Proposed Controller	Mode Based
Orbit 1	15.892%	2.088%	13.80%
Orbit 2	21.132%	3.918%	17.21%
Orbit 3	17.810%	4.336%	13.47%
Orbit 4	25.827%	4.997%	20.83%
Orbit 5	37.477%	7.116%	30.36%
Average	23.628%	4.491%	19.137%

In comparison to the mode-based method, the proposed system consumes on average 19.137% less energy under various simulated orbital scenarios. The highest difference of 30.36% occurs in the orbit 5 scenario, where the range of the system clock frequency level requirement varying from 4 to 7, and is the widest among the simulated scenarios. While the lowest difference of 13.47% occurs in the orbit 3 scenario, where the range of the system clock frequency level requirement ranging from 4 to 5, is the least varying among the simulated scenarios.

When benchmarked with the optimal system clock frequency, the proposed system consumes on average 4.491% more energy than the optimal. The lowest difference of 2.088% occurs in the orbit 1 scenario and the highest difference of 7.116% occurs in the orbit 5 scenario. For brevity, other experimental results are given in Appendix 3.

In summary, the proposed controller implemented in VELOX-I nano-satellite's motherboard has been shown to work and to response to tasks' dynamics in all orbital scenarios. It has also been shown that the proposed controller outperform the conventional methods in all given scenarios, regardless the variation of the tasks. Unlike the conventional methods, the energy consumed by the proposed controller has also been shown to be close to the optimal solution in all given scenarios while satisfying the time requirement. It cannot attain the optimal solution only because it is limited by the hardware capability.

6. Conclusions and Future Works

6.1. Conclusions

In low power applications such as pico- or nano-satellites, where one Watt could be the entire power budget of the satellite, a power efficient design is important. While there are research works in the pico- and nano-satellite to overcome the extreme power limitation by reducing the power consumption using various approaches, this work minimizes the power consumption by optimal use of microcontroller's system clock frequency.

This thesis has presented the system modeling, the design and implementation, as well as the experimental results of the proposed controller system. It uses both the time and energy estimators to determine an optimal frequency under specified time performance requirements and energy constraints. Tested under various orbital scenarios, the proposed system has shown higher energy efficiency as compared to the conventional methods such as the *fixed operating frequency method* and *mode-based operating frequency method*. From the experimental results, the proposed controller has on average of 42.44% and 19.137% energy saving as compared to the *fixed operating frequency method* and the *mode-based operating frequency method* respectively

Unlike the conventional methods, the proposed controller has autonomous behavior. It adjusts its system clock frequency while fulfilling the system time performance requirement (i.e. maintaining $M_{WCET} \geq 0$ at all time). The experimental results show that the proposed controller fulfill its designed requirement to optimize the system clock frequency level of VELOX-I while guaranteeing the user time requirement.

6.2. Future Works

There are some remaining works could be done in the future to improve the proposed system clock frequency controller.

The current work focuses on implementing an autonomous system clock frequency controller in a single microcontroller, while a miniaturized satellite often contains multiple microcontrollers. Thus, there is an opportunity for greater energy reduction by investigating multiple

microcontrollers' optimization using the proposed autonomous system clock frequency controller.

As multiple microcontrollers are being considered, more design complexity is expected due to different devices' requirements. For instance, the current implementation in the VELOX-I nano-satellite's motherboard falls under the category of soft-deadline system, as there is no other application that requires it to response under a specified response time requirement. But the inclusion of other subsystems' microcontrollers could make the entire system a combination of soft-deadline and hard-deadline system.

The system clock frequency controller investigated in this work is designed with autonomous property. This makes the design optimal in the sense that, given set of safe levels of system clock frequencies, the system time requirement would be maintained in the worst case scenario based only on off-line static time parameters and on-line dynamic binary task's parameters. The equations (4-1) to (4-5) ensure the proposed method to be fully optimize for energy saving, since it chooses the lowest possible frequency level which still result in positive energy gain among the lists of the frequency level that it has. It is not optimal in the sense that the controller does not apply the computational resources most efficiently, as close as possible to the dynamic requirement. Furthermore, it is also not optimal in the sense that it really chooses minimum frequency possible to satisfy the time constraint for every set of tasks due to its necessity to retain its peripherals speed within acceptable error rates (thus some frequencies, though optimal, may produce unacceptable error rates to the peripherals speed when applied).

Moving from autonomous controller to dynamic controller would require feedback of dynamic time parameters during run-time. Thus, there is a trade-off between optimal performance and computational requirements that could be investigated in future.

Bibliography

- [1] M. Peck, "Chips in Space," *IEEE Spectrum*, vol. 48, pp. 42-47, 2011.
- [2] Wikipedia. (2013, 24 July). *Miniaturized Satellite*. Available: https://en.wikipedia.org/wiki/Miniaturized_satellite
- [3] Y. Xue, Y. Li, J. Guang, X. Zhang, and J. Guo, "Small Satellite Remote Sensing and Applications – History, Current and Future," *International Journal of Remote Sensing*, vol. 29, 2008.
- [4] I. Nason, J. Puig-Suari, and R. Twiggs, "Development of a Family of Pico-Satellite Deployers Based on the CubeSat Standard," in *Proc. IEEE Aerospace Conf.*, 2002, pp. 1-457-1-464.
- [5] N. T. Univerisy. (2014). *Satellite Research Centre*. Available: <http://www.sarc.eee.ntu.edu.sg/Pages/Home.aspx>
- [6] A. L. M. Craig S. Clark, "Power System Challenges for Small Satellite Missions," in *45 Symp.*, Chia Laguna, Sardinia, 2006.
- [7] M. H. Heidt, P. J. Puig-Suari, P. A. S. Moore, P. S. Nakasuka, and P. R. J. Twiggs, "CubeSat: A New Generation of Picosatellite for Education and Industry Low-Cost Space Experimentation," in *14th Annual AIAA/USU Conf. on Small Satellites*, Utah State University, 2000.
- [8] D. L. Bekker, T. A. Werne, T. O. Wilson, P. J. Pingree, K. Dontchev, M. Heywood, *et al.*, "A CubeSat Design to Validate the Virtex-5 FPGA for Spaceborne Image Processing," in *IEEE Aerospace Conf.*, 2010, pp. 1-9.
- [9] C. Angadi, Z. Manjiyani, C. Dixit, K. Vigneswaran, G. S. Avinash, P. Raj Narendra, *et al.*, "STUDSAT: India's First Student Pico-Satellite Project," in *IEEE Aerospace Conf.*, 2011, pp. 1-15.
- [10] J. S. Robert Hoyt, Robert Twiggs, "The Multi-Application Survivable Tether (MAST) Experiment," in *39th AIAA/ASME/SAE/ASEE Joint Propulsion Conf. and Exhibit*, Huntsville, Alabama, 2003.
- [11] R. Burt, "Distributed Electrical Power System in Cubesat Applications," Master of Science, Electrical Engineering, Utah State, Logan, Utah, 2011.
- [12] J. A. Schaffner, "The Electronic System Design, Analysis, Integration, and Construction of the Cal Poly State University CP1 CubeSat," in *16th AIAA/USU on Small Satellites Conf.*, 2002, pp. 1-2.
- [13] E. S. Agency. (2013, 15 August). *Delfi-n3Xt (Delfi Triple CubeSat NeXt) mission*. Available: <https://directory.eoportal.org/web/eoportal/satellite-missions/d/delfi-n3xt>
- [14] C. S. Clark, Alejandro Lopez Mazarias, Chisato Kobayashi and Shinichi Nakasuka, "The Design of a Power System for the PETSAT Modular Small Spacecraft Bus," in *European Small Satellite Services Symp.*, Rhodes, 2008.
- [15] A. Caon, L. Brambilla, R. Contini, G. D'Accolti, E. Rossi, G. Verzeni, *et al.*, "The GaAs Deployable Solar Array for SAC-B," in *1st IEEE World Conf. on Photovoltaic Energy Conversion*, 1994, pp. 2014-2017.
- [16] R. Trabert, Andrew Klesh, Patrick Senatore, Patrick Martinchek, Daniel Becker, Andrew Chou, Clark Hoffman, Nathan McKay, Jeremy Nash, Jeffery Walters, James Cutler, Darren McKague, "The eXtendable Solar Array System: A Modular Nanosatellite Power System," in *AIAA/AAS Astrodynamics Specialist Conf.*, Toronto, Ontario Canada, 2010.
- [17] N. McKay, Andrew Chou, Daniel Becker, Clark Hoffman, Jeffery Walters, "Research and Development of the eXtendable Solar Array System," in *50th AIAA Aerospace Sciences Meeting*, Nashville, Tennessee, 2012.

- [18] F. P. Fabio Santoni, Serena Donatia, Massimo Perelli, Andrea Negri, Michele Marino, "An Innovative Deployable Solar Panel System for Cubesats," *Acta Astronautica*, vol. 95, pp. 210-217, 2014.
- [19] Z. Varadi, V. Gorocz, and J. Szabo, "Power Subsystem for Small Satellites with Unified Operational Mode Controller," in *2nd Int'l Conf. on Space Technology (ICST '11)*, 2011, pp. 1-4.
- [20] Z. Wu, Z. You, and J. Salvignol, "Tsinghua-1 Micro-Satellite Power System Architecture and Design," in *Proc. 3rd Int'l Power Electronics and Motion Control Conf. (IPEMC '00)*, 2000, pp. 1045-1049.
- [21] F. A. O. Aashoor and F. V. P. Robinson, "A Variable Step Size Perturb and Observe Algorithm for Photovoltaic Maximum Power Point Tracking," in *47th Int'l Universities Power Engineering Conf. (UPEC '12)*, 2012, pp. 1-6.
- [22] D. S. Karanjkar, S. Chatterji, S. L. Shimi, and A. Kumar, "An Improved Current Feedback Based Maximum Power Point Tracking Controller for Solar Photo-Voltaic System," in *Annual Int'l Conf. on Emerging Research Areas and Int'l Conf. on Microelectronics, Communications and Renewable Energy (AICERA/ICMiCR '13)*, 2013, pp. 1-6.
- [23] O. Yangjing, W. Chenghua, and H. Feng, "A Variable Step Maximum Power Point Tracking Method Using Taylor Mean Value Theorem," in *Asia-Pacific Power and Energy Engineering Conf. (APPEEC '10)*, 2010, pp. 1-4.
- [24] B. Tarek, D. Said, and M. E. H. Benbouzid, "Maximum Power Point Tracking Control for Photovoltaic System Using Adaptive Neuro-Fuzzy ANFIS," in *8th Int'l Conf. and Exhibition on Ecological Vehicles and Renewable Energies (EVER '13)*, , 2013, pp. 1-7.
- [25] Y. P. Siwakoti, B. B. Chhetri, B. Adhikary, and D. Bista, "Microcontroller Based Intelligent DC/DC Converter to Track Maximum Power Point for Solar Photovoltaic Module," in *IEEE Conf. on Innovative Technologies for an Efficient and Reliable Electricity Supply (CITRES '10)*, 2010, pp. 94-101.
- [26] M. Veerachary, "Fourth-Order Buck Converter for Maximum Power Point Tracking Applications," *IEEE Trans. on Aerospace and Electronic Systems*, vol. 47, pp. 896-911, 2011.
- [27] N. Femia, G. Petrone, G. Spagnuolo, and M. Vitelli, "Optimization of Perturb and Observe Maximum Power Point Tracking Method," *IEEE Trans. on Power Electronics*, vol. 20, pp. 963-973, 2005.
- [28] Y. Levron and D. Shmilovitz, "Maximum Power Point Tracking Employing Sliding Mode Control," *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 60, pp. 724-732, 2013.
- [29] Y. Chih-Yu, H. Chun-Yu, F. Fu-Kuei, and C. Ke-Horng, "Highly Efficient Analog Maximum Power Point Tracking (AMPPT) in a Photovoltaic System," *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 59, pp. 1546-1556, 2012.
- [30] B. Mocanu, M. M. Burlacu, J. Kohlenberg, M. Prathaban, P. Lorenz, and R. Tapu, "Determining Optimal Orbital Path of a Nanosatellite for Efficient Exploitation of the Solar Energy Captured," in *1st Int'l Conf. on Advances in Satellite and Space Communications (SPACOMM '09)*, 2009, pp. 128-133.
- [31] K. Dong-In, S. Jinwoo, and S. P. Crago, "An Optimal Voltage Synthesis Technique for a Power-Efficient Satellite Application," in *Proc. 39th Design Automation Conf.*, 2002, pp. 492-497.
- [32] V. Y. Chu, W. Sun, and M. N. Sweeting, "Optimising the LEO Satellite Communications Link through Hybrid-ARQ Techniques," in *Proc. of Int'l Conf. on Information, Communications and Signal Processing (ICICS '97)*, 1997, pp. 5-9.
- [33] P. K. Singh and S. Jayasimha, "A Low-Complexity, Reduced-Power Viterbi Algorithm," in *Proc. 12th International Conf. on VLSI Design*, 1999, pp. 61-66.

- [34] W. W. Wu, D. Haccoun, R. Peile, and Y. Hirata, "Coding for Satellite Communication," *IEEE Journal on Selected Areas in Communications*, vol. 5, pp. 724-748, 1987.
- [35] D. Miyakawa and Y. Ishikawa, "Process Oriented Power Management," in *Int'l Symp. on Industrial Embedded Systems (SIES '07)*, 2007, pp. 1-8.
- [36] S. Baskaran, P. Thambidurai, "Power Aware Scheduling for Resource Constrained Distributed Real-Time Systems " *Int'l Journal on Computer Science and Engineering*, vol. 02, pp. 1746-1753, 2010.
- [37] N. Linwei and Q. Gang, "Energy minimization for real-time systems with (m,k)-guarantee," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 14, pp. 717-729, 2006.
- [38] C. Jian-Jia and K. Chin-Fu, "Energy-Efficient Scheduling for Real-Time Systems on Dynamic Voltage Scaling (DVS) Platforms," in *13th IEEE Int'l Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA '07)*, 2007, pp. 28-38.
- [39] W. Jinggang, B. Ravindran, and T. Martin, "A Power-Aware, Best-Effort Real-Time Task Scheduling Algorithm," in *IEEE Workshop on Software Technologies for Future Embedded Systems*, 2003, pp. 21-28.
- [40] D. Bautista, J. Sahuquillo, H. Hassan, S. Petit, and J. Duato, "A Simple Power-Aware Scheduling for Multicore Systems when Running Real-Time Applications," in *IEEE Int'l Symp. on Parallel and Distributed Processing (IPDPS '08)*, 2008, pp. 1-7.
- [41] S. Dongkun and K. Jihong, "Power-Aware Scheduling of Conditional Task Graphs in Real-Time Multiprocessor Systems," in *Proc. Int'l Symp. on Low Power Electronics and Design (ISLPED '03)*, 2003, pp. 408-413.
- [42] S. Euseong, J. Jinkyu, P. Seonyeong, and L. Joonwon, "Energy Efficient Scheduling of Real-Time Tasks on Multicore Processors," *IEEE Trans. on Parallel and Distributed Systems* vol. 19, pp. 1540-1552, 2008.
- [43] L. Benini, A. Bogliolo, and G. De Micheli, "A Survey of Design Techniques for System-Level Dynamic Power Management," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 8, pp. 299-316, 2000.
- [44] R. Peng and M. Pedram, "Power-Aware Scheduling and Dynamic Voltage Setting for Tasks Running on a Hard Real-Time System," in *Asia and South Pacific Conf. on Design Automation*, 2006, p. 6.
- [45] O. S. Unsal and I. Koren, "System-Level Power-Aware Design Techniques in Real-Time Systems," *Proceedings of the IEEE*, vol. 91, pp. 1055-1069, 2003.
- [46] C. Lee, T. Lin, and R. Chang, "Power-Aware Code Scheduling Assisted with Power Gating and DVS," *The International Journal of Grid Computing and E-Science*, vol. 34, pp. 66-75, 2014.
- [47] J. Mei, K. L. Li, J. T. Hu, S. Yin, and E. H. M. Sha, "Energy-Aware Preemptive Scheduling Algorithm for Sporadic Tasks on DVS Platform," *Microprocessors and Microsystems*, vol. 37, pp. 99-112, 2013.
- [48] R. Różycki and J. Węglarz, "Power-Aware Scheduling of Preemptable Jobs on Identical Parallel Processors to Minimize Makespan," *Annals of Operations Research*, vol. 213, pp. 235-252, 2014.
- [49] T. Bleir, Paul Clarke, Jamie Cutler, Louis DeMartini, Clark Dunson, Scott Flagg, Allen Lorenz, Eric Tapio, "QuakeSat Lessons Learned: Notes from the Development of a Triple CubeSat," QuakeFinder, LLC2004.
- [50] ON Semiconductor, "LC87F2C64A Datasheet," ed, 2013.
- [51] M. Verle, *PIC Microcontrollers - Programming in C, 1st Edition*: mikroElektronika, 2009.
- [52] Freescale Semiconductor Inc., "MPC5646C Microcontroller Reference Manual Rev. 4," ed, 2012.
- [53] Silicon Laboratories, "EFM32TG110 DATASHEET F32/F16/F8/F4," ed, 2013.

- [54] Cypress Smart Corporation, "EZ-USB FX2LP USB Microcontroller Datasheet Rev. K," ed, 2006.
- [55] Freescale Semiconductor Inc., "Internal Clock Source (ICS) Module on the HCS08s in Depth," ed, 2005.
- [56] ON Semiconductor, "LC88F52H0A 16-bit 1-chip Microcontroller Datsheet," ed, 2013.
- [57] Analog Devices Inc., "ADuC7023 Datasheet Rev. E," ed, 2013.
- [58] NXP Semiconductors, "LPC1102-1104 Product Datasheet Rev. 6," ed, 2012.
- [59] Freescale Semiconductor Inc., "MPC5510 Microcontroller Family Reference Manual Rev. 1," ed, 2008.
- [60] Texas Instruments, "Clocking Options for Stellaris Family Microcontrollers Application Note," ed, 2009.
- [61] Atmel Corporation, "AT89C51 Datasheet," ed, 2000.
- [62] ON Semiconductor, "LC87F2J32A Datasheet Ver 1.20," ed, 2013.
- [63] Texas Instruments, "MSP430F525x Mixed Signal Microcontroller Datasheet," ed, 2013.
- [64] Silicon Laboratories, "C8051F120/1/2/3/4/5/6/7 C8051F130/1/2/3 Mixed Signal ISP Flash MCU Family Datasheet," ed, 2005.
- [65] Atmel Corporation, "ATmega8535 and ATmega8535L Datahseet," ed, 2006.
- [66] A. L. Matthew Long, Greg Rodgers, Eric Tapio, Glenn Tran, Keoki Jackson and Robert Twiggs, "A CubeSat Derived Design for a Unique Academic Research Mission in Earthquake Signature Detection," in *16th Annual AIAA/USU Conf. on Small Satellites*, Utah, USA, 2002.
- [67] V. B. Radu Dan Lazar, Ales Loidl, Lukas Formanek, Thomas Chlubna, "Optimized Design of Power Supply for CubeSat at Aalborg University," Aalborg University, 2002.
- [68] M. C. Joe Bauer, Kaitlyn Kelley, Ernie Mello, Sam Neu, Alex Orphanos, Tim Shaffer, Andrew Withrow, "Mechanical, Power, and Thermal Subsystem Design for a CubeSat Mission," Worcester Polytechnic Institute, 2012.
- [69] S. Waydo, D. Henry, and M. Campbell, "CubeSat Design for LEO-Based Earth Science Missions," in *Proc. IEEE Aerospace Conf.*, 2002, pp. 1-435-1-445.
- [70] A. Rakshit and A. Chatterjee, "A Microcontroller-Based Compensated Optical Proximity Detector Employing the Switching-Mode Synchronous Detection Technique," *Measurement Science and Technology*, vol. 23, 2012.
- [71] D. F. Pastorello, D. Holberg, W. G. Durbin, B. Sahu, and G. R. Chowdhury, "Microcontroller Unit (MCU) with Power Saving Mode," 2011.
- [72] E. Burke and M. Barsotti, "High-Power Deployable Solar Panel Arrays for NanoSats and MicroSats " in *AIAA Responsive Space Conf.*, 2011.
- [73] K. G. S. Padmanabhan Pillai, "Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems," in *18th ACM Symp. on Operating Systems Principles*, 2001, pp. 21-24.
- [74] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *Proc. 36th Annual Symp. on Foundations of Computer Science*, 1995, pp. 374-382.
- [75] CubeSat Kit Pumpkin Inc., "CubeSat Kit Motherboard Datasheet, Hardware Revision: D, Single Board Computer Motherboard for Harsh Environments," ed, 2009.

Appendix 1 – Peripheral Error Rates Analysis

The following section shows the peripheral error rate (percentage) analysis as different system clock frequencies are applied in the C8051F120 microcontroller installed in the Pumpkin OBDH motherboard used in VELOX-I nano-satellite application.

Universal Asynchronous Receiver Transmitter 0 (UART0)

- Peripheral's Name: UART0
- Peripheral's Related Register(s): RCAP4
- Peripheral's Related Register(s)'s Limit(s): $0 \leq RCAP4 \leq 65535$
- Peripheral's Rate Symbol: baud0
- Peripheral's Rate Requirement: 115,200 baud
- Peripheral's Rate Formulation: $baud0 = sysclk / (prescaler * RCAP4)$
- Maximum Error Percentage: 5.0564%

baud0												
Sysclk Level	sysclk (MHz)	prescaler	Desired RCAP4			Actual RCAP4			Actual baud0			Error (%)
			115,200	57,600	28,800	115,200	57,600	28,800	115,200	57,600	28,800	
7	98.000	16	53.17	106.34	212.67	53	106	213	115,566	57,783	28,756	0.1532
6	79.625	16	43.20	86.40	172.80	43	86	172	115,734	57,867	28,934	0.4636
5	67.375	16	36.55	73.11	146.21	36	73	146	116,970	57,684	28,842	0.1460
4	49.000	16	26.58	53.17	106.34	27	53	106	113,426	57,783	28,892	0.3177
3	36.750	16	19.94	39.88	79.75	20	40	80	114,844	57,422	28,711	0.3092
2	24.500	16	13.29	26.58	53.17	13	27	53	117,788	56,713	28,892	0.3177
1	12.250	16	6.65	13.29	26.58	7	13	27	109,375	58,894	28,356	2.2469
												1.5400

Universal Asynchronous Receiver Transmitter 1 (UART1)

- Peripheral's Name: UART1
- Peripheral's Related Register(s): TH1
- Peripheral's Related Register(s)'s Limit(s): $0 \leq TH1 \leq 255$
- Peripheral's Rate Symbol: baud1
- Peripheral's Rate Requirement: 115,200 baud

Peripheral's Rate Formulation: $\text{baud1} = \text{sysclk}/(\text{prescaler}*\text{TH1}*2)$
Maximum Error Percentage: 2.8555%

baud1														
Sysclk Level	sysclk	pre-scaler	Desired TH1			Actual TH1			Actual baud1			Error (%)		
			115,200	57,600	28,800	115,200	57,600	28,800	115,200	57,600	28,800	115,200	57,600	28,800
7	98.000	48	8.86	17.72	35.45	9	18	35	113,426	56,713	29,167	1.5400	1.5400	1.2731
6	79.625	48	7.20	14.40	28.80	7	14	28	118,490	59,245	29,622	2.8555	2.8555	2.8555
5	67.375	48	6.09	12.18	24.37	6	12	24	116,970	58,485	29,243	1.5369	1.5369	1.5369
4	49.000	12	17.72	35.45	70.89	18	35	71	113,426	58,333	28,756	1.5400	1.2731	0.1532
3	36.750	12	13.29	26.58	53.17	13	27	53	117,788	56,713	28,892	2.2469	1.5400	0.3177
2	24.500	12	8.86	17.72	35.45	9	18	35	113,426	56,713	29,167	1.5400	1.5400	1.2731
1	12.250	4	13.29	26.58	53.17	13	27	53	117,788	56,713	28,892	2.2469	1.5400	0.3177

Serial Peripheral Interface 0 (SPI0)

Peripheral's Name: SPI0
Peripheral's Related Register(s): SPI0CKR
Peripheral's Related Register(s)'s Limit(s): $0 \leq \text{SPI0CKR} \leq 255$
Peripheral's Rate Symbol: spi0clk
Peripheral's Rate Requirement: 500,000 bps
Peripheral's Rate Formulation: $\text{spi0clk} = \text{sysclk}/(\text{SPI0CKR}*2)$
Maximum Error Percentage: 2.0833%

spi0clk														
Sysclk Level	sysclk	pre-scaler	Desired SPI0CKR			Actual SPI0CKR			Actual spi0clk			Error (%)		
			1,500,000	1,000,000	500,000	1,500,000	1,000,000	500,000	1,500,000	1,000,000	500,000	1,500,000	1,000,000	500,000
7	98.000	NA	32.67	49.00	98.00	33	49	98	1,484,848	1,000,000	500,000	1.0101	0.0000	0.0000
6	79.625	NA	26.54	39.81	79.63	26	39	79	1,531,250	1,020,833	503,956	2.0833	2.0833	0.7911
5	67.375	NA	22.46	33.69	67.38	22	33	67	1,531,250	1,020,833	502,799	2.0833	2.0833	0.5597
4	49.000	NA	16.33	24.50	49.00	16	25	49	1,531,250	980,000	500,000	2.0833	2.0000	0.0000
3	36.750	NA	12.25	18.38	36.75	12	18	37	1,531,250	1,020,833	496,622	2.0833	2.0833	0.6757

2	24.500	NA	8.17	12.25	24.50	8	12	25	1,531,250	1,020,833	490,000	2.0833	2.0833	2.0000
1	12.250	NA	4.08	6.13	12.25	4	6	12	1,531,250	1,020,833	510,417	2.0833	2.0833	2.0833

Serial Management Bus (SMBus)

Peripheral's Name: SMBUS

Peripheral's Related Register(s): SMB0CR

Peripheral's Related Register(s)'s Limit(s): 0 <= SMB0CR <= 255

Peripheral's Rate Symbol: smb0clk

Peripheral's Rate Requirement: 400,000 Hz

Peripheral's Rate Formulation: $\text{smb0clk} = \text{sysclk} / ((257 - \text{SMB0CR}) * 8)$

Maximum Error Percentage: 4.4034%

smb0clk																
Sysclk Level	sysclk	pre-scaler	Desired SMB0CR				Actual SMB0CR				Actual smb0clk			Error (%)		
			400,000	300,000	200,000		400,000	300,000	200,000		400,000	300,000	200,000	400,000	300,000	200,000
7	98.000	NA	226.38	216.17	195.75		226	216	196		395,161	298,780	200,820	1.2097	0.4065	0.4098
6	79.625	NA	232.12	223.82	207.23		232	223	207		398,125	292,739	199,063	0.4688	2.4203	0.4688
5	67.375	NA	235.95	228.93	214.89		235	228	214		382,813	290,409	195,858	4.2969	3.1968	2.0712
4	49.000	NA	241.69	236.58	226.38		242	237	226		408,333	306,250	197,581	2.0833	2.0833	1.2097
3	36.750	NA	245.52	241.69	234.03		246	242	234		417,614	306,250	199,728	4.4034	2.0833	0.1359
2	24.500	NA	249.34	246.79	241.69		249	247	242		382,813	306,250	204,167	4.2969	2.0833	2.0833
1	12.250	NA	253.17	251.90	249.34		253	252	249		382,813	306,250	191,406	4.2969	2.0833	4.2969

Analog to Digital Converter 0 (ADC0)

Peripheral's Name: ADC0

Peripheral's Related Register(s): ADC0CF

Peripheral's Related Register(s)'s Limit(s): 0 <= ADC0CF <= 255

Peripheral's Rate Symbol: adc0clk

Peripheral's Rate Requirement: 750,000 Hz

Peripheral's Rate Formulation: $\text{adc0clk} = \text{sysclk} / ((\text{ADC0CF} + 1) * 2)$

Maximum Error Percentage: 2.0833%

adc0clk														
Sysclk Level	sysclk	pre-scaler	Desired ADC0CF			Actual ADC0CF			Actual adc0clk			Error (%)		
			1,500,000	1,000,000	750,000	1,500,000	1,000,000	750,000	1,500,000	1,000,000	750,000	1,500,000	1,000,000	750,000
7	98.000	NA	31.67	48.00	64.33	32	48	64	1,484,848	1,000,000	753,846	1.0101	0.0000	0.5128
6	79.625	NA	25.54	38.81	52.08	25	38	52	1,531,250	1,020,833	751,179	2.0833	2.0833	0.1572
5	67.375	NA	21.46	32.69	43.92	21	32	43	1,531,250	1,020,833	765,625	2.0833	2.0833	2.0833
4	49.000	NA	15.33	23.50	31.67	15	24	32	1,531,250	980,000	742,424	2.0833	2.0000	1.0101
3	36.750	NA	11.25	17.38	23.50	11	17	24	1,531,250	1,020,833	735,000	2.0833	2.0833	2.0000
2	24.500	NA	7.17	11.25	15.33	7	11	15	1,531,250	1,020,833	765,625	2.0833	2.0833	2.0833
1	12.250	NA	3.08	5.13	7.17	3	5	7	1,531,250	1,020,833	765,625	2.0833	2.0833	2.0833

Timer 2 (TMR2)

Peripheral's Name: TMR2
Peripheral's Related Register(s): RCAP2
Peripheral's Related Register(s)'s Limit(s): 0 <= RCAP2 <= 65535
Peripheral's Rate Symbol: tmr2period
Peripheral's Rate Requirement: 1 interrupt(s) per millisecond
Peripheral's Rate Formulation: tmr2period = (65536 - RCAP2)*prescaler*1000/sysclk
Maximum Error Percentage: 0.0163%

tmr2period														
Sysclk Level	sysclk	pre-scaler	Desired RCAP2			Actual RCAP2			Actual tmr2period (ms)			Error (%)		
			2.0000	1.0000	0.5000	2.0000	1.0000	0.5000	2.0000	1.0000	0.5000	2.0000	1.0000	0.5000
7	98.000	12	49,202.67	57,369.33	61,452.67	49202	57369	61452	2.0001	1.0000	0.5001	0.0041	0.0041	0.0163
6	79.625	12	52,265.17	58,900.58	62,218.29	52265	58900	62218	2.0000	1.0001	0.5000	0.0013	0.0088	0.0088
5	67.375	12	54,306.83	59,921.42	62,728.71	54306	59921	62728	2.0001	1.0001	0.5001	0.0074	0.0074	0.0252
4	49.000	12	57,369.33	61,452.67	63,494.33	57369	61452	63494	2.0001	1.0002	0.5001	0.0041	0.0163	0.0163
3	36.750	12	59,411.00	62,473.50	64,004.75	59411	62473	64004	2.0000	1.0002	0.5002	0.0000	0.0163	0.0490
2	24.500	12	61,452.67	63,494.33	64,515.17	61452	63494	64515	2.0003	1.0002	0.5001	0.0163	0.0163	0.0163
1	12.250	12	63,494.33	64,515.17	65,025.58	63494	64515	65025	2.0003	1.0002	0.5006	0.0163	0.0163	0.1143

Appendix 2 – Time Estimator’s Parameters’ Model Error Extra Statistical and Graphical Results

Extra Statistical Result

Below is the table shows the model errors (in μs) of the time estimator parameters,

No of samples = 3,600 per system clock frequency level

Task Code & Symbol		Error (μs)										
		Level 1	Level 2	Level 3	Level 4	Level 5	Level 6	Level 7	Min	Mean	Max	Stdev
Second Upd	T _{f1}	4.282	4.520	16.139	10.861	2.563	3.400	4.677	2.563	6.635	16.139	4.984
	T _{f2}	1.947	2.825	2.165	2.736	1.528	2.148	2.103	1.528	2.207	2.825	0.448
PwrShk	T _{f3}	3.552	1.233	0.152	0.342	10.909	8.049	7.294	0.152	4.504	10.909	4.267
AdcsHk	T _{f4}	0.037	0.379	0.931	0.113	1.705	0.824	0.430	0.037	0.631	1.705	0.578
CommHk	T _{f5}	3.279	5.415	4.212	2.199	4.182	2.632	1.732	1.732	3.379	5.415	1.301
XdataGet	T _{f6}	6.913	0.256	1.420	2.223	15.753	11.857	10.591	0.256	7.002	15.753	5.952
WireUp	T _{f7}	1.025	1.070	1.623	1.935	0.670	1.601	1.332	0.670	1.322	1.935	0.432
RadioUp	T _{f8}	4.925	8.670	5.566	4.125	6.245	4.585	3.646	3.646	5.395	8.670	1.685
Cmd	T _{f9}	0.734	0.727	1.189	1.386	0.126	1.534	0.907	0.126	0.943	1.534	0.477
Sched	T _{f10}	1.209	2.763	0.819	0.607	1.471	0.354	0.650	0.354	1.125	2.763	0.816
Downlink	T _{f11}	4.519	0.278	0.777	1.374	7.859	10.162	8.503	0.278	4.782	10.162	4.088
Camera	T _{f12}	1.773	0.496	4.582	3.590	11.159	7.228	6.471	0.496	5.043	11.159	3.603
Xbee	T _{f13}	1.763	3.975	3.779	2.129	6.410	3.003	3.109	1.763	3.453	6.410	1.530
GPS	T _{f14}	2.345	5.440	4.987	2.883	9.257	4.181	3.454	2.345	4.649	9.257	2.311
Rtc	T _{f15}	0.668	1.775	0.055	0.055	1.269	0.147	0.309	0.055	0.611	1.775	0.673
PwrCqt	T _{f16}	0.248	0.183	0.353	0.618	0.082	0.249	0.739	0.082	0.353	0.739	0.239
Debug	T _{f17}	56.501	123.945	30.311	13.130	20.162	19.730	44.734	13.130	44.073	123.945	38.396
Cqt	T _{f18}	50.996	115.345	19.114	7.549	18.571	21.355	35.987	7.549	38.417	115.345	36.725
	Min	0.037	0.183	0.055	0.055	0.082	0.147	0.309	0.037	0.353	0.739	0.239
	Mean	8.151	15.516	5.454	3.214	6.662	5.724	7.593	2.042	7.474	19.247	6.028
	Max	56.501	123.945	30.311	13.130	20.162	21.355	44.734	13.130	44.073	123.945	38.396
	Stdev	16.716	37.983	8.157	3.677	6.441	6.384	12.381	3.335	12.505	36.873	11.610

Below is the table shows the model errors' percentages of the time estimator parameters,

No of samples = 3,600 per system clock frequency level

Task Code & Symbol		Error Percentage (%)										
		Level 1	Level 2	Level 3	Level 4	Level 5	Level 6	Level 7	Min	Mean	Max	Stdev
Second Upd	T _{f1}	0.570	1.135	5.934	4.495	1.414	2.024	3.131	0.570	2.672	5.934	1.956
Meta	T _{f2}	0.278	0.776	0.851	1.376	0.967	1.533	1.738	0.278	1.074	1.738	0.504
PwrHk	T _{f3}	0.759	0.528	0.098	0.295	14.833	12.694	14.366	0.098	6.225	14.833	7.272
AdcsHk	T _{f4}	0.021	0.424	1.548	0.255	5.573	3.109	1.974	0.021	1.843	5.573	1.978
CommHk	T _{f5}	0.730	2.336	2.714	1.905	5.347	3.922	3.151	0.730	2.872	5.347	1.480
XdataGet	T _{f6}	0.839	0.063	0.524	1.100	11.856	10.410	11.564	0.063	5.194	11.856	5.716
WireUp	T _{f7}	0.495	1.038	2.364	3.767	1.672	4.530	4.561	0.495	2.632	4.561	1.669
RadioUp	T _{f8}	0.718	2.449	2.361	2.333	5.234	4.510	4.410	0.718	3.145	5.234	1.606
Cmd	T _{f9}	0.426	0.846	2.080	3.238	0.385	5.218	3.801	0.385	2.285	5.218	1.870
Sched	T _{f10}	0.426	1.863	0.821	0.808	2.662	0.718	1.618	0.426	1.274	2.662	0.800
Downlink	T _{f11}	0.572	0.071	0.297	0.704	5.818	9.173	9.473	0.071	3.730	9.473	4.306
Camera	T _{f12}	0.355	0.200	2.691	2.808	14.085	10.438	11.624	0.200	6.029	14.085	5.821
Xbee	T _{f13}	0.559	2.446	3.451	2.615	12.507	6.558	8.510	0.559	5.235	12.507	4.184
GPS	T _{f14}	0.534	2.404	3.276	2.544	13.030	6.557	6.675	0.534	5.003	13.030	4.191
Rtc	T _{f15}	0.628	3.203	0.154	0.204	6.915	0.895	2.234	0.154	2.033	6.915	2.429
PwrCqt	T _{f16}	0.636	0.922	2.620	6.134	1.020	3.388	10.923	0.636	3.663	10.923	3.733
Debug	T _{f17}	0.423	1.798	0.661	0.383	0.800	0.919	2.572	0.383	1.080	2.572	0.810
Cqt	T _{f18}	0.731	3.144	0.792	0.417	1.402	1.893	3.941	0.417	1.760	3.941	1.330
	Min	0.021	0.063	0.098	0.204	0.385	0.718	1.618	0.021	1.074	1.738	0.504
	Mean	0.539	1.425	1.846	1.966	5.862	4.916	5.904	0.374	3.208	7.578	2.870
	Max	0.839	3.203	5.934	6.134	14.833	12.694	14.366	0.730	6.225	14.833	7.272
	Stdev	0.199	1.045	1.509	1.680	5.156	3.681	4.094	0.231	1.690	4.277	1.982

Graphical Results

Below are the graphical results showing the actual versus the modeled tasks' execution time in different system clock frequency levels (reflected in different System Periods, T_s), the model errors, and the model errors' percentages,

Foreground Tasks

The graphs below show the actual (blue line) versus the modeled (red line) tasks' execution time for VELOX-I nano-satellite's motherboard's simulated foreground tasks,

The graphs below show the modeled tasks' execution time errors of the VELOX-I nano-satellite's motherboard's simulated foreground tasks,

The graphs below show the modeled tasks' execution time error percentages of the VELOX-I nano-satellite's motherboard's simulated foreground tasks,

Event-Dependent Background Tasks

The graphs below show the actual (blue line) versus the modeled (red line) tasks' execution time for VELOX-I nano-satellite's event-dependent background tasks,

The graphs below show the modeled tasks' execution time errors of the VELOX-I nano-satellite's motherboard's event-dependent background tasks,

The graphs below show the modeled tasks' execution time error percentages of the VELOX-I nano-satellite's motherboard's event-dependent background tasks,

Time-Dependent Background Tasks

The graphs below (from left to right) show: (left) the actual (blue line) versus the modeled (red line) tasks' execution time for VELOX-I nano-satellite's motherboard's simulated foreground tasks, (middle) the modeled tasks' execution time errors of the VELOX-I nano-satellite's motherboard's time-dependent background tasks, and (right) the modeled tasks' execution time error percentages of the VELOX-I nano-satellite's motherboard's time-dependent background tasks,

Appendix 3 – Experiment’s Extra Statistical and Graphical Results

Extra Statistical Results

The table below shows other system time performance statistical results obtained

No of samples = 5,700 per orbit

Time Horizon (ms)													
Orbit No	Scenario Name	$\sum_{i=1}^{i=n_s} t_{totacti}$				$\sum_{i=1}^{i=n_s} t_{totacti} - \sum_{i=1}^{i=n_s} t_{si}$							
		Min	Mean	Max	Stdev	Min	Mean	Max	Stdev				
1	Power Safe-hold	52.27	62.97	72.33	5.20	-5.07	0.87	6.31	1.68				
2	Normal Operation	36.57	50.71	63.09	6.92	-15.41	-1.55	10.07	4.10				
3	Normal Operation + Ground pass	35.66	46.37	57.69	6.09	-15.25	-2.47	9.84	3.69				
4	Mission + Ground pass	36.17	45.36	54.94	5.54	-17.64	-1.65	12.07	4.07				
5	Multiple Missions + Ground pass	36.00	44.08	55.55	5.53	-15.55	-0.28	13.89	4.98				
Orbit No	Scenario Name	$\sum_{i=1}^{i=n_s} t_{toti}$				$n_s t_{WCET}$							
		Min	Mean	Max	Stdev	Min	Mean	Max	Stdev				
1	Power Safe-hold	72.36	77.28	84.62	3.41	82.00	82.00	82.00	0.00				
2	Normal Operation	69.62	74.73	80.04	2.76	82.00	82.00	82.00	0.00				
3	Normal Operation + Ground pass	68.75	74.02	76.99	2.16	82.00	82.00	82.00	0.00				
4	Mission + Ground pass	68.78	73.96	81.15	3.19	82.00	82.00	82.00	0.00				
5	Multiple Missions + Ground pass	66.80	72.93	81.15	3.69	82.00	82.00	82.00	0.00				

Graphical Results

The graphs below show the time performance parameters versus time, the proposed controller’s response, and the current drawn by the motherboard for the first 5 minutes of each of the simulated orbit scenario. The span of time displayed (that is, 5 minutes) is chosen such that the reader may get better visual impression of the simulated tasks’ dynamics, the proposed controller’s responses towards the dynamics, and optimization of the energy consumed due to the proposed controller’s response than if the full display of the entire orbit is shown.

Orbit 1

Proposed Controller Response

Motherboard's Average Current Drawn for Different Methods

Motherboard's Average Current Drawn for Different Methods

Orbit 3

System Time Performance

Proposed Controller Response

Motherboard's Average Current Drawn for Different Methods

Motherboard's Average Current Drawn for Different Methods

Orbit 5

System Time Performance

Proposed Controller Response

Motherboard's Average Current Drawn for Different Methods