Development and Evaluation of an Embedded System for a CubeSat Electrical Power Supply

by

Junqi Zhu

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Engineering

Department of Electrical and Computer Engineering
University of Alberta

Abstract

AlbertaSat is a group of researchers and students at the University of Alberta who are developing low-cost satellites with open-source software that follow the CubeSat form factor. It is one of the projects funded by the Canadian CubeSat Project, whose goal is to provide a hands-on satellite experience to students. The primary mission of the on-going CubeSat design of AlbertaSat, Ex-Alta2, is to predict, track, and assess the aftereffects of wildfires. The subject of the present thesis is the software design, implementation and evaluation of the electrical power supply for future AlbertaSat missions.

In this thesis dissertation, we describe the design and test of the real-time software system that controls a CubeSat electrical power supply (EPS). The development of this system takes the reliability, efficiency, and functionality of the EPS into account, at each fundamental design step. Major features of the new EPS include: (1) a software-optimized maximum power point tracking algorithm; (2) a battery protection module which has separate charging and discharging control; (3) a predictive algorithm for battery heater control that saves power by keeping the battery temperature lower in eclipse for discharge only, and heating the battery up before exiting eclipse in preparation for battery charging; (4) eighteen switched power channels, each with software-controlled overcurrent protection, to supply power to separate loads in the satellite; (5) auto-adjusted overcurrent thresholds that accommodate the increased semiconductor currents that occur as chips are exposed to increasing total lifetime doses of radiation; and (6) the backup design for configuration data to protect system from data corruption. The design and implementation of each module is discussed in detail.

Preface

This thesis describes research that was conducted as part of the AlbertaSat project at the University of Alberta, with PhD student Stefan Damkjar being the lead collaborator at the University of Alberta. The hardware system provided in Chapter 3 was mostly designed by Stefan Damkjar. The software system design in Chapter 4 is my original work. The system testing and data collection referred in Chapter 5 was done in collaboration with Stefan Damkjar.

Acknowledgements

I would like to acknowledge those who supported me throughout the completion of this work.

First, I would like to thank my supervisors Dr. Duncan Elliott and Dr. Bruce Cockburn for their guidance, assistance, patience, and contributions throughout this project process and my study at the University of Alberta. I would not have been able to complete this project without you.

Next, I would like to thank my fellow researcher, PhD student Stefan Damkjar, for his PCB design and his invaluable and selfless help all the time. It has been a pleasure to collaborate with you on this project and I wish you speedy progress as you complete your own thesis.

I would like to thank the Canadian Space Agency's Canadian CubeSat Project, NSERC, and the Faculty of Engineering for financial support. I thank CMC Microsystems, MathWorks and Texas Instruments for their support in the form of software tools and documentation. I also thank AlbertaSat members who helped me in various fields.

I also would like to thank all my colleagues in the VLSI lab, especially Mr. Behdad Goodarzy. Thanks for providing and maintaining the excellent and stressfree working atmosphere in the lab. Your encouragement and comfort helped me get through the hard times.

I would like to thank my friends, especially Dr. Bozheng Sun, Mr. Ningyuan Pei, Mr. Zhuowen Qian, Dr. Zhuobin Zheng, Mr. Zhuoheng Wu, and all of my roommates and landlords. Thanks for your company in the recent years and your help in every aspect of my daily life. I would never forget the time with all of you, which greatly enriched my spare time and brought me joy and precious memories.

Finally, and most importantly, I would like to thank my family. Thanks to my maternal grandparents, my paternal grandparents, my parents, and all my relatives who care about me. I really appreciate your support from both moral and materials. Your love and trust gave me the courage and perseverance to complete my study.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

Abbreviation	Definition
ADCS	Attitude Determination and Control System
BIST	Built-In Self-Test
C	C programming language
CAN	Controller Area Network
CCS	Code Composer Studio
CMOS	Complementary Metal–Oxide–Semiconductor
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSP	CubeSat Space Protocol
DAC	Digital-to-Analog Converter
DD	Displacement Damage
DMA	Direct Memory Access
ECC	Error Correction Code
EEPROM	Electrically-Erasable Programmable Read-only Memory
EPS	Electrical Power Supply
FB	FeedBack
GPIO	General-Purpose Input/Output
HD	Hamming Distance
INC	Incremental Conductance
I2C	Inter-Integrated Circuit
KCL	Kirchhoff's Current Law
LED	Light-Emitting Diode
LEO	Low Earth Orbit
LPO	Low-Power Oscillator
LPM	Low-Power Mmode
LSB	Least Significant Bit
MCU	Microcontroller Unit
MOSFET	Metal-Oxide-Semiconductor Field-Effect Ttransistor
MPP	Maximum Power Point

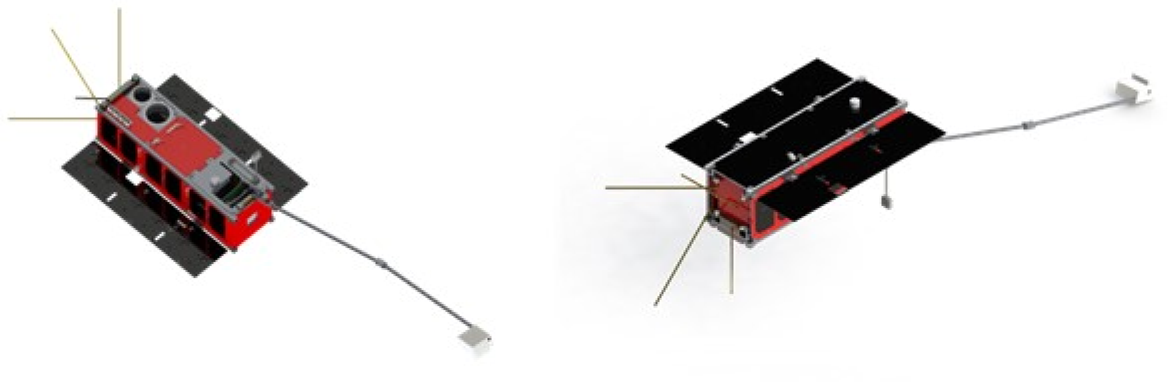
MPPT	Maximum Power Point Tracking
OBC	On-Board Computer
Op-amp	Operational amplifier
PLL	Phase-Locked Loop
PV	PhotoVoltaic
P&O	Perturb and Observe
RAM	Random-Access Memory
RTI	Real-Time Interrupt
RTOS	Real-Time Operating System
SECDED	Single-bit Error Correction Double-bit Error Detection
SEE	Single-Event Effect
SPI	Serial Peripheral Interface
TI	Texas Instruments
TID	Total Ionizing Dose
UART	Universal Asynchronous Receiver-Transmitter
UHF	Ultra-high Frequency
UVLO	UnderVoltage LockOut
WDI	Watchdog Interrupt
2S2P	Series-Parallel

Chapter 1: Introduction

1.1 Background and Motivation

For the past 65 years since the launch of Sputnik in 1957, satellite technology has been a high-cost and high-risk endeavor that has been affordable only for governments or large companies. Moreover, because of the high budgets and long development timeframe, space projects tend to rely on old, proven technologies with a successful flight history instead of relying on state-of-the-art technologies and devices.

In 1999, Jordi Puig-Suari at California Polytechnic State University, San Luis Obispo and Bob Twiggs at Stanford University started the CubeSat Project [1], which firmly established the nanosatellite category. This ongoing project provided an affordable standard for the design of small satellites to reduce their cost and development time. CubeSat-compatible design very quickly became a popular topic for university student-based projects.



(a) Ex-Alta 2 Front Isometric View (b) Ex-Alta 2 Back Isometric View

Figure 1.1: 3D Rendering Views of Ex-Alta 2. Image Credit: AlbertaSat [2]

This thesis investigates future electrical power supply designs that are intended for future missions in the AlbertaSat student satellite project at the University of Alberta. This project is in turn a part of the Canadian CubeSat Project funded by the Canadian Space Agency (CSA) [3]. The main mission of the Ex-Alta 2 CubeSat is to demonstrate the potential capabilities possessed by a relatively inexpensive cube satellite platform for predicting, tracking, and assessing the aftereffects of wildfires. Two other CubeSats, part of the Northern SPIRIT collaborative project among the University of Alberta, Yukon University and Aurora College are being designed along with Ex-Alta 2 and share many subsystems with Ex-Alta 2. More generally, the AlbertaSat project provides an educational experience which supports student interest in space and provides an opportunity for students from several disciplines to work together towards a common goal of designing and building a real satellite that will be launched into space [3].

The previous and on-going satellites of AlbertaSat, Ex-Alta 1 and Ex-Alta 2, used a commercial electrical power supply (EPS) board. The main objective of this thesis research is to investigate and design a flexible open-source EPS module that will satisfy the requirements for future AlbertaSat missions.

1.2 Objectives

The objectives of this project are to investigate and design a CubeSat-compatible EPS that will be an attractive alternative to the commercial modules currently used in the AlbertaSat project.

1.2.1 Main Requirements of the Custom EPS

- Four solar panel input channels with independent maximum power point tracking (MPPT) converters
- Three regulated output power buses at 1.2 V, 3.3 V, 5 V, and an unregulated output voltage equal to the battery voltage
- Eighteen configurable output channels with 4-A maximum current and a software-programmable overcurrent protection module
- Fast overcurrent protection mechanism, battery voltage and temperature safety features
- Current, voltage and temperature housekeeping data logging
- Communication interfaces (Serial and CAN) with the satellite's main on-board control computer

The main novel achievements in this thesis research includes (1) a software-optimized maximum power point tracking algorithm using a boost converter and some control circuits that we developed; (2) a battery protection module which has separate charging and discharging control; (3) multiple switched power channels, each with software-controlled overcurrent protection, to supply various loads in the satellite; (4) auto-adjusted overcurrent threshold to accommodate increased semiconductor currents as chips are exposed to increasing total lifetime doses of radiation; (5) a predictive algorithm for battery heater control that saves power by keeping the battery temperature lower in eclipse for discharge only, and heating the battery up before exiting eclipse in preparation for battery charging; and (6) a backup mechanism for configuration data that protects the EPS software system from data corruption. The general objective is therefore to design a system that takes the reliability, efficiency, and functionality into account, at each fundamental design step.

1.3 Outline of the thesis

The next chapter reviews the background theory related to space applications and the relevant literature on power system design. This section is followed by a brief review of the commercial EPS modules that were used in the two previous CubeSats in AlbertaSat. These modules formed the starting point for new EPS design as well as references for performance comparisons.

In Chapter 3, the general specifications of the new EPS and the high-level architecture of the module are introduced. The design of each subsystem is then presented in detail, including the selection of commercial components.

The design and implementation of the software system is then described beginning with a high-level software architecture that supports the flow of operation tasks. Essential software features are also introduced. The comparison and selection of various alternative algorithms and designs for some modules will be discussed in this chapter.

Design verification tests for the hardware and software subsystems are presented in Chapter 5.

The final chapter summarizes the accomplished work and provides concluding remarks. Some possible future design improvements to the EPS system are proposed as well.

Chapter 2: Literature Review

There has been much nanosatellite design activity over the last decade [4]. Hundreds of nanosatellites have been designed and launched into orbit. However, compared to other subsystems, research on EPS design for CubeSats and nanosatellites has been relatively limited. Perhaps the reason is that the design of advanced communication systems and other payloads is more attractive than that of the seemingly ordinary power system. In actual fact, according to research on the major causes of failures of CubeSats, EPS failures have been reported to be the main cause of mission failure, with about 44% of all failures occurring in the first 30 days and about 36% in the first 90 days after launch [5]. Due to the EPS module's crucial role in a satellite's success, developers might understandably tend to adopt a proven commercial system to minimize potential risks.

Considering the critical role played by the EPS in CubeSats and the objective to design a reliable and flexible module to replace the existing commercial EPS design, this chapter will focus on fundamental EPS requirements and system constraints.

2.1 The Space Environment

The harsh operating environment in space is the most significant difference between an electronic system designed for on-earth use and one designed for use in space. This section covers the main potential hazards that could damage the components in Low Earth Orbit, mainly focusing on ionizing radiation, thermal variation, and vacuum effects. In addition, the additional challenges of efficiently operating solar panels and the requirement to keep track of orbital position is also considered.

2.1.1 Radiation

With respect to microelectronics reliability, three main space radiation effects must be taken into account: total ionizing dose, displacement damage and single-event effects [6].

Damage caused by total dose effects is called *total ionizing dose* (TID). When energized particles pass through the materials in electronic devices, such as semiconductors or insulators, the ionization process caused by the impact of the particles will transfer kinetic energy into the material, and such impacts can create electron-hole pairs in the semiconductor lattice [6]. The charges induced by radiation can accumulate on the gate of the metal–oxide–semiconductor field-effect transistor (MOSFET) and change the value of threshold voltage, which can lead the transistor to be on for an extended time if the change is large enough. *Displacement damage* (DD) is the result of energy loss, which happens when energetic particles impact into a solid material. The influence of DD depends on many factors such as particle type, measurement temperature, and the duration and intensity of the irradiation. DD can cause degradation of materials and changes in device properties such as minority carrier lifetime, which leads to degradation of gain and increased leakage current in transistors.

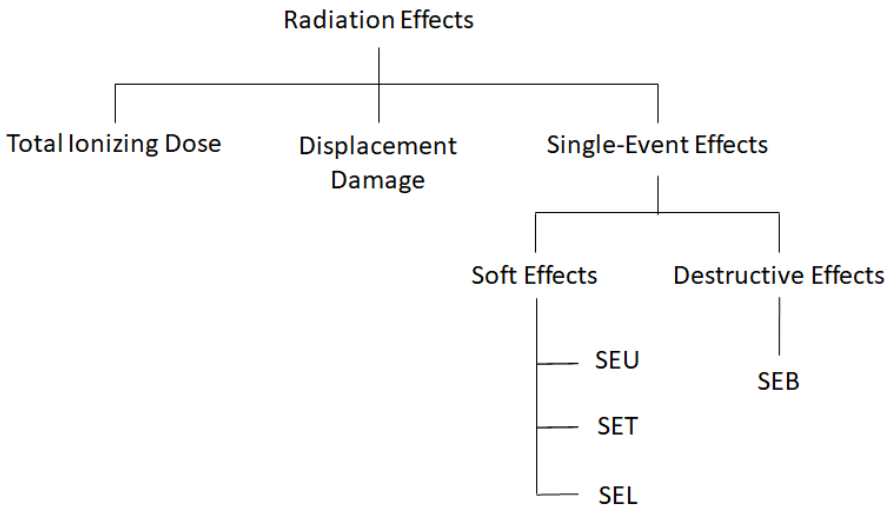


Figure 2.1: Classification of Radiation Effects [6],[7]

The previous two effects are both cumulative [6], and the most common method to avoid them is to use shielding. Shielding in satellites is often provided by placing electronic systems inside an aluminum enclosure [7]. This bulkier and more expensive approach will not be discussed in this thesis. Compared to material degradation over time, *single-event effects* (SEE) occur more randomly and instantly. SEEs are caused by single particle strikes on a component; therefore, they may happen any time after the satellite is launched into orbit.

SEEs are normally classified into two types: destructive effects that are non-reversible and soft effects that are temporary, which can be recovered by system reset or some other operations [6]. However, the demarcation between the two is not fixed. If a soft error effect is not located and corrected fast enough, it may still disrupt the system operation. On the other hand, some destructive effects may not do permanent damage if detected and removed quickly. Some major kinds of the SEEs are listed below [7],[8]:

1. *Single Event Upset* (SEU): SEU is a change of state of a digital circuit caused by ions or electromagnetic radiation striking a sensitive area (e.g., a storage node in a flip-flop or memory) in a micro-electronic device[6]. As a result, logic signals can be flipped in value, and this can lead to a reset or re-writing in the device as well as affecting the operation of the surrounding interface circuitry.
2. *Single Event Transient* (SET): The free charge generated by ionizing particles may gather in a sensitive area of the circuit and cause a current or voltage transient, which may lead to flipped logic bits or the incorrect sensing of logic signals.
3. *Single Event Latch-up* (SEL): SEL occurs when a radiation particle induced a self-sustaining current (typically a short circuit between power and ground) in the parasitic SCR (silicon-controlled rectifier) found in bulk CMOS circuits. Damage can be avoided (a soft error) if the overcurrent condition can be detected and power removed promptly, otherwise SEL can result in single event burnout, as described below.
4. *Single Event Burnout* (SEB): SEB can occur in power MOSFETs, BJTs and CMOS when a strike induces the activation of transistors, including due to SEL, above. It causes an overly high current that can permanently damage electronic components.

For the space system, SEEs can be observed at any time during a mission. TID radiation effects are cumulative over the spacecraft's lifetime and are less of a concern for short-lived satellites in low earth orbit.

2.1.2 Low Earth Orbit Considerations

The orbit that the satellite follows will produce a periodically varying radiation environment as well as varying sunlight intensity as the satellite repeatedly passes into eclipse and then exits from the Earth's shadows (without considering satellite tumbling). For the application of Ex-Alta 2, this section is focused on Low Earth Orbit (LEO).

An LEO is an orbit that is at an altitude between 180 and 650 km. The orbital period for LEO is also defined to be less than 128 minutes [9]. Most of the satellites in space are in this orbit.

The equation for the orbital period P of a satellite can be derived from Newton's formulation of Kepler's Third Law [10] as follows

$$P \approx 2\pi\sqrt{\frac{a^3}{\mu}} \tag{2.1}$$

where a is the semi-major axis, which is the sum of the satellite's altitude and the Earth's radius, and μ is the standard gravitational parameter of a celestial body, which equals 398600 km^{3}s^{-2} [11]. The expected altitude of Ex-Alta 1 is 400 km, and the Earth's radius is 6371 km. According to Equation 2.1, the approximate orbital period of our CubeSat is 92 minutes.

The equation for the eclipse period, which represents the time that a satellite is in the shadow of the Earth, of the entire period is given by [12]:

$$f_e = \frac{1}{\pi}\left[\frac{\sqrt{A^2+2R_EA}}{(R_E+A)coscos\,\beta}\right] \tag{2.2}$$

where A is the orbit altitude, R_E is the Earth's radius and β is the sum of orbit inclination and angle between the sunline and ecliptic plane. According to result of calculation shown in Fig 2.2, we note that the eclipse period is in the range of 0% to 40 % of the entire period, with an altitude of 400 km. The inclination of Ex-Alta 2 is very similar to that of the International Space Station (ISS), which is about 51.6 °. We assume that the future CubeSat will be projected from the ISS while already in orbit. Therefore, the expected eclipse duration should be about 33% of the orbital period.

2.1.3 Temperature

The main heat sources in a satellite include direct solar flux and heat generated by the operation of electronic components. However, compared to the thermal environment on Earth, the vacuum and low-pressure environment in space is very harsh and this produces two main challenges in heat dissipation management. First, there is the accumulation and required dissipation of heat: the common heat transfer methods in space are internal conduction and radiation from the outside surface of the satellite, which is a relatively limited heat dissipation method. Another consideration is that the satellite must survive fairly large and extreme variations in temperature as it orbits the Earth. When the satellite is in eclipse, it is in a very cold environment; when exposed to unshielded sunlight, the temperature rapidly increases. The temperature range of satellite in LEO is about -65 °C to +125 °C with thermal cycling dependent on the orbit altitude [13].

For the solar panels, the temperature will greatly influence the efficiency of energy harvesting, as will be discussed in detail in the following section. For the battery, it is likely to suffer from capacity loss when the temperature is low. Most batteries in satellites require active heating to keep the temperature within a certain range, which is dependent on the chemistry composition [11]. On the other hand, high temperature also decreases the reliability of electronic components. When over 50 °C, the failure rate of semiconductor devices doubles for each 8 °C rise [11]. Moreover, the rapid variation in temperature also degrades the lifetime of components and may lead to failure such as degradation of chip solder joints, which can be subject to cracking or loss of electrical contact.

2.2 Nanosatellite Technology

2.2.1 Overview

The term CubeSat, refers to a class of nanosatellite, typically weighing between 1 to 10 kilograms [14]. Its name is defined by the ten-centimeter cube unit that defines the basic unit of volume of a CubeSat. One cube in this size is referred to as "one unit", and a satellite of this size is called a "1U CubeSat". Multiple cubes can be combined together, going only along one axis, to form larger CubeSats, such as 2U, 3U and 6U. As part of the CubeSat Community, all designers and developers have an obligation to ensure safe operation of their systems and to meet the design and minimum testing requirements outlined in the CubeSat design specification [14].

A major motivation for initial CubeSat development was for education. CubeSat technology allowed small satellites to operate within a low budget in development, launch, and maintenance. Consistent with cost and risk reduction, CubeSat projects tend to use commercial off-the-shelf (COTS) components. These components are cheaper than radiation-hardened counterparts, and they reduce the number of tests since their quality is already proven. In order to guarantee the reliability of a COTS components, a CubeSat is normally launched into LEO within a limit of 100 – 400 km altitude. At these low altitude orbits, the satellite is still well within the Earth's magnetosphere, which can partly protect it from ionizing radiation from the sun and outer space sources [12].

Since the CubeSat project has a much higher risk tolerance than the traditional satellite development methods, it is a competitive solution for many experimental space applications. In addition, it also changes the way of launching satellites. In 2016, Polar Satellite Launch Vehicle (PSLV) made an innovation that launched 101 CubeSats along with 3 other satellites using a single rocket [4].

2.2.2 The Electrical Power Supply

The electrical power supply (EPS) is designed to provide the regulated power source for the whole satellite using energy produced by solar panels. Meanwhile, as mentioned in the previous section, the EPS is also a major likely cause of CubeSat failure. Therefore, the EPS is one of the most critical subsystems of a spacecraft. In the typical EPS design process, EPS functionality can be divided into the following requirements [15]:

1. Reliably supply electrical power for the duration of the CubeSat mission
2. Control the distribution of power to the CubeSat
3. Provide necessary power regulation
4. Satisfy power requirements when in eclipse and for both average and peak loads
5. Collect and provide on-demand housekeeping data

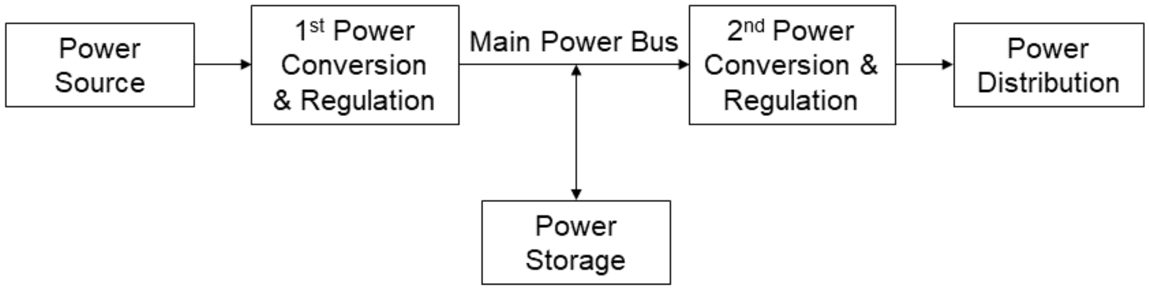

Power Source
1st Power Conversion & Regulation
Main Power Bus
2nd Power Conversion & Regulation
Power Distribution
Power Storage

Figure 2.2: Typical EPS Subsystems

Typical EPS subsystems are shown in Figure 2.2. The power source is an array of solar panels, which will be discussed in the next section. The solar panels convert solar energy into electrical energy in the form of direct current at a relatively low voltage. This unregulated source of electrical power must be converted by a direct energy conversion and regulation module, which is sometimes also controlled by a maximum power point tracking (MPPT) algorithm. The converted power is then transmitted to main power bus or stored in some form, such as in a battery, so that the stored energy can be used and provide electrical power at regulated voltages when the satellite is in eclipse or during peak loads. The next step is to convert and regulate the power for a second time before its distribution to other satellite subsystems. The second conversion and regulation module should satisfy the voltage and current requirements of those loads. After that, power can be distributed to different power channels to subsystems within the satellite.

The consideration of mission requirements is essential when designing the power system for any satellite. For example, one must estimate the average and peak power load requirements in order to properly size the power system (such as determining the efficiency and the total number of solar cells and the properties of the primary energy storage system). The orbital parameters are required to predict the incident solar energy, radiation environments, and eclipse periods. Estimating the mission lifetime can be used for component selection and redundancy design. Furthermore, the satellite specification also determines the characteristics of power source (such as fixed-body or deployable) [15].

2.3 The Photovoltaic Cell

2.3.1 Overview

The photovoltaic (PV) cell, also called a solar cell, is the most popular choice in nanosatellites for the power source. It has many characteristics that make it ideal to be used in space: excellent reliability, high power-to-weight ratio, and relative low cost. This section will focus on the features needed to be considered during the design of EPS.

A PV cell is a semiconductor device that contains a P-N junction. The structure is made from light absorbing materials, such as crystalline silicon, cadmium telluride (CdTe) and gallium arsenide (GaAs), which absorb photons and generate free electrons through the photovoltaic effect [16]. When the PV cell is exposed to sunlight, photons strike on its surface so that some electrons are energized so that they become free to move through the semiconductor lattice in the conduction band. A built-in-potential barrier at the PN junction acts on these electrons in order to produce a voltage, by which the current is driven [16]. The stronger the incident irradiance, the more power the solar cell will generate. The specifications of solar cell are usually calculated assuming that the solar constant is 1361 W/m^2 [17].

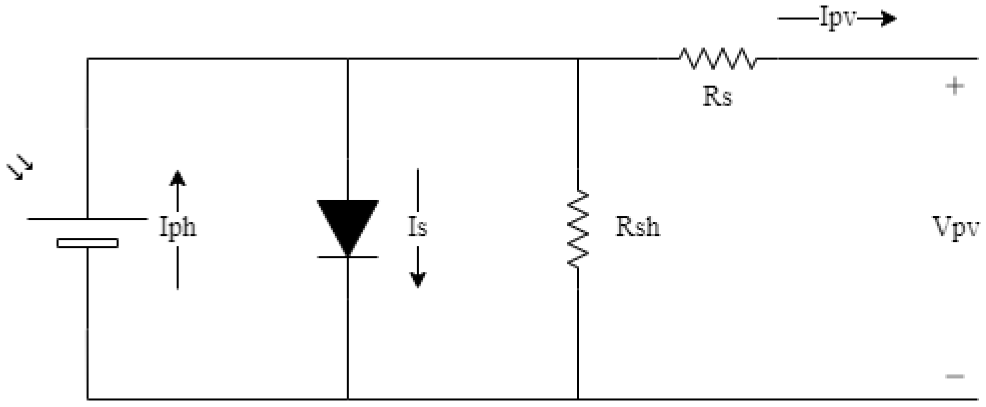


Figure 2.3: Equivalent Circuit for the PV Cell [18]

2.3.2 Mathematical Model

A mathematical model is necessary to evaluate the voltage, current and power behavior of a PV cell in different circumstances. Figure 2.3 represents the equivalent circuit of a simplified solar cell, which consists of simple discrete electrical components.

In the figure, V_{pv} denotes the voltage of the cell, while I_{pv} denotes the current. The power source, with a photocurrent I_{ph}, is connected in parallel with the exponential diode in order to simulate an ideal solar cell. In practice, there will be electrical power loss during the charge transfer, and we will use shunt resistor R_{sh} and series resistor R_s to model these kinds of resistive loss. The output current of the cell, I_{pv}, can be modelled as follows [18]:

$$I_{pv} = I_{ph} - I_s\left(e^{\frac{q(V_{pv}+I_{pv}*R_S)}{nKT}} - 1\right) - \frac{(V_{pv}+I_{pv}*R_S)}{R_{sh}} \tag{2.3}$$

where:

- I_{ph} = solar induced current in Amperes
- I_s = diode saturation current in Amperes
- q = electron charge in Coulombs (1.6e^{-19} C)
- K = Boltzmann constant (1.38e^{-23} J/K)
- n = ideality factor for a diode (1)
- T = temperature in Kelvins

When the cell is operated in an open circuit configuration, the output current equals zero and the output voltage is called the *open-circuit voltage* V_{oc}. Assuming that the shunt voltage is high enough to neglect the final term of the Equation 2.3, we can calculate the V_{oc} as [19]:

$$V_{oc} \approx \frac{KT}{q}\ln\left(\frac{I_{ph}}{I_s} + 1\right) \tag{2.4}$$

On the other hand, when the cell is shorted, the output voltage is forced to zero and the resulting output current is defined to be the *short-circuit current* I_{sc}. Similarly, we assume it is an ideal cell, which means it has low R_s and I_s and high R_{sh}. Therefore, I_{sc} can be formulated by [19]:

$$I_{sc} = I_{ph} \tag{2.5}$$

The *short-circuit current* I_{sc} can be equivalent to the solar-induced current, depending on the irradiance level. Meanwhile, the *open-circuit voltage* V_{oc} is less affected by the variations in the irradiance level but mostly depending on the logarithm of I_{ph} and I_s ratio according to Equation 2.4.

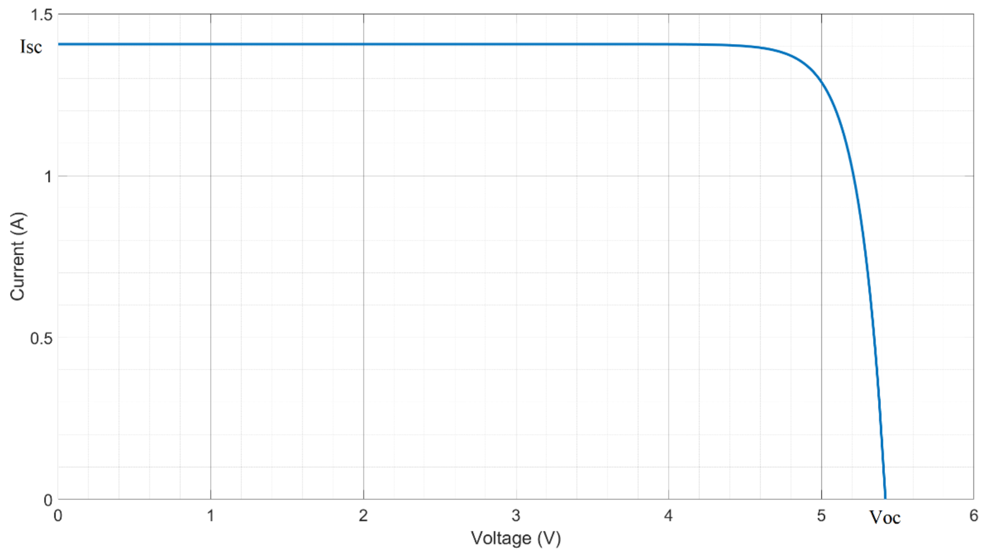


Figure 2.4: Simulated Characteristic I-V Curve of a Solar Panel. Based on the specifications of PV XTJ Solar Cell [20] with area of 79.86 cm^2 and AM0 condition, at a temperature of 15°C.

With Equations 2.3, 2.4 and 2.5, we can plot the characteristic current-voltage curve of a given solar panel, as shown in Figure 2.5. It show the possible combination of current and voltage output of the PV XTJ solar cell at the certain condition.

2.3.3 Maximum Power Point

According to the I-V curve from Figure 2.4, we note that the power of the solar cell varies with changes in the load voltage and the corresponding current. Because of the non-linear relationship, there is a specific operating point on this curve where the solar cell will produce the maximum output power.

Figure 2.5 shows the power-voltage curve over the I-V curve. The point with maximum power Pmax is called the maximum power point (MPP).

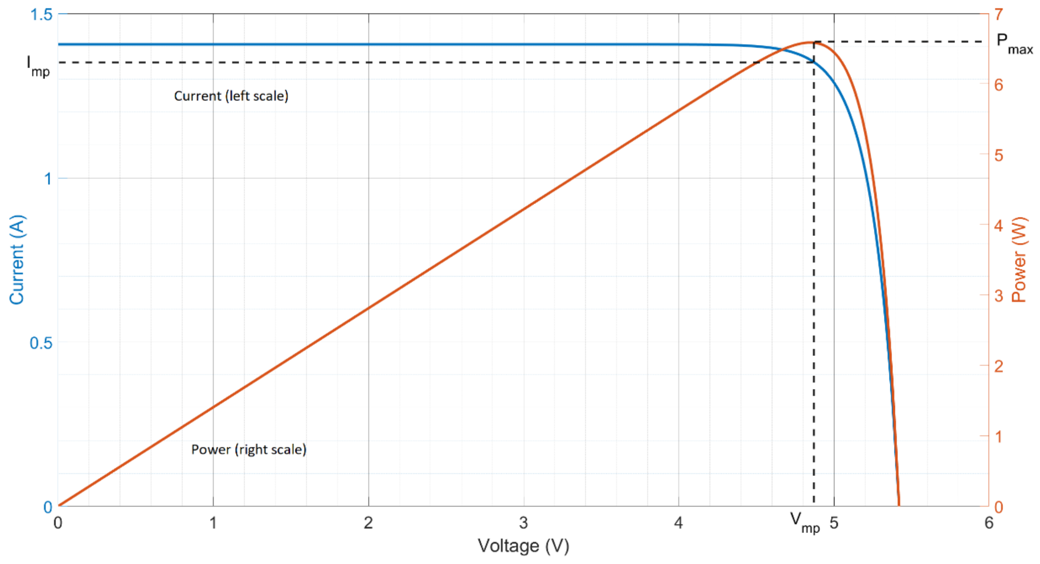


Figure 2.5: Simulated P-V & I-V Curve of a Solar Panel. Based on the specifications of PV XTJ Solar Cell [20] with area of 79.86 cm^2 and AM0 condition, in a temperature of 15 °C.

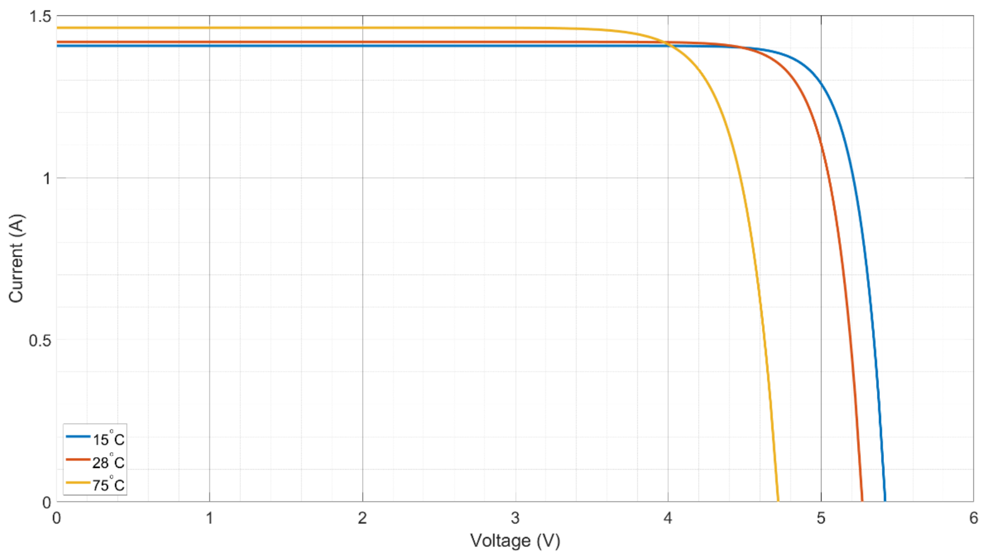
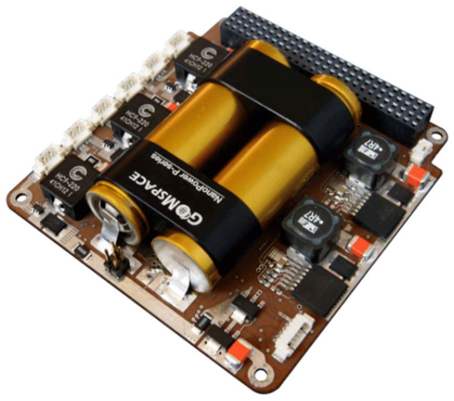


Figure 2.6: Simulated Characteristic I-V Curves of a Solar Panel with Respect to Temperature. Based on the specifications of the PV XTJ Solar Cell [20] with area of 79.86 cm^2 and AM0 condition.

2.4 Existing Commercial Systems


(a) GomSpace NanoPower P31u [21]

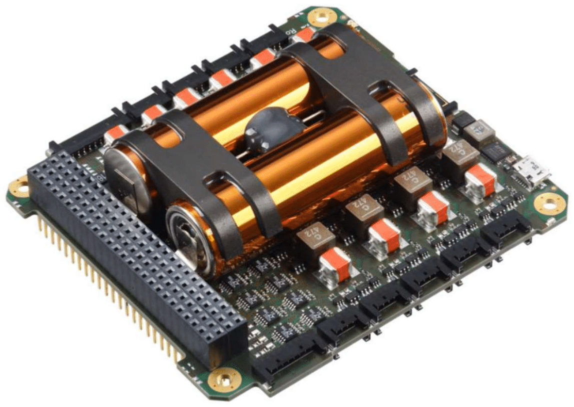

(b) NanoAvionics EPS-G0-R3 [22]

Figure 2.7: CubeSat EPS Modules, Top View

In this section, commercial EPS modules from two major manufacturers, GomSpace and NanoAvionics, are discussed. These modules were used on Ex-Alta 1 and Ex-Alta 2, respectively.

Both systems have their advantages and disadvantages, but neither of them can fully satisfy all the requirements for future CubeSat projects. Moreover, budget is also a factor of consideration. These commercial systems don't have a fixed market price, but both are higher than 5000 Canadian dollars. The high cost is another motivation to design our own EPS board.

2.4.1 GomSpace [21]

GomSpace is one of the leading manufacturers and suppliers of CubeSat. Its EPS module for small nanosatellite, P31u, was updated to V30 as of January 2022. In this section, we only discuss the older version (V9) used on Ex-Alta 1.

NanoPower P31u EPS is designed for those small and low-cost satellites, which have power demands of around 1-30 W. The hardware architecture is similar to the typical design shown in Figure 2.2. The three high-efficiency boost converters are used to condition the output power from three solar cells and to supply it to main power bus. The incoming power along with the battery power are then fed into two buck converters to supply regulated 3.3-V and 5-V output, respectively. At the terminal, six output channels can be configured to either 3.3 V or 5 V individually.

The power converter block has two methods of power-point tracking: one is a software-defined constant voltage that works at a default operating point. Another is the MPPT algorithm that is controlled using software executing on a microcontroller. The system has four running modes: critical, safe, normal and full. The software will switch between these four modes depending on the battery output voltage to ensure safe operation and long battery life. An on-board kill-switch is used to power-cycle reset the system. When the current monitors detect a battery overcurrent condition, the kill-switch will be switched off for 100 ms and then switch back on again. The output channels have latch-up protection, with a user-configurable overcurrent threshold.

However, the big weakness of this EPS is that some of the system configuration, such as overcurrent threshold, are hardware-configured. In this case, those configurations are fixed, and users have no chance to modify them after the satellite is launched.

2.4.2 NanoAvionics [22]

NanoAvionics is a younger company (founded in 2014) that is focused on satellite mission integration. Specifically, it is focused on delivering new-generation satellite buses and propulsion systems for the satellite applications market.

NanoAvionics EPS has similar design architecture as GomSpace EPS, but it has a wider range of input and output power EPS designs. The basic design has four MPPT converters which supports up to eight solar panel input channels. For the power regulation side, besides the basic 3.3-V and 5-V supply voltages, it also supplies another two configurable rails ranging from 3-18 V. Furthermore, it has 10 independent output channels that allows more subsystems to be driven by the EPS. And with the additional EPS output channel expander, it supports up to 18 output channels.

Compared to the GomSpace EPS, the NanoAvionics has both hardware and software-configured overcurrent protection. The software-programmable threshold can be modified by command after satellite launch. It also has a software-controlled MPPT mode that enables users to set the power point manually. This mode can prevent damage from battery charging overcurrent condition.

As a newly designed EPS, it is more advanced than the GomSpace unit, but it still has limitations. All of the software-programmable variables must be configured manually, which means they are still fixed values when there is no one maintaining the system. When some errors occur, if users don’t manually modify the system configuration in time, destructive effect can still occur.

Chapter 3: Hardware System Design

The previous chapters reviewed background knowledge and previous work. That material highlighted the main design requirements for the new EPS system, including the output voltage and current ranges, reliability considerations, compatibility constraints, and the need for flexibility and programmability. This chapter brings those factors together and proposes a high-level block design, along with design of subsystem and selection of components.

Note that the hardware design was mostly done by my colleague, PhD student Stefan Damkjar. The author only participated in part of the design and test of the power conversion module.

3.1 High-Level Design

The high-level block diagram of the required EPS is shown in Figure 3.1. Note the similarity to the typical EPS structure introduced in Chapter 2.

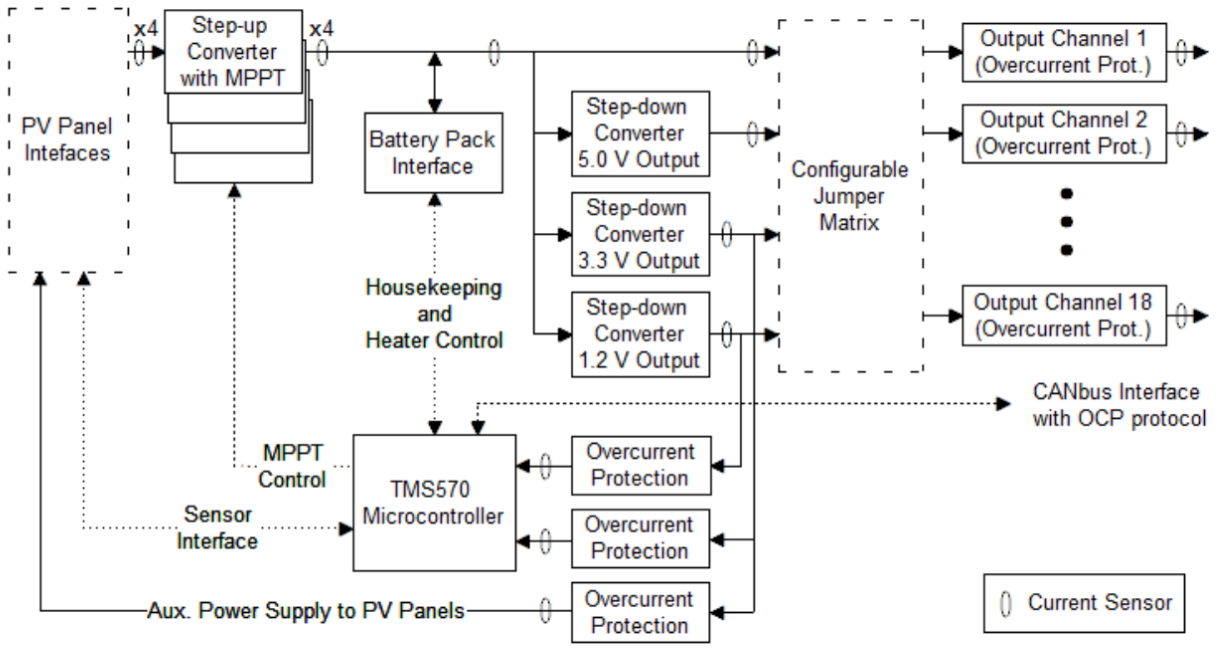

PV Panel Intefaces
x4
Step-up Converter with MPPT
x4
Battery Pack Interface
Step-down Converter 5.0 V Output
Step-down Converter 3.3 V Output
Step-down Converter 1.2 V Output
Configurable Jumper Matrix
Output Channel 1 (Overcurrent Prot.)
Output Channel 2 (Overcurrent Prot.)
Output Channel 18 (Overcurrent Prot.)
Housekeeping and Heater Control
CANbus Interface with OCP protocol
MPPT Control
Sensor Interface
TMS570 Microcontroller
Overcurrent Protection
Overcurrent Protection
Overcurrent Protection
Aux. Power Supply to PV Panels
Current Sensor

Figure 3.1: System Block Diagram provided by Damkjar [23]

There are eight solar panels on Ex-Alta 2 and the electrical energy gathered by the solar panels is raised in voltage and regulated by four boost converters. Each boost converter implements the MPPT algorithm in addition to implementing voltage step-up. The power conversion circuit will be covered in more details in the following section. The main power bus is unregulated, and its voltage follows the raw battery voltage.

The power distribution module provides power to eighteen output channels supporting satellite subsystems typically including an On-board Computer (OBC), an Ultra-high Frequency (UHF) communication board, an Attitude Determination and Control System (ADCS) board and a payload. The payloads on Ex-Alta 2 were a magnetometer and a multispectral imager. Each output channel can be configured to provide a regulated 1.2-V, 3.3-V, 5-V supply voltage or a connection to the unregulated main bus voltage. Each channel has hardware-controlled overcurrent protection and software-controlled overvoltage and undervoltage protection.

The energy storage is implemented on a separate battery board, named Prometheus in Ex-Alta 2. There are four lithium-ion cells on the battery board which are wired in the *series-parallel* (2S2P) configuration, which will be introduced in a later section. The battery pack is required to be monitored and protected by software. Specially, besides voltage and current, battery temperature is also a constraint. Going beyond its safe temperature range will decrease a battery's efficiency and lifetime and increase the potential of battery damage. The temperature is monitored by a microcontroller which controls a battery heater that keeps the battery board within a safe temperature range.

The digital control system is based on a TI TMS5701224PGE microcontroller. This device is a 32-bit ARM Cortex-R4F MCU with maximum 180-MHz clock frequency. The program image and configuration data are stored in non-volatile flash memory. The EPS current and temperature sensors are controlled through an I2C interface [24],[25], while digital-to-analog converters (DAC) are controlled through an SPI interface. The MCU receives commands from the OBC through a CAN interface and it also supports a serial UART protocol for debugging. The microcontroller is powered with a regulated 3.3-V power supply. The detailed digital control circuitry will be covered in the following section.

3.2 Power Conversion

Figure 3.2 shows the high-level architecture of the power conversion stage. It consists of a fully integrated boost converter, a DAC and controlling circuit subsystem. The main idea is to control the boost converter using the analog DAC that converts the digital signal from the MCU. The output of this module is then connected to the main power bus.

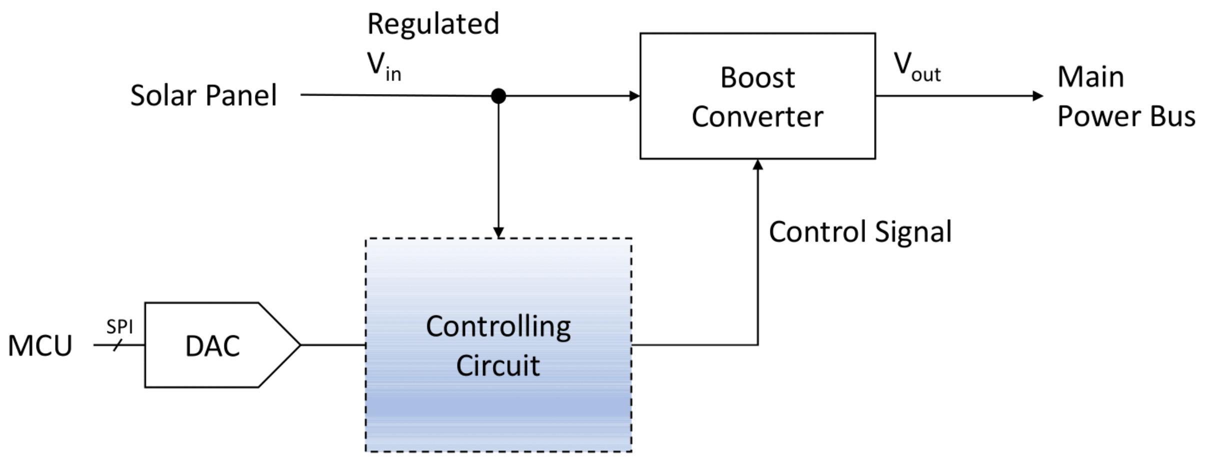


Figure 3.2: Overview of the Power Conversion Stage

3.2.1 Boost Converter

For this EPS design, boost converters are used in the power conversion stage to condition the power from the PV cells. Considering many factors, including input and output voltage range, operating temperature, and physical parameters such as mounting type, two commercial boost converters were chosen to be candidates.

Converter Type	Efficiency (%)	Input Voltage (V)	Output Voltage (V)	Max. Switch Current (A)	Operating Temp. (°C)
TPS61088	Up to 91	2.7 – 12	4.5 – 12.6	10	-40 – 150
MP3432	> 95	2.7 – 13	Up to 16	10	-40 – 125

Table 3.1: Parameters Comparison Between Two Commercial Boost Converters [26],[27]

Key specifications of the two selected boost converters, TPS61088 and MP3432, are listed in Table 3.1. Both of them use an adaptive constant-off-time (COT) control method to regulate the output voltage. Both of them have an integrated rectifier switch that blocks the current coming back from the opposite direction, so an external rectifier diode is not needed. They have very similar functions, while MP3432 has larger range on most of the parameters. However, TPS61088 was selected for the EPS because of its smaller start-up delay. Since the boost converter is switching at a high frequency, this factor greatly influences the efficiency of the boost converter.

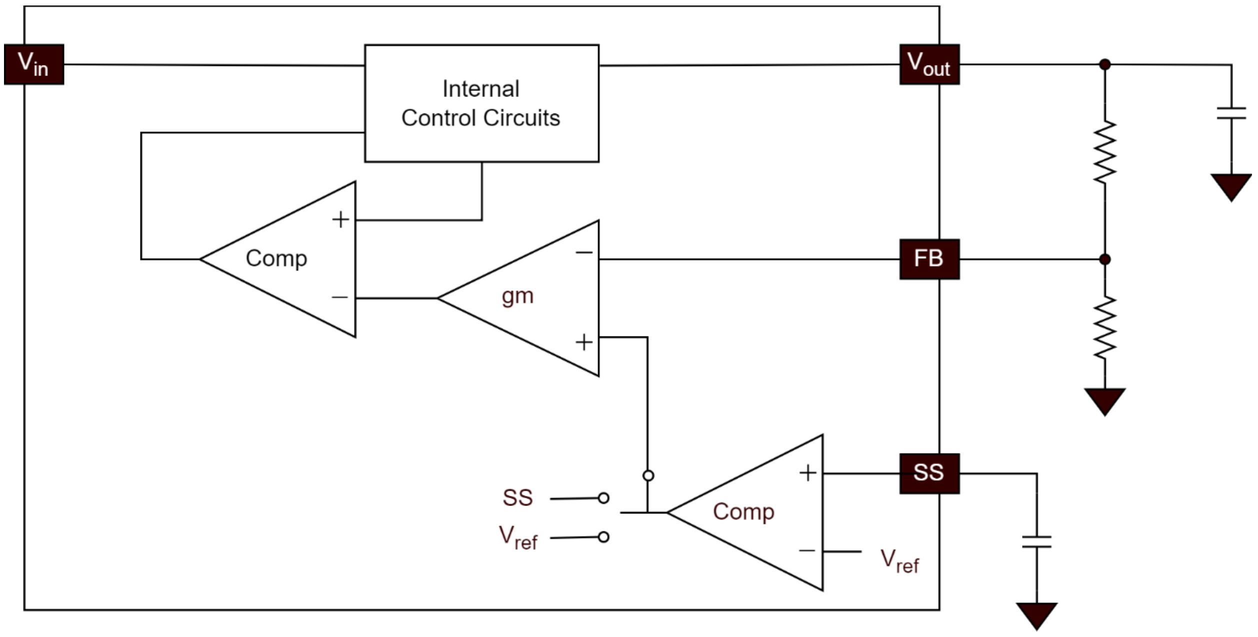


Figure 3.3: TPS61088 Partial Block Diagram [26]

3.2.2 Method for Controlling the Boost Converter

EN pin control (undervoltage lockout control)

The main purpose of the EN control pin of the boost converter is to set the undervoltage lockout (UVLO) threshold voltage. In this design, a DAC and a comparator are used to implement this method. The output voltage of the DAC, V_{DAC}, is controlled in software by the MCU. As shown in Figure 3.4, it is connected to *Vin* and non-inverting input of the comparator through a resistor network, which is used as voltage divider. The comparator compares the inverting input voltage, V_{Comp}, with its reference voltage V_{ref} and generates a High/Low signal to the EN pin. When the

V_{in} is lower than the UVLO threshold, it means that *V_{Comp}* is lower than *V_{ref}*, the comparator will disable the boost converter to let the input voltage increase. And when *V_{Comp}* is higher than *V_{ref}*, the boost converter will be enabled again by the comparator.

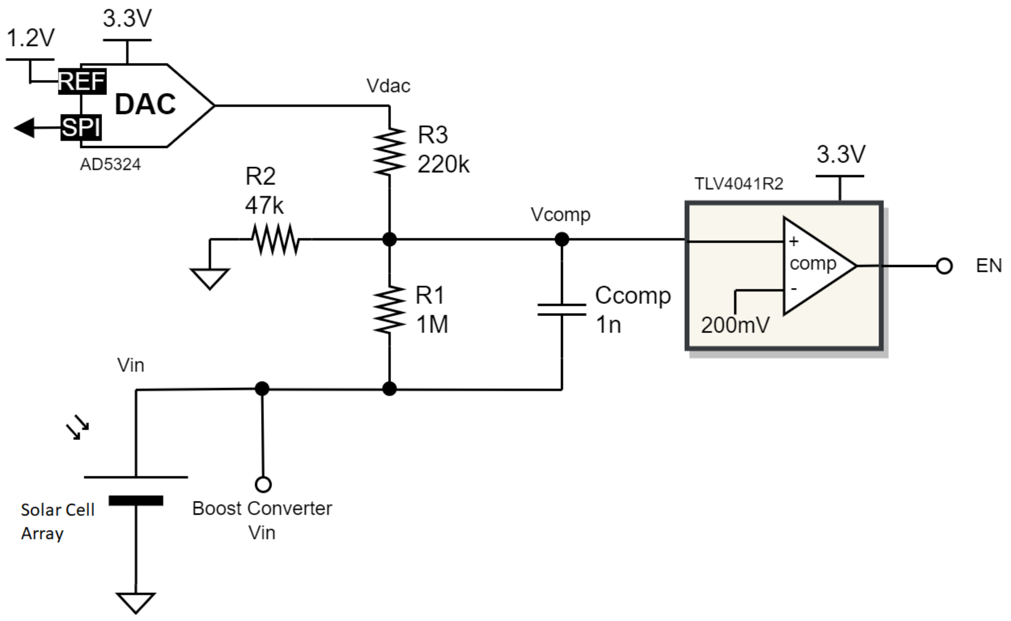


Figure 3.4: Schematic of the UVLO Control Circuit provided by Damkjar

According to Kirchhoff's Current Law (KCL), we can derive the relationship among these parameters as follows:

$$\frac{V_{in}-V_{Comp}}{R_1}+\frac{V_{DAC}-V_{Comp}}{R_3}=\frac{V_{Comp}}{R_2} \tag{3.1}$$

The comparator tends to make the *V_{Comp}* close to *V_{ref}*, so for convenience, *V_{Comp}* can be regarded as having close to the same value as *V_{ref}*, which is 0.2 V. *R_1* should have a high resistance to decrease current leakage, so a 1 MΩ resistor is used here. The EPS is designed to be able to operate when the MCU is down, so *Vin* should be set to a default value by the resistors with a high-impedance inactive the DAC output. There is also a capacitor connected in parallel to *R_1*. This is actually a type 2 ripple injection circuit that makes the ripple seen by the comparator larger relative to the DC voltage, by reducing the DC voltage with the resistors while the ripple passes through the capacitor.

SS pin control

An alternative implementation method is to adjust the voltage to match the error input signal to the boost converter, which is related to the output voltage. In this case it is the voltage at the soft-start pin (SS). In the TPS61088, the SS pin is intended to enable a soft start function that prevents high in-rush current during start-up [26]. According to the datasheet, it is recommended that the SS be connected to an external capacitor, as shown in Figure 3.3, to slowly ramp up the internal non-inverting input of the error amplifier. When the boost converter is enabled, the soft-start capacitor is charged with a constant current. During this period, the voltage of the SS pin, V_{SS}, is compared with the internal reference voltage, which is 1.2 V, and the lower pin voltage is fed into the non-inverting input of the second error amplifier. The inverting input of this amplifier is connected to the feedback pin (FB), which is the voltage of the FB pin, V_{FB}, will track the output of the first error amplifier: V_{FB} will equal to V_{SS} if it is lower than reference and it will stay at 1.204 V when V_{SS} exceeds this value. Therefore, controlling the SS pin is equivalent to controlling the duty cycle of the boost converter.

FB pin control

The SS pin control method is recommended in an MPPT design [28] using LT8611 step-down regulator. For TPS61088, this method has a relatively long response time and leads to a large ripple voltage at certain voltage level. A detailed discussion of this problem is included in Stefan Damkjar's PhD thesis. To avoid this problem, the SS pin control is improved to directly control the voltage of FB pin. The schematic of the FB control circuit is shown in Figure 3.5. The output of the DAC is connected to the non-inverting input of an op-amp and the *Vin* from solar panel with some resistors. The inverting input of the op-amp is connected to the solar panel array and the feedback from the output of the op-amp. This feedback loop design is expected to output a voltage very nearly equal to the voltage at non-inverting input. The equilibrium point is reached quickly as it is a feedback-stabilized system. This stability gives the op-amp the capacity to work in its linear mode, not being fully "on" or "off" as it was used for a comparator, with no feedback at all.

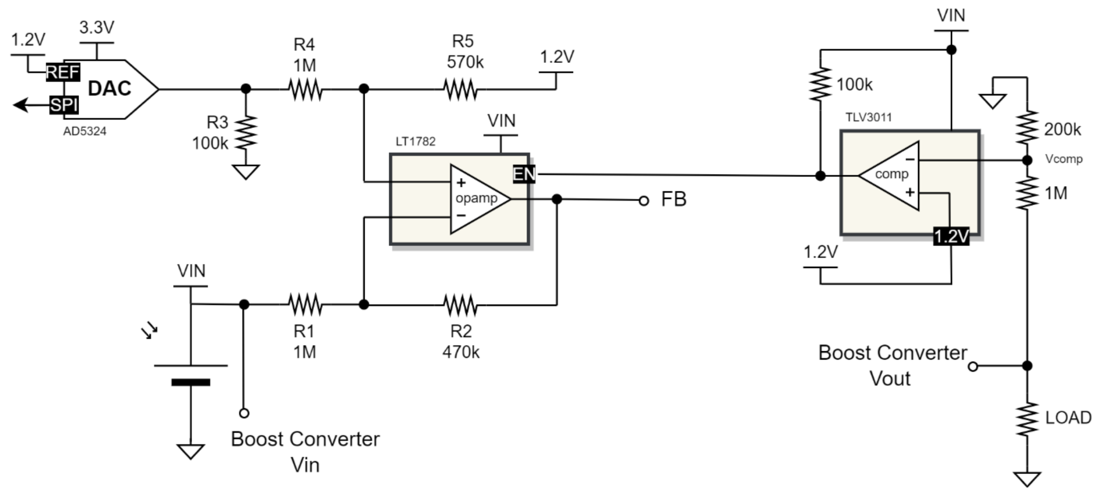


Figure 3.5: Schematic of a Possible Feedback Control Circuit provided by Damkjar

There is another circuit connected to the EN pin of the op-amp, which can disable the device and make the op-amp output high impedance. The schematic of this part is shown at the right in Figure 3.5. The comparator compares the voltage at V_{comp} and reference voltage 1.2 V. If the output voltage of the boost converter is higher than the threshold 8.2 V, which is set by the resistors of the voltage divider, the transistor will disable the op-amp to decrease the voltage.

With these two sub-circuits, the control signal V_{FB} can enhance voltage regulation accuracy and efficiency. And when the microcontroller is off, the circuit can still keep the voltage at a stable safe value.

This control circuit is an improved design derived from the circuit that controls the EN input to the boost converter. A comparison of the performance of the two alternative designs is included in Stefan Damkjar's PhD thesis. After considering the accuracy as well as the efficiency, the control circuit of the FB pin to the boost converter was selected as the final design of power conversion module.

3.2.3 Other Components

Digital-to-Analog Converter

The selected DAC was the Analog Devices AD5324 [29]. It is a 4-channel 12-bit converter that operates from a 2.5-V to 5.5-V supply, consuming only 500 μA at 3 V [29]. Its output voltage ranges from 0 up to the reference voltage. It is compatible with a standard SPI interface, so it can be controlled by an MCU. Furthermore, it operates at clock rates of up to 30 MHz, which is higher than the required frequency.

The resolution of the DAC is 2^{12}, which means that its minimum step size of output voltage is $V_{ref}/4096$. However, this value is limited in practice by the accuracy of other sensors. The step size must be large enough for the current monitor to detect it so that the MPPT algorithm can work properly. The reference voltage used here for the DAC is 1.2 V, so according to the measurement, the minimum step size that can be detected is 32x the least significant bit (LSB). Therefore, we can calculate the value of minimum step size, which is 2 mV.

Current Sensor

There is a current monitor in each power conversion module that measures the current and voltage. The sensor used here is the Texas Instruments INA226 [24], which is a current shunt and power monitor with an I2C interface. It measures the shunt voltage of the reference resistor and the bus voltage. With the internal logic functions, it can calculate the current and the power depending on the shunt voltage and the reference resistance. It is connected to the V_{in} pin of the boost converter. There are also alert and warning functions, but they are disabled here.

The sampling rate of the senor depends on the number of average measurements it takes and the conversion time of each measurement. There are several modes available which can be selected by the MCU. A larger number of average and a longer conversion time leads to a more accurate measurement. Considering the requirement of housekeeping task (which will be introduced in Chapter 4), the sampling rate should be higher than 20 Hz.

Depending on the datasheet of current sensor, four combinations of configurations that close to 20 Hz are available. Note that the total time for each measurement is "# of average" multiplied conversion time, times two (measuring both the shunt voltage and bus voltage). We tested it by measuring a constant current from a power supply and calculating the deviation over 500 times measurements. According to Table 3.2, 28.41 Hz (32.2 ms period) mode was used for the final configuration.

Frequency (Hz)	# of average	conversion time	Deviation of 500 Measurements
23.53	64	332 µs	1704.2
28.41	16	1.1 ms	1528.7
30.08	4	4.156 ms	1548.1
38.3	64	204 µs	1575.5

Table 3.2: INA226 Accuracy in Various Sampling Modes

Op-amp

The op-amp used in this circuit is the Linear Technology LT1782 [30]. It is a 200-KHz op-amp which operates from a supply voltage of 2.5 V to 18 V. It draws less than 55 µA of quiescent current and can drive loads up to 18 mA and still maintain rail-to-rail output drive capability. An unusual and important feature of this device is that it is able to operate with either or both of its inputs above the positive rail.

Comparator

The Burr-Brown TLV3011 [31] is a low-power, open-drain output comparator with 5 µA (max) quiescent current, input common-mode range 200 mV beyond the supply rails, and single-supply operation from 1.8 V to 5.5 V. It has integrated 1.242 reference voltage which can provide up to 0.5 mA of output current.

3.3 Power Storage

The power storage stage is a separate battery board which is connected to the main power bus, as shown in Figure 3.1. The battery pack consists of four Li-ion cells (Panasonic NCR18650B [32]) which are connected in 2S2P arrangement (two series strings connected in parallel). According to the datasheet of the cell [32], the specification of battery pack is as follows:

Nominal Capacity (at 25°C)	Typ. 6700 mAh
Charging Voltage	8.4 V
Charging Current	Std. 1625mA (for each series string)
Ambient Temp. (Charge)	10 – 45 °C
Ambient Temp. (Discharge)	-20 – 60 °C

Table 3.3: Specification of the Battery String [32]

Note that the output voltage of the boost converter is resistor-programmable. According to a research [33], charging the Li-ion battery with voltage lower than its theoretical value will extend the lifetime of the battery. So we set the real charging voltage to 8.1 V.

The battery protection circuit is also integrated into the battery board. There is one temperature sensor while each series pair has an independent heater which is controlled by the MCU. Each series pair also has one current sensor and two switches, which control the charging and discharging current. In this case, one string of solar cells can be isolated from the battery pack if there is any failure. All the sensors are configured by microcontroller through I2C protocol.

3.4 Power Regulation

The power regulation stage is implemented using three buck converters (TPS53319). They are responsible for regulating the 1.2-V, 3.3-V and 5-V power buses. The three converters are connected with some components according to the typical application schematic from the datasheet [34]. Each regulator is connected to a built-in protection module, which will reset the system if an overcurrent condition is caused by latch-up. There is no special functionality or digital control logic for this module, so it will not be discussed in detail here.

3.5 Power Distribution

The high-level design of the power distribution module is also shown in Figure 3.1. It consists of a jumper matrix generating 18 output channels that are connected to the regulated 1.2-V, 3.3-V, 5-V buses or the direct voltage from the main power bus. They supply power for other subsystems of the CubeSat, and so the configuration of each channel depends on the load requirement. There are also two internal channels supplying 1.2 V and 3.3 V for the microcontroller and other components on the EPS.

Each output channel has an independent control switch and overcurrent protection module, which will disable the single channel when an overcurrent condition is detected. The channel switches can also be controlled by the MCU, and they will be switched ON or OFF in different situations, as will be described in Chapter 4.

3.6 Protection Circuits

3.6.1 Latch-up and Overcurrent Protection Module

This module mainly consists of two components. One is the Texas Instruments INA381 [35], which is a current-sensing amplifier with integrated comparator. It is able to detect overcurrent conditions caused by latch-up by measuring the voltage developed across the current-shunt resistor and comparing that voltage to a user-defined threshold limit applied to the comparator reference pin. The ALERT pin of the INA381 is connected to an alert bus, which will assert when overcurrent happens. This sensor is fully configured by hardware and does not require any digital control logic.

The other component is the INA226, which has been introduced before [24]. It is connected to the same shunt resistor as the INA381. This sensor is mainly used to monitor data and store as housekeeping in MCU through I2C interface. It also has alert functions which can be configured to alert over-/undercurrent, over-/undervoltage and overpower conditions depending on the requirements. In this system, only the overcurrent alert function is used.

3.6.2 Watchdog Timer

The Maxim Integrated MAX16998A [36] watchdog timer is used in the EPS to detect and recover from failure conditions. It is integrated with two independent timers with different timeout periods.

The RESETIN pin is connected to the alert bus of the current sensors at the buck converters. When the overcurrent protection module leads the alert bus to low, the RESET pin will assert, which switches off the main power bus to reset (that is, power-cycle) the whole satellite, for a reset timeout period. This function is intended to protect the circuit from serious hardware failure.

Another timer petted by the microcontroller is for the watchdog timeout period. The WDI pin receives a signal from the MCU. Two consecutive WDI falling edges must occur at WDI within the watchdog timeout period, otherwise this will lead to a WDI fault. The watchdog timer clears when a falling edge occurs on WDI or whenever RESET is asserted. The ENABLE output pin asserts if three consecutive watchdog timeout periods have expired without a falling edge at WDI [36]. However, we are not using the ENABLE output pin in this EPS system. Instead, we use the other function of the RESET pin, which it will assert for a reset timeout period when a WDI fault occurs. Such a RESET will lead to a power-cycle of the satellite as well. In conclusion, the RESET pin will assert when a WDI fault occurs, or when the RESETIN pin asserts.

Both timeout periods are configured by capacitors connected to the SRT and SWT pins. The value tested on the system is a 100 nF / 200 nF capacitor at the SWT pin, which sets the watchdog timeout period to be about 1 s / 2 s, and a 1uF capacitor at the SRT pin to set the reset timeout period to be about 2.9 s.

Chapter 4: Software Design

This chapter introduces the software that was designed to run on the microcontroller that manages the EPS, including the housekeeping data, the power point tracking algorithm and other additional functions. Some implemented functions will be shown in tables as examples in the following sections. All of the project files and codes can be accessed on GitHub under the MIT License, with additional documentation.

The software design is divided into two layers: one is a set of low-level drivers that control the integrated modules and peripherals, along with abstraction functions that deals the interactions between the external interrupt controllers and processor. These software drivers are provided by the manufacturer, in this case Texas Instruments (TI). The other higher layer is an RTOS that provides event-driven and preemptive scheduling of interrupt requests. Functions such as the collection of housekeeping data and system monitoring, are called by tasks running in the RTOS multitasking environment. This layer is implemented in the C programming language.

This chapter first briefly introduces and discusses the requirements of the software design and some important peripherals and features of the embedded microcontroller. The chapter then covers details of the software architecture and functions.

Before discussing the microcontroller, one important thing should be noted: the final EPS board was designed and tested with Texas Instruments TMS570LS1224PGE (hereinafter referred to as TMS570) microcontroller, but the early software implementation and tests were done on a development board that had an RM46852PGE (hereinafter referred to as RM46) microcontroller. These two MCUs have very similar features, except for two important differences: the coupled two CPUs in the TMS570 support the big-endian byte numbering format while the RM46 supports little-endian. Also the operating clock frequency is set to the same as the maximum values, while TMS570 is 180 MHz and RM46's 220 MHz.

The original C code was compiled using the developing Integrated Development Environment (IDE), Code Composer Studio (CCS), which manages the endianness problem itself. The system frequency difference has little influence since the operating system clock frequency is 1 KHz. In conclusion, the difference between two MCUs has a very limited effect on the software design of the EPS. The features discussed in following sections are mainly based on TMS570 datasheet and technical reference manual. All of the CPU features are the same as in the RM46 if not otherwise noted.

4.1 Software Requirements

According to the requirements of the CubeSat, the software of the EPS board must provide the following basic features:

- Access to a non-volatile memory, which stores the configuration settings of the EPS board and stores an error log.
- An initialization and reset routine that initializes the whole system, including both the software and hardware components.
- Recording housekeeping data for the EPS, including the status of the boost converters, batteries, heaters, and output channels.
- Implementation of the MPPT algorithm that controls the power conversion module so that the boost converters draw the maximum power from the solar panels and supply current at several regulated voltages.
- A battery protection module, which controls the switches of the batteries and heaters and that ensures that the batteries are operated safely within their operating ranges.
- An output channel protection module which controls the switches of channels according to the channel limits.
- A user command interface that allows the users to easily modify configuration, and to monitor and control key parts of the EPS board.

The details of each requirement will be discussed in detail in the following sections.

4.2 Microcontroller Features

The TMS570 is a member in the Hercules Safety MCU family produced by Texas Instruments. The MCUs in this family are designed to allow customers to develop products for safety-critical industrial and transportation applications. It is an ARM architecture-based microcontroller with dual ARM Cortex-R4F cores that detects failures at the core boundary, in which special measures in processor layout, clock distribution, power distribution, reset distribution, and temporal diversity are all implemented to mitigate common causes of failures in the logical CPU and its checker [37]. There is built-in Single-Bit Error Correction Double-Bit Error Detection (SECDED) module for both Flash and RAM memory. There are also built-in hardware Built-in Self-Test (BIST) controllers that provide a claimed high level of diagnostic coverage for the CPUs and SRAMs in the system. Compared to equivalent software-based self-test solutions, the lock-step CPU architecture runs the same set of operations at the same time in dual cores to determine if there is a fault, so it executes faster and consumes less memory [37].

The TMS570 MCU enhances the lock-step CPUs with peripherals for real-time control-based applications, including two timing coprocessors with up to 40 input/output terminals and a 12-bit analog-to-digital converter (ADC) that supports up to 24 multiple inputs. It also has multiple communication interfaces including three SPIs, one SCI, one I2C and three CANs [37].

With all these integrated safety features and communication and control peripherals, the TMS570 is an ideal solution for high-performance real-time CubeSat EPS control applications which has mission-critical reliability requirements. Therefore, it was selected as the MCU used to implement the EPS system.

4.2.1 Memory

The TMS570 has a 1.25-MB integrated Flash as well as a 192-KB data RAM. Due to the harsh operating environment in space, the data stored in non-volatile memory has higher possibility to be damaged. As a MCU designed for safety-critical applications, both Flash and RAM are protected by the built-in SECDED module using an Error Correction Code (ECC) algorithm. This encoding algorithm allows the decoder to correct 1-bit errors by itself and to detect 2-bit errors. The ECC check bits of Flash can be generated by an external tool, in this case is CCS IDE. With certain configuration, this tool can calculate the ECC check bits and flash them to the memory with the code. ECC check bits of RAM are calculated in software by the CPU. There are 8 bits of ECC for every 64 bits of data accessed from Flash and RAM [37].

The error signaling module (ESM) of the MCU will generate an error event whenever an ECC error occurs. The 1-bit correctable error leads to a group 1 channel 6 error event, while the address of the error will be stored in the FCOR_ERR_ADD register and the FEDACSTATUS register flags will be updated to indicate the type of error. This error event can also trigger an interrupt, which is disabled in this system. Meanwhile, the ESM will generate a group 3 channel 7 error event whenever an uncorrectable (2 or more bits) error occurs. The address of the error will be stored in FUNC_ERR_ADD register and the FEDACSTATUS register flags will be updated to indicate the type of error as well. This error event is not user-configurable, and it will generate an abort instruction instead of the interrupt. The abort handler will lead the CPU into an infinite loop. In this case, the microcontroller is not able to pet the watchdog timer, and this will cause a power-cycle reset of the system (this will be discussed later).

The ECC controller is implemented in hardware inside the two Cortex-R4F processors and is enabled after the device coming out of the reset sequence.

Figure 4.1 shows the memory-map of this microcontroller. The Cortex-R4F CPU uses a 32-bit address bus, which allows it access a memory space of 4 GB. This space is divided into several regions for different memory selections. Note that the figure only lists the memory regions related to the EPS software; the other regions are omitted.

The address starting at 0x00000000 is the main flash memory which stores instructions. Compiled operating system software is stored in this region. This is also the reset vector location. The processor starts execution from the reset vector address of 0x00000000 whenever it gets reset. The CPU data section starts at memory address 0x08000000 and it stores readable and writeable data structures such as global variables. The ECC check bits of the Flash and RAM can be accessed starting from address 0x08400000 and 0xF0400000, respectively.

The mirrored Flash memory region starting at 0x20000000 is physically the same memory as the main Flash memory and it is basically used for diagnostics by the BIST. Any data written to the Flash will be mapped to both the Flash region and the mirrored Flash region at the same time. Users cannot erase or program the Flash using the addresses in the mirrored Flash region.

The region starting at address 0xF0200000 is used to create an emulated Electrically Erasable Programmable Read-Only Memory (EEPROM). It is implemented in hardware as a separate Flash bank that is dedicated for use as an emulated EEPROM. This device supports 64 KB of Flash for emulated EEPROM. The factory copies and the boot copy of EPS system, which will be introduced in later section, are stored in this region.

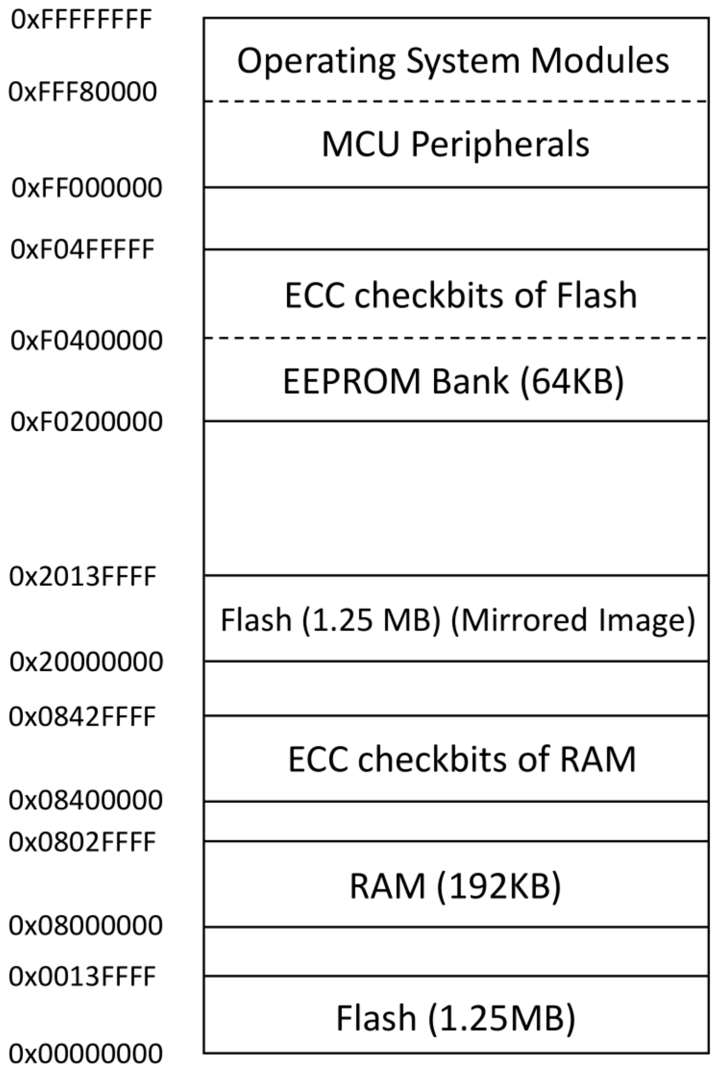


Figure 4.1: Partial Memory-map of the Cortex-R4F [37] Microcontroller

At the top of the memory map is the address for peripherals and system modules, which includes the Direct Memory Access (DMA) control register, flash wrappers, power management and so on.

4.2.2 Clocks

The TMS570 device supports up to seven clock sources. Table 4.1 lists the clock sources that are used in the developed EPS controller design.

The main oscillator OSCIN is the primary clock source of the microcontroller. It is enabled by connecting the appropriate fundamental resonator/crystal and load capacitors across the external OSCIN and OSCOUT pins [38].

The phase-locked loop (PLL) is a circuit in the microcontroller that is used to multiply the input frequency of OSCIN to a higher frequency. It is the default clock source for most of the clock domains.

The on-chip low-power oscillator (LPO) uses a relaxation oscillator to generate an internal clock whose frequency is not tightly controlled. It provides two clock sources: a low-frequency (LF) one, which is nominally 80 KHz, and a high-frequency (HF) one, which is nominally 10 MHz [38].

Clock Source Name	Description
OSCIN	Main oscillator. This is the primary clock for the microcontroller and is the only clock that is input to the phase-locked loops. The oscillator frequency must be between 5 and 20 MHz. The oscillator used on the prototype board is 16 MHz.
PLL1	This is the output of the main PLL. The PLL can modulate its output frequency can be programmed in a controlled manner as a multiple of OSCIN to reduce the radiated emissions.
LF LPO	This is the low-frequency output of the internal reference oscillator, which is independent of OSCIN. This is typically an 80-KHz signal which is used by the real-time interrupt module for generating periodic interrupts to wake up the MCU from a low power mode.

Table 4.1: Partial Clock Sources of the TMS570 Microcontroller [37]

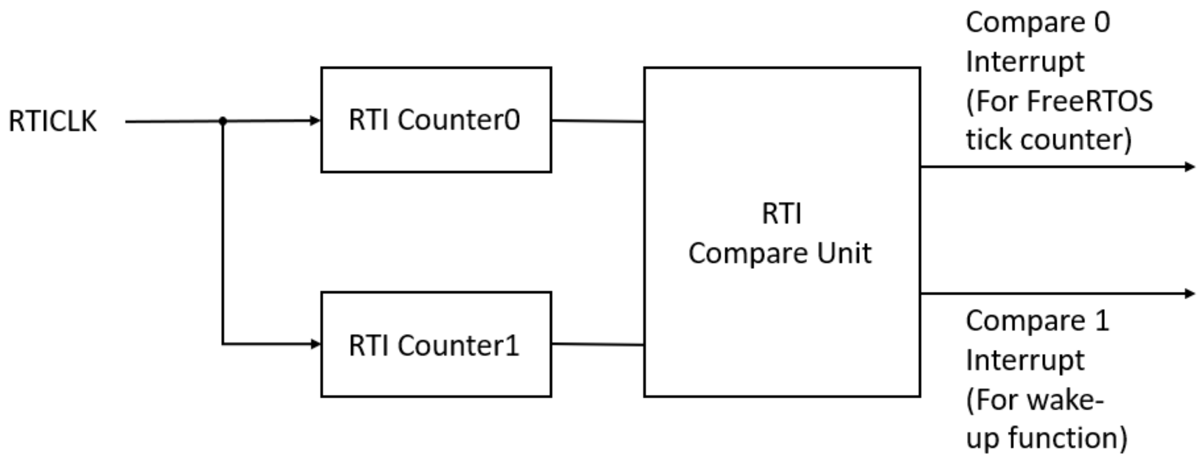
The clocking on this device is divided into multiple clock domains for flexibility in control and clock source selection. On this device, there are 10 clock domains in all. The used clock domains are listed in Table 4.2.

GCLK is used as the CPU clock and HCLK is used by the high-speed modules, and they must be the same frequency. Therefore, we set both of them to the maximum frequency of the microcontroller to obtain the best performance. VCLKs and VCLKAs are divided down from HCLK, and their frequency can be no greater than half of HCLK. RTICLK is used by the RTOS as well as the wake-up functions, which will be introduced later. It can be mapped to either OSCIN or LF LPO according to the mode selected (this will also be described later), so the frequency is unfixed.

Clock Domain	Reference Source	Frequency	Description
GCLK	PLL1	180 MHz	Clock domain used by Cortex-R4F CPU operating in lock-step.
HCLK	PLL1	180 MHz	Clock domain used by the high-speed system modules such as Flash memory and DMA.
VCLK 1- 4	PLL1	90 MHz	Clock domain used by some system modules.
VCLKA 1,3,4	VCLK	90 MHz	Clock domain dedicated for peripheral controllers.
RTICLK	OSCIN/LF LPO	16 MHz / 80 KHz	Clock domain dedicated for timebase generation in the Real-Time Interrupt (RTI) generation module.

Table 4.2: Partial Clock Domains of the TMS570 Microcontroller [37]

The real-time interrupt (RTI) module provides timer functionality for the operating systems and for benchmarking code. Figure 4.2 illustrates the simplified high-level block diagram of this module.



RTICLK
RTI Counter0
RTI Counter1
RTI Compare Unit
Compare 0 Interrupt (For FreeRTOS tick counter)
Compare 1 Interrupt (For wake-up function)

Figure 4.2: Simplified RTI Block Diagram [37]

There are two independent counter blocks in the RTI module. They both depend on the RTICLK clock domain but generate different frequencies. The compare unit compares the counters with programmable values and then generates four independent interrupts. Each compare interrupt can be selected to use either counter block 0 or block 1 as the compare source. In this system, only two compare registers are used. Compare 0 is based on counter block 0 and compare 1 is based on block 1.

4.2.3 Low-power Mode

As an application system used in a CubeSat, the design of the EPS software should employ power saving strategies since the power capacity is limited. The TMS570 microcontroller has *low-power modes* (LPM) for power saving, which provide a trade-off of the current used during low-power versus functionality and fast wakeup response.

The typical software sequence to enter a low-power mode has four steps [37]:

- Program the flash banks and flash pump modes to be in “sleep” mode.
- Disable the clock sources that are not required to be kept active.
- Disable the clock domains that are not required to be kept active.
- Idle the Cortex-R4F core.

Since the system can choose to disable a particular clock source, clock domain and peripheral, there are many possible low-power modes that are configurable by the application. Table 4.3 lists three particular modes and their typical characteristics. Note that the peripherals for the CAN and SCI interfaces also use the corresponding active clock source in LPM (they are configured to use LF LPO in sleep mode).

In "doze" mode, the main oscillator is kept active and RTICLK uses it as the clock source. Other unused clock sources and domains are disabled and will be enabled again only after the system is awakened by an interrupt. The snooze mode is similar to doze mode, but the LF LPO is used, instead of OSCIN, as the clock source for RTICLK.

Mode Name	Active Clock Source	Active Clock Domain	Wake Up Option
Doze	Main oscillator	RTICLK	RTI interrupt, GIO interrupt, CAN message, SCI message
Snooze	LF LPO	RTICLK	RTI interrupt, GIO interrupt, CAN message, SCI message
Sleep	None	None	GIO interrupt, CAN message, SCI message

Table 4.3: Typical Low-Power Modes [37]

In "sleep" mode, all of the clock sources and domains are disabled, and the MPU can only be woken up out of sleep mode by external interrupts. This mode saves a lot of power, but the MPU cannot wake up out of sleep mode by itself, which doesn't satisfy the requirement of the EPS system.

Mode	Power (mW)

Doze	8.28
Snooze	19.2

Table 4.4: Power Consumption of LPMs

The comparison of power consumption of doze and snooze mode are shown in Table 4.4, which was tested with a 1.2-V power supply. Doze mode surprisingly consumes less power than snooze mode, although it uses the main oscillator as the active clock source. Therefore, doze mode was selected as the low-power mode to be used for the EPS software.

4.3 System Features

4.3.1 Selection of the RTOS

According to the embedded markets study published by EETimes.com and Embedded.com [39], 65% of the current embedded projects use an OS/RTOS. If we exclude operating systems based on Microsoft Windows or Linux, the most popular RTOS for small MCUs is FreeRTOS (20%), followed by uC-OS/II (4%) and Keil RTX (4%). Since Keil RTX doesn't support the ARM Cortex-R architecture, the candidate operating systems of the EPS were limited to FreeRTOS and uC-OS/II.

Parameters	FreeRTOS	uC-OS/II
Scheduling policies	Pre-emptive and cooperative	Pre-emptive
Memory management	Automatic or manual dynamic allocation from the RTOS heap	Manual fixed-sized memory blocks from heap partitions
Interrupt	Standard interrupt service routine (ISR) handling, application-controlled deferred interrupt handling, and centralized deferred interrupt handling	Functions to send direct or delay signals, flags, and messages from ISRs to tasks

Table 4.5: Key Characteristics of FreeRTOS and uC-OS/II [40], [41]

Table 4.5 shows the major characteristics of importance between the two systems. uC-OS/II only supports pre-emptive scheduling while FreeRTOS supports both pre-emptive and cooperative scheduling. Since our system is designed to be pre-emptive, the two RTOSs can be regarded as the same in their scheduling support. For memory management, they have quite different memory allocation methods. uC-OS/II consumes at least 5 KB RAM while FreeRTOS can run very well with only 2-3 KB [42]. FreeRTOS is also more flexible on memory allocation. However, compared to TMS570's 192-KB RAM, this difference can be ignored. Both FreeRTOS and uC-OS/II provides a set of functions and mechanisms that enable developers to implement an interrupt management strategy in a simple manner. According to published RTOS studies, uC-OS/II performs slightly better than FreeRTOS at task switching time test (about 18% on ARM Cortex-M0) [43]. The switching time difference between the two OSs will be less than 2 us from lowest priority task to highest priority task, which is still acceptable.

Since the above characteristics are similar in performance, the main consideration then depends on the applicability of the system. Since FreeRTOS is open-source while uC-OS/II is commercial, the TMS570 has more supporting functions and built-in codes for FreeRTOS. Therefore, FreeRTOS was selected as the RTOS for the EPS embedded system.

4.3.2 Software Architecture

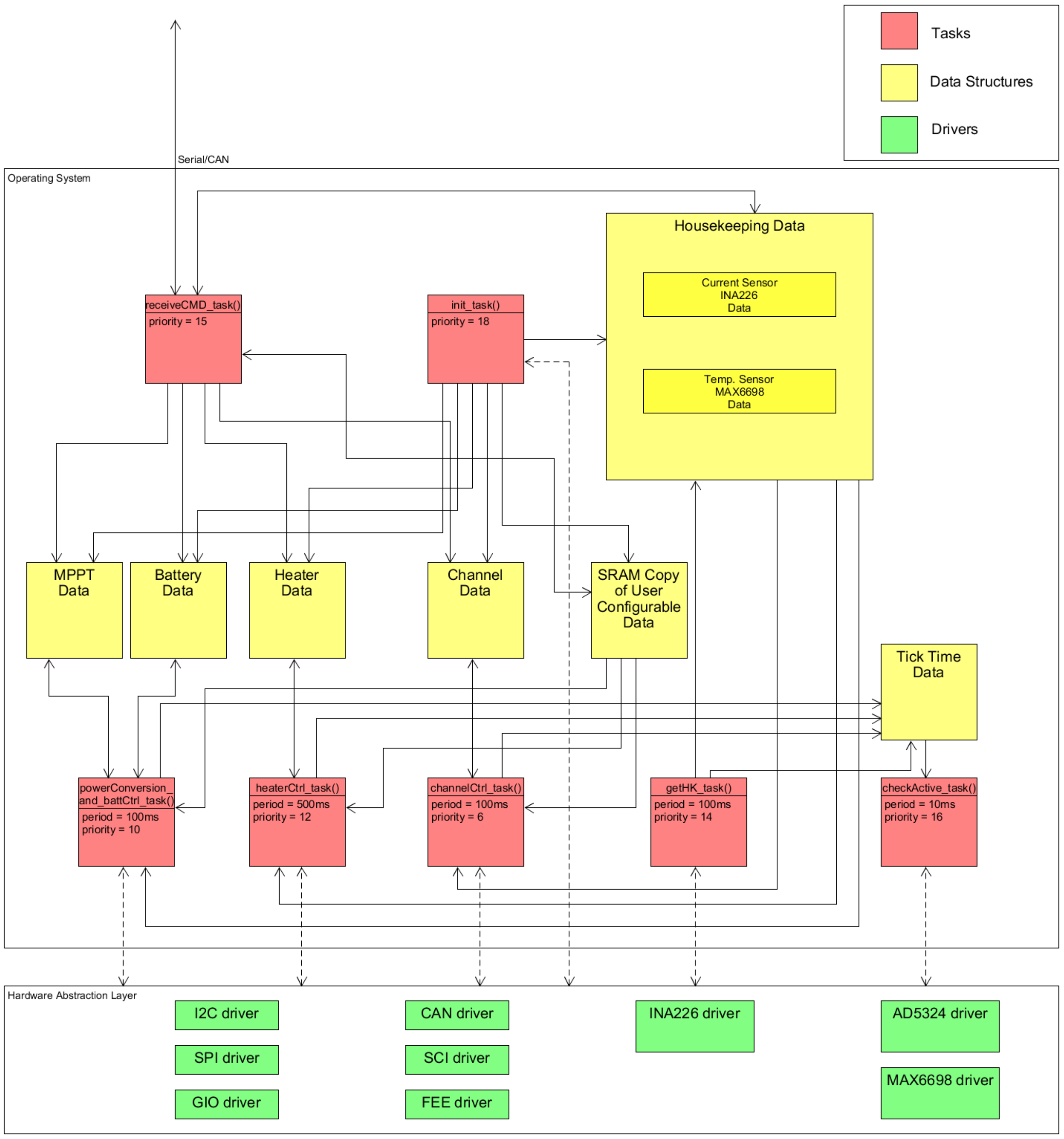
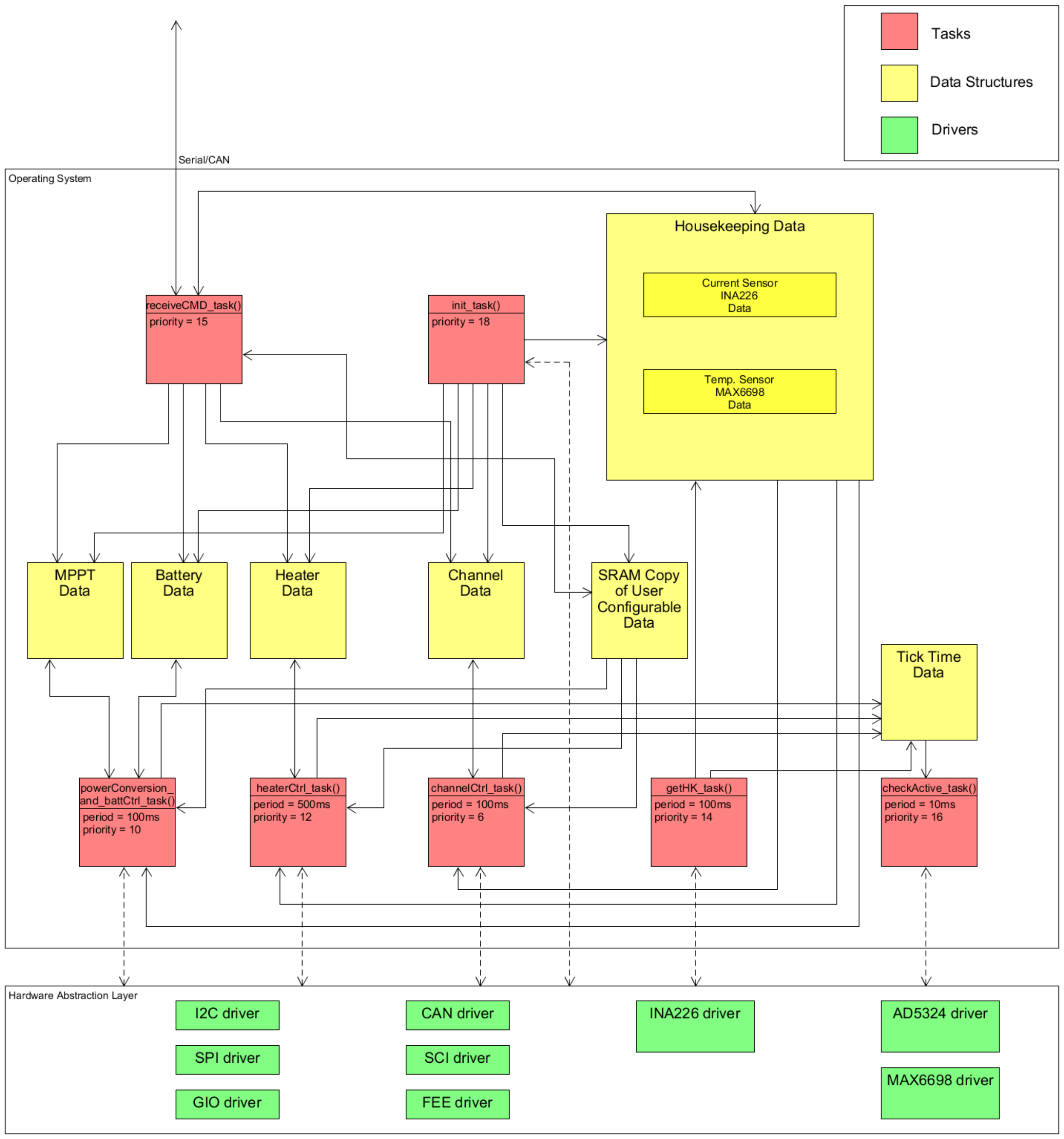

Tasks
Data Structures
Drivers
Serial/CAN
Operating System
Housekeeping Data
Current Sensor INA226 Data
Temp. Sensor MAX6698 Data
receiveCMD_task() priority = 15
init_task() priority = 18
MPPT Data
Battery Data
Heater Data
Channel Data
SRAM Copy of User Configurable Data
Tick Time Data
powerConversion_and_battCtrl_task() period = 100ms priority = 10
heaterCtrl_task() period = 500ms priority = 12
channelCtrl_task() period = 100ms priority = 6
getHK_task() period = 100ms priority = 14
checkActive_task() period = 10ms priority = 16
Hardware Abstraction Layer
I2C driver
SPI driver
GIO driver
CAN driver
SCI driver
FEE driver
INA226 driver
AD5324 driver
MAX6698 driver

Figure 4.3 shows the software architecture that was developed to satisfy the software requirements. It is based on the FreeRTOS multitask scheduling environment. The direction of the solid arrows represents the flow of data. All the tasks, except the initialization task, are scheduled to execute at predefined frequencies. This section gives a brief introduction for each task, and the details will be discussed in the following sections.

The initialization task, init_task, is responsible for initializing data structures and for creating the other system tasks.

The checkActive_task checks the active status of all the other tasks and pets the watchdog timer.

The receiveCMD_task receives commands from the OBC/ground station (or from the PC in debugging mode) and then executes them.



Tasks
Data Structures
Drivers
Serial/CAN
Operating System
Housekeeping Data
Current Sensor INA226 Data
Temp. Sensor MAX6698 Data
receiveCMD_task() priority = 15
init_task() priority = 18
MPPT Data
Battery Data
Heater Data
Channel Data
SRAM Copy of User Configurable Data
Tick Time Data
powerConversion_and_battCtrl_task() period = 100ms priority = 10
heaterCtrl_task() period = 500ms priority = 12
channelCtrl_task() period = 100ms priority = 6
getHK_task() period = 100ms priority = 14
checkActive_task() period = 10ms priority = 16
Hardware Abstraction Layer
I2C driver
SPI driver
GIO driver
CAN driver
SCI driver
FEE driver
INA226 driver
AD5324 driver
MAX6698 driver

Figure 4.3: Software Architecture

Housekeeping data is collected by the getHK_task, which reads from all the sensors and updates the data in the housekeeping data structure.

The powerConversion_and_battCtrl_task includes the MPPT module that controls the boost converter and the battery control module, which is responsible for the functions related to battery charging.

Battery heaters and output channels are controlled by heaterCtrl_task and channelCtrl_task, respectively. They control the switches of heaters and channels depending on various functions.

4.3.3 FreeRTOS Features

The FreeRTOS real time kernel measures time intervals using a system variable called the tick counter. The RTOS tick interrupts, which are produced by a hardware counter, increments the tick count with high accuracy, which allows the real time kernel to produce accurate time periods that produce accurate task scheduling frequencies [40]. For the selected prototype system, one of the RTICLK interrupts was used as the source of tick interrupts.

Each time the tick count is incremented, the FreeRTOS kernel will check whether there is a task that needs to be woken up or unblocked, and then do the corresponding task switch. However, a faster tick rate is not necessarily better. Since the kernel checks task switch for each period of tick, it will consume more power for unnecessary checking. If there are tasks in the same priority, a high tick rate will also lead to a waste of time on task switching. Considered the smallest repetition period of tasks, 10 ms (this will be discussed later) according to Table 4.6, the appropriate tick rate of this system was determined to be 1 kHz.

Table 4.6 also shows the runtime of each task in a simulation system on the prototype board. The receiveCMD_task may execute for longer time in practice because the runtime analysis was tested with no external commands active. Note that the larger the number, the higher the priority. According to the test, the idle task occupies a large fraction of the system's runtime. Since the system does nothing in most of the idle time, we can set MCU into LPM, when it is otherwise idle, to save power.

Task Name	Priority	Repetition period (ms)	Total Runtime
receiveCMD_task	18	100	<0.0001%
checkActive_task	16	10	<0.0001%
getHK_task	15	100	37.84%
heaterCtrl_task	12	500	<0.0001%
powerConversion_and_battCtrl_task	10	100	<0.0001%
channelCtrl_task	6	100	<0.0001%
IDLE	0	Not applicable	62.16%

Table 4.6: Task Runtime Statistic of Simulation System
(on the prototype board reading 31 times from the same current sensor)

Idle hook in FreeRTOS allows a function to be called when the scheduler has no task to run, so this can be used to enter an MCU low power mode. The task scheduler calculates the expected idle time whenever the system starts executing the idle task. If this period is longer than 2 ticks, it will call the user defined idle hook function to cause the system to enter into LPM. When the expected idle time has elapsed, or any interrupt occurs, the wake-up function will be called by corresponding ISR to wake up the whole system.

Unfortunately, as of writing this thesis, the LPM design has not been implemented successfully. The RTI interrupt was indeed able to wake the system up from LPM in a bare metal implementation, but it failed to work in FreeRTOS for some unknown reasons. The problem seems to lie within FreeRTOS when it initializes MCU hardware. This code still needs improvement, as described in the future work. The calculated power consumption of the system in various modes, according to Table 4.4 and Table 4.6, is shown below. Both LPMs can save more than half of the power consumption compared to normal mode.

Mode	Power (mW)
Normal	184.2

Doze	75.0
Snooze	81.7

Table 4.7: Power Consumption of Various System Modes

4.4 EEPROM Driver and Data Structures

As mentioned before, some important data is stored in the non-volatile memory called EEPROM so that the data can be retrieved after a system reset. These data include in configuration settings which are used to initialize the system and create error log entries that record error messages.

4.4.1 EEPROM Driver

The EEPROM driver used in this project is provided by TI and is called TI Flash EEPROM Emulation (FEE) Driver. It includes a set of software functions that use a 64-KB sector of the on-chip Flash memory as the emulated EEPROM.

Function name:	TI_Fee_WriteSync	
Syntax:	Std_ReturnType TI_Fee_WriteSync (uint16 BlockNumber, uint8* DataBufferPtr)	
Parameters (in):	BlockNumber	Number of logical blocks, also denoting the starting address of that block in Flash memory
	DataBufferPtr	Pointer to data buffer
Return value:	Std_ReturnType	E_OK: The write job was accepted by the TI_Fee module.
		E_NOT_OK: The write job was not accepted by the TI Fee module.

(a)

Function name:	TI_Fee_Read
Syntax:	Std_ReturnType TI_Fee_Read (uint16 BlockNumber, uint16 BlockOffset,

		uint8* DataBufferPtr, uint16 Length)
Parameters (in):	BlockNumber	Number of logical blocks, also denoting start address of that block in Flash memory
	BlockOffset	Read address offset inside the block
	DataBufferPtr	Pointer to data buffer
	Length	Number of bytes to read
Return value:	Std_ReturnType	E_OK: The write job was accepted by the TI_Fee module.
		E_NOT_OK: The write job was not accepted by the TI Fee module.

(b)

Table 4.8: FEE Driver API Examples [44] (a) Function to program data to a Block, (b) Function to read data from a Block

The EEPROM Emulation Flash bank is divided into two or more Virtual Sectors. The initialization routine (TI_Fee_Init) identifies which Virtual Sector is to be used and marks it as Active. The data is written to the first empty location in the Active Virtual Sector. If there is insufficient space in the current Virtual Sector to update the data, it switches over to the next Virtual Sector. Each Virtual Sector is further partitioned into several Data Blocks. The minimum unit of writing to this non-volatile memory at a time is the Block. Each Block can be considered as an array of bytes [44].

Table 4.8 shows the API functions that write and read from EEPROM. The read and write functions of this driver only support the uint8 type data, and so all the data must be converted into an array of uint8 bytes. The writing function updates an entire Block when it is called, while the reading function supports an offset as input parameter, which allows users to read the Block starting from any byte.

4.4.2 Cyclic Redundancy Check Data Protection

As mentioned above, the built-in ECC controller can only correct the 1-bit errors and it causes the satellite to start a reset cycle whenever 2-bit error occurs. To increase the robustness of the system, the built-in ECC protection for EEPROM is disabled and a customized Cyclic Redundancy Check (CRC) protection function is created to solve the potential 2-bit error in configuration data.

The CRC code is an error-detecting code, which has been widely used for many years in digital networks and storage devices to detect accidental changes to stored and/or transmitted digital data [45]. It encodes data by adding a fixed-length check value, which is called checksum. Whenever the encoded data is read, the custom functions will perform a CRC calculation on the data block and check the remainder. If the remainder equals zero, then the data passes the CRC constraint, and the data block is declared to be error-free. The checksum can be easily calculated by dividing the raw data with a given polynomial.

In general, longer CRC checksums are used to protect longer data blocks. The probability that errors will go undetected using an *n*-bit CRC is $1/(2^n)$ [46]. In conclusion, the greater the number of CRC bits, the better protection it will provide. However, a longer bit-size CRC also requires slightly more power to calculate it. As the application on microcontroller, we need to choose the best CRC polynomial for the system.

CRC standard	Polynomial	Max. data length at HD=4 (bit)	Max. data length at HD=5 (bit)
CRC-8	$x^8+x^5+x^4+1$	119	9
CRC-12	$x^{12}+x^{11}+x^3+x+1$	2035	53
CRC-16	$x^{16}+x^{15}+x^2+1$	32751	241
CRC-32	$x^{32}+x^{26}+x^{23}+...+x^2+x+1$	2147483615	65505

Table 4.9: Comparison of Some Popular CRC Standards [46]

There are a lot of reliable CRC standards available. Table 4.9 shows the features of some popular standards. Hamming Distance (HD) is the minimum number of errors that could transform one error-free CRC-protected data block into another seemingly error-free CRC-protected data block, which is used to characterize error detecting power of a code. We should select a standard code that has the smallest polynomial with the largest HD that covers required data length [46]. HD ≤ 3 is barely an improvement over good checksum, while HD ≥ 6 will likely be overkill [46], so only the code and block length combinations when HD=4 or 5 are considered. The data blocks which will be stored in EEPROM in this system contain 200 - 500 bytes. According to Table 4.9, only the CRC-32 code is able to provide such data lengths with both HD=4 and HD=5. Thus, the CRC-32 standard was chosen for EEPROM data protection, and so the checksum length is 4 bytes.

Function name:	crc32_calculate	
Syntax:	uint32_t crc32_calculate (uint8_t *data, uint16_t length)	
Parameters (in):	data	Pointer to the array of the data to be checked
	length	Length of the array in bytes
Return value:	uint32_t	Return the calculated checksum

Table 4.10: CRC-32 Checksum Calculation Function

4.4.3 Configuration Settings

All of the EPS configuration settings are stored in a data structure named system_config_t. The settings include the configuration version number, the configuration of all the sensors and channels, and other important parameters such as the threshold values for the battery protection module, the initial values for the DAC, etc. The size of one instance of the structure is 454 bytes. Here is a part of the corresponding C type definition how the data structure is implemented (the complete data structure is included in Appendix A):

```
typedef struct
{
   uint16_t configuration_version;                              //The version number of the configuration
   uint16_t current_monitor_alert_mA[NUM_OF_INA226_MONITOR];  //mA. overcurrent alert for ina226 of
                                                                  current monitor module
   uint16_t current_monitor_Rshunt[NUM_OF_INA226_MONITOR];     //mΩ. Shunt resistance for ina226 of
                                                                  current monitor module
   …
   uint16_t batt_charging_current_limit_mA;                    //mA. battery charging current
   uint16_t batt_discharging_current_limit_mA;                 //mA. battery discharging current
```

```
    …
    uint16_t dac_init;                                              //initial output value of DACs
    uint16_t dac_stepsize_init;                                     //initial step size of DACs
    …
    int32_t batt_charging_temp_min_c;                               //C. battery minimum charging temperature
    int32_t batt_charging_temp_max_c;                               //C. battery maximum charging temperature
    …
    sensor_config_t sensor_config_data;                             //data structure stores sensors' configuration
    channel_config_t chan_config_data[NUM_OF_CHANNELS];             //data structure stores configuration for
                                                                      output channels
}system_config_t;
```

To guarantee the reliability of the data, configuration settings are stored in three redundancy copies, and each copy has a CRC checksum that is generated by the system. Therefore, each CRC-protected copy is 458 bytes in size and is stored in an independent block in EEPROM.

The first copy of the configuration data is called the reboot copy. When the system is reboot or reset, it reads the data from this copy and then implements the configuration and performs initialization. Every time the user commands or tasks modify the configuration settings, the change is updated to the reboot copy by an internal function (along with a recomputed CRC checksum)

The other two copies are called factory copy 1 and 2, which are separate backup copies of the reboot copy. They contain the same data as the reboot copy when the satellite is launched and then remain unaltered. When the data in the reboot copy is damaged (it fails a CRC check), the system will read from these copies to avoid using a corrupted configuration setting. These two copies are backups for each other. If both of them are damaged, which should be a rare situation, there is also an independent command that would be sent from the ground station to reload them.

4.4.4 Error Message Log

Logs that record errors as they occur are very important for system maintenance and error recovery. Log entries provide information for failure diagnosis and should greatly reduce the system recovery time. Some of the errors should be solved by the system itself. Therefore, a simple error log format is used to record the critical error messages.

Each error message includes an identifying type number for the error, a related parameter value of the error, and a timestamp. Both type of the error and the data of the error are recorded in uint8 format while the timestamp includes a uint32 data and a uint16 data (timestamp will be introduced in later section). Hence, one error message is 8 bytes in size.

Table 4.11 shows the types of errors what will be logged. For each error type the table gives the corresponding number and the possible error values. The complete value of each error can be found in the corresponding header files.

Type of the error	Corresponding number of the error	Possible value of the error	
CRC failure	1	1	Factory copy 1
		2	Factory copy 2
		3	Reboot copy
Task not created	2	1	Initialization task
		2	Command receiving task
		3	Check active task
		…	
Task inactive	3	1	Initialization task
		2	Command receiving task
		3	Check active task
		…	
Watchdog timeout	4	1	OBC command timeout
		2	Ground station command timeout
Sensor overcurrent alert	5	1	Sensor of buck converter 1
		2	Sensor of buck converter 2
		3	Sensor of buck converter 3
		…	

Table 4.11: The Type of Errors and Their Corresponding Numbers and Values

A CRC failure happens when a CRC checksum calculation fails because a damaged data block (including the checksum) is read from EEPROM. The task-not-created error may happen during the initialization sequence, and it will lead to a software reset. The task-inactive error and the watchdog-timeout error will be reported by the check active task, as will be discussed later. The sensor-overcurrent-alert error is used to record the overcurrent condition of the current sensors at the power bus. The above three kind of errors will cause a power cycle reset of the whole satellite and users can retrieve the reason of the reset from error log.

The error buffer stored in RAM is implemented as a circular buffer which can save 100 error messages. There are two uint8 variables in the buffer. One records the number of errors stored and the other points to the tail of the error buffer. When the buffer is full, the new error message will overwrite the oldest one depending on the pointer variable. In conclusion, the error buffer always stores the latest 100 messages, and it is 814 bytes in size (with three CRC checksums, one for error message and two for two parts of timestamps). Since most of the logged error will lead to a system reset, the error buffer will be updated to the EEPROM every time the new message is recorded to avoid data loss.

4.5 Initialization and Reset

4.5.1 MCU Initialization Sequence and Available Reset Types

The sequence code for initialization sequence and configuration data of a Hercules MCU is generated using TI-provided software called the Hardware Abstraction Layer Code Generator (HALCoGen) and modified by programmers as required by the application. The whole sequence is complicated but not all of them are relevant to this project. Therefore, only some key steps are summarized and listed below (in order) [47]:

- Initialize the CPU registers and FPU registers, including the stack pointers
- Enable the flash interface module and SECDED logic to enable access to Flash memory
- Handle the cause of reset to determine whether to continue with the start-up sequence (See Table 4.12 for details)
- Enable the clock sources
- Release the peripherals from their reset and enable the clocks to all peripherals

- Set up the Flash bank and pump power modes
- Run the self-test on the SECDED logic embedded inside the Flash module
- Map the device clock domains to the desired clock sources
- Run the self-test on the CPU RAM using the BIST controller
- Initialize the CPU RAM and the related ECC region
- Run the self-test on the CPU's SECDED logic for accesses to RAM and Flash
- Configure and enable the desired interrupts
- Initialize the copy table, global variables, and structures
- Set up the real-time interrupt (RTI) module for generating periodic interrupts as required by the application
- Call the main() function to start the RTOS

The TMS570 microcontroller can be reset by several conditions. For this system, only two kinds of resets are used, as shown in Table 4.12, power-on reset and software reset. Software reset can be realized by writing to a 1 to bit 15 of System Exception Control Register (SYSECR) [37]. For power-on reset, since we are not able to control it by external voltage supervisor, a hardware watchdog timer is used. If the petting signal for the watchdog timer is no longer generated, then this will cause a watchdog timer timeout that will lead to a signal that cuts off the power to the MCU. This method will also power-cycle the whole satellite, so it should be used for critical condition only.

Condition	Description
Power-on reset	Reset the whole MCU. This reset signal is typically driven by an external voltage supervisor. In this system, this reset is triggered by the hardware watchdog by power-cycling the whole CubeSat.
Software reset	Reset the RAM, clocks, peripheral control registers and related power domains. This reset is generated by the application software.

Table 4.12: The Two Most Relevant Reset Types [37]

4.5.2 System Initialization Sequence

The system initialization sequence includes both the main() function in the application and the initialization task (init_task) of FreeRTOS.

The objective of the main() function is to start the FreeRTOS RTOS environment. It will first initialize the enabled peripherals with predefined values stored in FLASH. Then it reads error messages from the EEPROM, and then decodes and saves them to the error buffer. After that, it creates FreeRTOS user task, an initialization task, and then starts the task scheduler. The FreeRTOS multitasking environment starts from this point.

The initialization task, init_task, is used to do sensor and software initialization. As the most important task, it is the first and the only task created when the system starts, and it will delete itself when it completes execution. And to protect the initialization process, the whole task will run in a critical section, which guarantees that it will not be interrupted.

The initialization task first reads the configuration settings from the EEPROM and verifies the CRC checksum. If the data block passes the CRC checking, it will be saved to a global data structure. Since the working data is stored in RAM, it is called the “RAM copy”. If the CRC check fails, the task will read “factory copy 1” from EEPROM and do a CRC check as well. If the CRC check also fails, the task will read “factory copy 2” instead. Every CRC failure will be recorded to the error log. This sequence of steps is designed to ensure the correctness of the used configuration settings.

The second step is to initialize all the data structures (no hardware is initialized in this step), including the sensors, the MPPT module, the heater module, the battery module and the output channel module, with the configuration settings. Some of the settings will be used to initialize the hardware components later, other settings will be used as threshold values that determine the operation of other tasks.

Then init_task creates the other tasks. The first task being created is the housekeeping data task, getHK_task. The initialization task will exit the critical section after this creation to allow the housekeeping task to run for one execution cycle. In this cycle, the housekeeping task initializes all the sensors according to the sensor data structures and then reads and stores the resulting measurements for the first time. Functions called by other tasks will operate depending on these settings and measurements, which will be introduced later. After that, the system switches back to initialization task and enters a second critical section.

The remaining of tasks, as shown in Section 4.3.2, will then be created. Each failure during task creation, including failures to create the initialization task and the housekeeping task, will be logged in the error buffer and will lead to a software reset. When the initialization steps have been completed successfully, the initialization task will delete itself and exit the critical section. The other tasks will carry on executing at their own frequencies and priorities after the initialization sequence.

The priority of the initialization task is 14, but it can be changed to any priority other than 0 since it is running in a critical section.

4.6 System Monitoring and Housekeeping Data Collection

4.6.1 The Timestamp Mechanism

Timestamping is widely used in database management systems and operating systems [48]. It records the time when the certain information was created, exchanged, modified, or deleted so that users can later on track the sequence of data changes to help understand the causes of unexpected updates and failures. The real-time timestamp used in this system is stored along with housekeeping data and error messages. There is no real-world timer module on this chip, and so the timestamp depends on two different sources: one is the tick counter of FreeRTOS, which measures time intervals in ticks, and the other is Unix time in seconds transmitted from the OBC or ground station by command. Its data type is also divided into a uint32 type that stores seconds and a uint16 type that stores milliseconds.

The timestamp represents the runtime of the system by default when the system starts. The tick counter values, xTickCount and xTickOverflowCount, are internal variables in FreeRTOS that stores the current tick count and its carry. When the tick counter variable is full, the tick counter will be reset to zero and the overflow counter will increase by one. Both of them are 32 bits long. As mentioned above, in this system, the tick rate is programmed to be 1 KHz, so we can directly use them to calculate a real-time clock in milliseconds. The EPS software divide the number by 1000 and store the quotient as seconds while the remainder is stored as milliseconds.

Unix time is an integer-based representation of the number of non-leap seconds elapsed since 0:00 Jan 1, 1970 UTC. This method of computational timekeeping has a near monopoly on the representation of time in digital systems [48] so we chose to use it as the timebase of the timestamp. After the system starts, users can send the Unix time to the EPS system using the command "set_base_time", which will be introduced later. The system will store the Unix time and the tick counter value at this point in RAM as a timebase. The timestamp, after this command is executed becomes a real-world time depending on the timebase.

4.6.2 Check Active Task

The main responsibility of check active task, with name check_other_tasks_activity_task, is to pet the watchdog timer MAX16698A. As introduced in the previous section, the WDI pin of the watchdog timer is connected to a general-purpose input/output (GPIO) pin of the MCU. The check active task periodically sends the petting signal through this pin according to the activity of other tasks, described below.

```
while (1)
{
        if  (last_ticktime == prev_ticktime)            //Compares all the last_ticktime
                                                        variables in actual code
        {
                log the error message;
                suspend check active task;
        }
        else
        {
                pet the watchdog;
                prev_ticktime = last_ticktime;
        }
}
```

The pseudo code of the original design is shown above. last_ticktime is a set of global variables that stores tick counter values. Every task, except for the check active task and the receive command task, will update the current tick counter value at the end of each task period to its corresponding last_ticktime variable. prev_ticktime is a set of internal variables of the check active task that stores tick counter values of other tasks in the previous period. These variables are initially zero.

For each period, the check active task compares all the last_ticktime values with the prev_ticktime values. If all of the last_ticktime variables are different from their previous values, the check active task pets the watchdog timer and records the new ticktime. If any last_ticktime equals to the previous value, it means that the task has not completed successfully (is not active). Check active task will then log the error message with a timestamp and suspend itself, which will cause a WDI fault for the watchdog timer. This will lead to a power-cycle reset of the CubeSat.

However, in this design, the execution frequency of the check active task is limited by the other tasks. Since it won't pet the watchdog timer until it checks all the last_ticktime values, this task cannot be scheduled more frequently than all the other tasks, which means that check active cannot pet the watchdog timer very frequently. In this case, we have to use a larger capacitor at the SWT pin of MAX16698 to set a longer watchdog timeout period. According to the datasheet [36], the maximum recommended watchdog timeout is 217.36 ms, which is much faster than the task frequency. Although a larger capacitor is also acceptable, using one will lead to a lower accuracy. In the onboard test, a capacitor of 200 nF should set the period to be about 2 s, but the actual measured timeout delay is 1.69 s. Therefore, the previous design was replaced by following one. The pseudo code is shown below:

```
while (1)
{
        if  (last_ticktime != prev_ticktime)          //Compares all the last_ticktime
                                                      variables in actual code
        {
                prev_ticktime = last_ticktime;
                wdt_counter = 0;
        }

        wdt_counter++;                                //Increases all the wdt_counter
                                                      variables in actual code

        if (wdt_counter > task timeout period)        //Compares all the wdt_counter
                                                      variables in actual code
        {
                log the error message;
                suspend check active task;
        }
        else
        {
            pet the watchdog timer;
        }
    }
```

wdt_counter is a set of newly created internal variables in check active task that store an independent watchdog counter value for each task. These counter values are initially created zero as well. The check active task will first compare the last_ticktime with prev_ticktime. If they are not equal, the task records the new ticktime and clears the wdt_counter. This increases the wdt_counter by one. After that, check active task compares wdt_counter of every other task with its corresponding task timeout period. If all the wdt_counter values are smaller than their timeout periods, check active task pets the watchdog timer. If any value is larger than its timeout period, it means that the corresponding task has not successfully completed in a certain time (or is simply not active). Check active task will then log the error message with timestamp and suspend itself.

The design of check active task is a novelty which is equivalent to setting an independent software watchdog timer for each task. The check active task monitors the timeout condition for other tasks when petting the hardware watchdog timer. If any software watchdog timer of task times out, the situation will stop the pet signal. In this way, we eliminated the limitation on the frequency of check active task. We could use a smaller capacitor on the hardware watchdog timer so that the timeout period becomes much shorter, such as 10 ms, to avoid accuracy problems.

Moreover, with this design, we are able to add a new function, the software watchdog timer for command communication, to this task. There are two independent wdt_counter variables that can be cleared by a command sent from the OBC or the ground station. If these variables timeout, the situation will power-cycle reset the satellite as well. This function requires the OBC or ground station to repeatedly send a specific command, pet_watchdog, to the EPS board within a predefined timeout period. In this case, the EPS board cannot operate without OBC running. These two periods are part of the configuration settings. They are now set to 60 s and 1 week, respectively, and can be modified by command as well.

The priority of this task is 16, which is higher than all the tasks to be checked and its frequency is set to 100 Hz (10-ms task period).

4.6.3 Get Housekeeping Data Task

Get housekeeping data task (hereinafter referred to as the housekeeping task), with name getHK_task, is responsible for collecting data from all of the sensors. These data will be used by other operation tasks. The architecture of this task is very simple. In the acyclic part, it initializes all the sensors according to the configuration settings, which has been mentioned before. In the subsequent infinite loop, it periodically calls functions that read from all 31 INA226 current sensors and the one MAX6698 temperature sensor through MCU's I2C interface and then stores them in the corresponding sensor data structures. The data stored here is in raw form from the sensor register. Other tasks must convert the data to proper measurement units before using them.

The key consideration of this task is its execution frequency. This frequency is mainly limited by two factors: the measurement time of the sensors and the I2C transmission speed. The measurement time of current sensor and temperature sensor are configured to be 32.2 ms and 31 ms, respectively [24],[25]. The I2C communication speed is set to standard mode, which is 100 Kbit/s. For each task period, the housekeeping task reads data from 4 registers (16-bit register) in each current sensor and 2 register (8-bit register) in temperature sensor. The TI-supplied I2C driver that was incorporated using polling, so the housekeeping task will be in busy waiting for IO from sensors. During the onboard test, total transmission time for each period has been measured to be 65-66 ms. Therefore, the frequency must be set to be lower than 15.15 Hz according to the largest time period 66 ms. However, it is unnecessary to use a very high frequency since other tasks are not requesting sensor data very frequently. If the frequency is set too high, it will lead to an unnecessarily large power consumption, which should be avoided in satellite system. It is now set to 10 Hz (100 ms task period) to suit the frequency of the output channel control task.

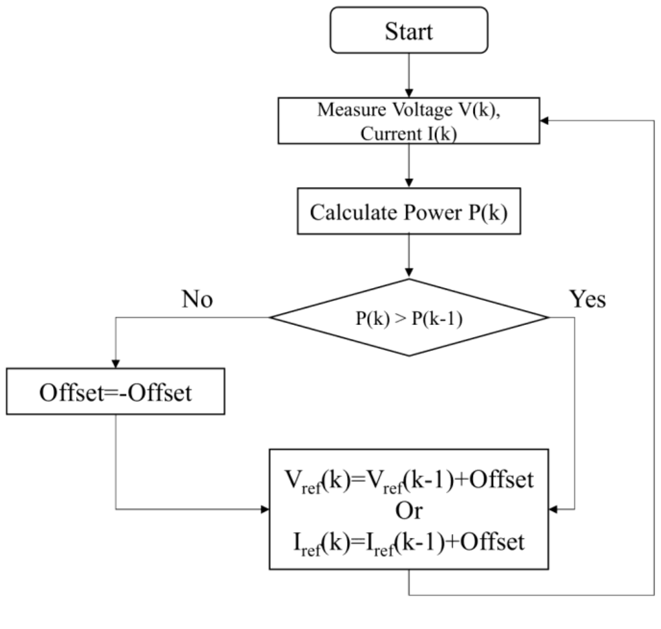
The priority of this task is 15, which is higher than all the tasks that rely on sensor.

4.7 Power Conversion and Management

4.7.1 Maximum Power Point Tracking

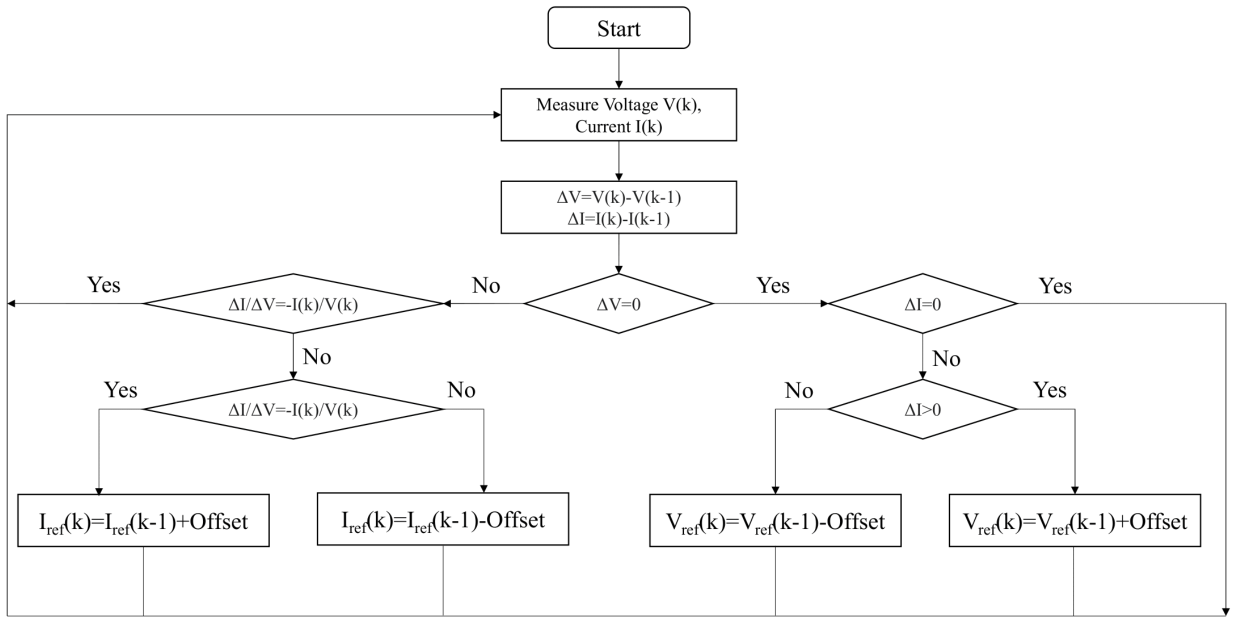
The theory concerning the controlling of the input power was introduced in Chapter 3. Therefore, in this section, the algorithm for tracking the MPP will be discussed.

As a popular research field, there are many available MPPT algorithms, and the number is still increasing [49]. Among them, the perturb and observe (P&O) method and the incremental conductance (INC) method are the most widely used due to the simplicity. Both methods are regarded as conventional algorithms, which are based on certain observations by applying a control signal to the power converter [50].



Start
Measure Voltage V(k), Current I(k)
Calculate Power P(k)
No
P(k) > P(k-1)
Yes
Offset=-Offset
$V_{ref}(k)=V_{ref}(k-1)+Offset$
Or
$I_{ref}(k)=I_{ref}(k-1)+Offset$

(a)


Start
Measure Voltage V(k), Current I(k)
ΔV=V(k)-V(k-1)
ΔI=I(k)-I(k-1)
ΔV=0
Yes
No
ΔI=0
ΔI/ΔV=-I(k)/V(k)
ΔI>0
I_ref(k)=I_ref(k-1)+Offset
I_ref(k)=I_ref(k-1)-Offset
V_ref(k)=V_ref(k-1)-Offset
V_ref(k)=V_ref(k-1)+Offset

(b)
Figure 4.4: (a) Flow Chart of the P&O Algorithm (b) Flow Chart of the INC Algorithm

Figure 4.4(a) shows the flow chart of the P&O algorithm. It works by adding an offset to the PV panel's operating voltage or current, which in our case is the output voltage of the DAC, according to the variation in operating power that is observed using the samples of voltage (V(k)) and current (I(k)) [50]. When the control algorithm finds the MPP, the operating power will constantly oscillate around the point with the offset value.

The theory behind the INC algorithm is based on the fact that the derivatives of (dP/dV) and (dP/dI) are zero at the MPP [50]. The relationship can be derived to the Equation 4.1:

$$\frac{dI}{dV} \cong \frac{\Delta I}{\Delta V} = -\frac{I}{V} \tag{4.1}$$

Therefore, the basic idea of this method is to calculate and compare the ratio of the increments of PV voltage and current, and then add suitable offsets to the operating voltage or current. The Figure 4.4(b) shows the flow chart of the INC algorithm. Similar to the P&O algorithm, there will also be an oscillation around the MPP.

Table 4.13 shows a qualitative performance comparison of the two algorithms, and the results are very close. The efficiency of an algorithm basically defines its tracking accuracy. According to this research [51], they have similar levels of efficiency. Both of them require sensor measuring voltage and current and both of them produce oscillations at steady state. The tracking speed of the P&O algorithm is lower, which can be improved by using a variable step size in the offset. The key point is the computation complexity. The calculation of INC includes division, which uses floating-point numbers. Although the TMS570 MCU has a floating-point coprocessor [38], it requires extra functions to enable division, and this will increase the power consumption. Considering this limitation, the P&O algorithm was chosen for the final design of the MPPT algorithm.

Algorithm	Steady-state oscillations	Efficiency	Tracking speed	Computation complexity	Sensor required
P&O	Yes	Medium	Low	Simple	I and V
INC	Yes	Medium	Medium	Medium	I and V

Table 4.13: Comparison of Two MPPT Algorithms [51]

```
if  (power < previous power)
{
        reverse direction of increment;
        counter=0;
}
else
{
        counter++;
}

call hunt algorithm function;
```

```
if  (direction of increment is positive)
{
        increase DAC output by one;
        step size;
}
else
{
        decrease DAC output by one;
        step size;
}
```

The pseudo-code of the implementation is shown above. The process is similar to the flow shown in Figure 4.4(a) with an extra function controlling the step size of the offset, called the hunt algorithm. In each comparison of the PV power, the P&O algorithm determines the direction of the next increment. If the direction remains the same, an internal counter will increase by one (it is initially zero). If the direction reverses, the counter will be cleared to zero. Then it calls the hunt algorithm function to update the step size. The pseudo-code of the hunt algorithm is shown below. If the direction of the increment remains the same for three consecutive periods, it will double the step size and then clear the counter to zero. If the direction reverses, the hunt algorithm will half the step size. In conclusion, the step size becomes larger when changing in the same direction and increases when the step direction is reversed. This method should greatly increase the tracking speed of the P&O algorithm without decreasing its oscillation at steady state.

```
if  (direction == previous direction)
{
        if  (counter>3)
        {
                double the step size;
        }
        counter=0;
}
else
{
        half the step size;
}
```

Figure 4.5 shows the simulation results of the variable step size P&O algorithm. The source of the input voltage and current is the same as that used in Section 2.3.2. The slope of the waveform represents the step size. The step size is small at the beginning and keeps increasing during the tracking process. When the power is oscillating around the MPP, the step size is decreased. The minimum and maximum values of the step size can be modified according to the hardware limitations. The real measured oscillation around the MPP is larger than this software simulated result.

The MPPT module functions are called by the power conversion task, named powerConversion_and_battCtrl_task in the code. This task first reads raw data from the corresponding sensor data structures, converts them into standard units and then stores them into the MPPT data structure. The algorithm function then calculates the DAC output value depending on these data. Finally, the DAC driver is called to actually control the output to be the calculated value.

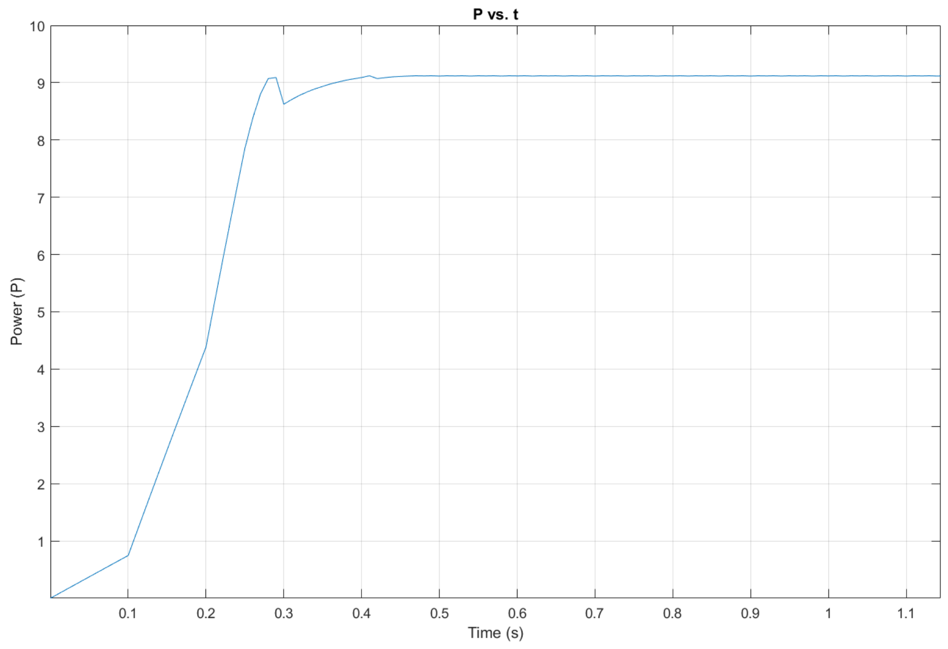


Figure 4.5: Simulation Power Waveform with the Improved P&O Algorithm

A hardware failure could possibly cause the boost converter to stop drawing current from the PV during the MPPT process. The algorithm will regard this as a low power condition and will keep increasing the offset, which would possibly damage the hardware. To protect the system from this situation, a protecting function was added. When the DAC output is set to be relatively high, but the power measured by sensor is close to zero, the algorithm will initialize DAC output to attempt a recovery from the situation.

4.7.2 Battery Protection

The battery, as the power storage of the EPS system, is the lifeline of the whole CubeSat when it is in eclipse. The objective of battery protection module is to prevent batteries from damage caused by overcurrent or over-/under-temperature conditions. The functions of this module are also called by the power conversion task, along with the MPPT module functions.

First, the power conversion task reads the raw data from corresponding sensor data structure, converts the measurements and stores the data to the battery data structures. Then the task checks the charging status of the batteries according to the direction of the current. The value read from shunt voltage register of INA226 current sensor is stored in 2’s complement format. If this number is negative (positive), this means that the battery is discharging (charging).

Next, the task controls the battery switches depending on the charging status and the temperature. For example, when the battery is charging, and its temperature falls lower than the minimum threshold allowed for safe charging (according to Table 3.3, it is 10 °C for the NCR18650B cell), the task function will turn the charging switch off. As mentioned before, the two battery pairs are connected in parallel with their own temperature sensors. Therefore, the switches are controlled separately as well.

Finally, the battery overcurrent protection function is called. It also checks the charging status of the battery, as well as the charging current. If there is battery overcurrent when charging, it means the current from the boost converter is too large. The protection function will then increase the minimum threshold of the DAC output to decrease the charging current and the MPPT algorithm is in hence adjusted. The algorithm cannot reach the real MPP, but only the MPP with the current limit. It keeps the battery safe at the price of drawing less power from the solar panels. When the overcurrent condition disappears, the function will reset the DAC threshold to its initial value so that the MPPT algorithm can recover to its normal operation. If the battery experiences overcurrent when discharging, then another function will be called by the channel control task, which will be discussed later.

The threshold temperatures for both charging and discharging, and the threshold for the current, are part of the configuration settings and can be modified by command.

The priority of the power conversion task is set to 10, which is lower than the heater control task that maintains battery temperature within proper range. The execution frequency of the task is set to 2 Hz (500-ms task period), which is limited by the boost converter and the comparator.

4.7.3 Heater Control

The heater control task, named heaterCtrl_task, controls the switches to the heaters that maintain the temperature of battery pairs. This control algorithm is a novelty that saves battery heater power when satellite is in eclipse.

First of all, the task reads raw data from sensor data structures, converts the data to standard units, and then stores to the heater data structure, just like in other tasks.

Next, the heater task updates the profile of the heater. Profile is a special variable in the heater data structure. There are two possible profiles: the "sunshine" profile means that the satellite is receiving sunlight, or is just about to exit eclipse, and the battery is charging; and the "eclipse" profile indicates that the satellite is in the Earth's shadow while the battery is discharging. The pseudo-code of the profile updating function is shown below. The minimum_threshold represents the minimum power that the solar panel can generate when in sunshine, which is currently set to 100 mW. The tumble_time is the waiting time to help ensure a correct profile switch and to avoid being misled by satellite tumbling. The orbit_period is the time that the satellite takes to complete one orbit. Since our CubeSat is designed to be operating in LEO, this value should be in the range of 88 - 92 minutes. The battery_heat_up_time is the time that the heater requires to heat up the battery to a proper temperature for safe charging, which depends on many factors. All four of the variables are included in the configuration settings and can be modified by command. The time_light_last_seen and time_of_first_light_per_orbit are the internal variables in the heater data structure and their contents are evident from their names. If the peek power from solar panels reaches the minimum_threshold at any time in a period of tumble_time, it means that satellite is likely in sunshine. The time_of_first_light_per_orbit records the time at this point and time_light_last_seen is kept updated with the current time when the profile state is sunshine. If the power generated by the solar panels is smaller than the minimum_threshold and remains for a period of tumble_time, it means that the satellite should go into the eclipse state. The profile will remain in the eclipse state until the difference between current_time and time_light_last_seen is equal to the difference between orbit_period and battery_heat_up_time, indicating that the satellite will go into sunshine soon. The heater profile state is switched to sunshine in advance to heat up the battery so that when the satellite gets out of the eclipse, the temperature is already increased to the safe charging range.

```
if  (solar power > minimum_threshold)
{
        if  (current_time - time_light_last_seen > tumble_time)        // in sunshine
        {
                time_of_first_light_per_orbit = current_time;
                set heater profile to sunshine;
        }

        if  (heater profile is sunshine)
        {
                time_light_last_seen = current_time;
        }
}

if  (current_time - time_light_last_seen > tumble_time)                // in eclipse
{
        if  (current_time - time_of_first_light_per_orbit > delay_time)
        //delay_time = orbit_period - battery_heat_up_time
        {
                set heater profile to sunshine;
        }
        else
        {
                set heater profile to eclipse;
        }
}
```

The other functions of this task are to control the heater switch depending on the heater profile and the temperature threshold setting. Since the heaters are relatively power hungry, we only use the heater to keep the battery temperature above the minimum thresholds. The default settings used for the NCR18650B battery are shown in Table 4.14.

Profile	Heater ON temperature (°C)	Heater OFF temperature (°C) (+3 of ON temperature)
In sunshine	< 12	> 15
In eclipse	< -18	> -15

Table 4.14: Temperature Thresholds of Heater in Both Profiles with NCR18650B Cell

With this control algorithm, the heater can keep the battery in different temperature ranges based on whether battery charging is anticipated, and it also predicts the time of exiting the eclipse to preheat the battery. Compared to the typical heater controller, which only keeps battery in a fixed range, heater power can be saved. However, its real power consumption and the comparison with other heaters still need to be tested in the future.

One consideration is that although the battery is able to work at the minimum threshold temperature while discharging in eclipse, operating at the low temperature will still decrease the efficiency of the battery. The optimal balance between consuming more power to heat up the battery to a higher temperature and keeping the battery operating at a low temperature with lower efficiency requires more tests and research.

Note that both the battery heater and the battery protection use information from the temperature sensor. If the sensor fails, there is a risk of damaging the battery. Future design should take this problem into account. Possible solutions can be adding range checking for measured temperature or adding redundant sensors.

4.8 Power Distribution

4.8.1 System States

The EPS has three running states. The first is “safe” mode, which is the initial mode of the system. In this mode, only the output channels to the OBC and UHF transceiver loads are available to be connected. This maintains the communication link between EPS and OBC and ground station (through the UHF transceiver), which is a basic function of the satellite. Human operators on the ground must always be able to safely contact the satellite in this mode despite the presence of other satellite activities. Second is the “full” mode, in which all the output channels are available to be on. All the other satellite subsystems are allowed to be active in this mode, such as the ADCS and the imager board, so that the CubeSat can be fully operational. Note that the output channels set to be available doesn’t mean that they will be always on. The task will still switch them off when error conditions occur (such as overcurrent condition). The last mode is “critical” mode, which should be rarely used. All the output channels are forced to be off in this mode. Entering critical mode will cause the ground station to lose contact with the satellite for a while.

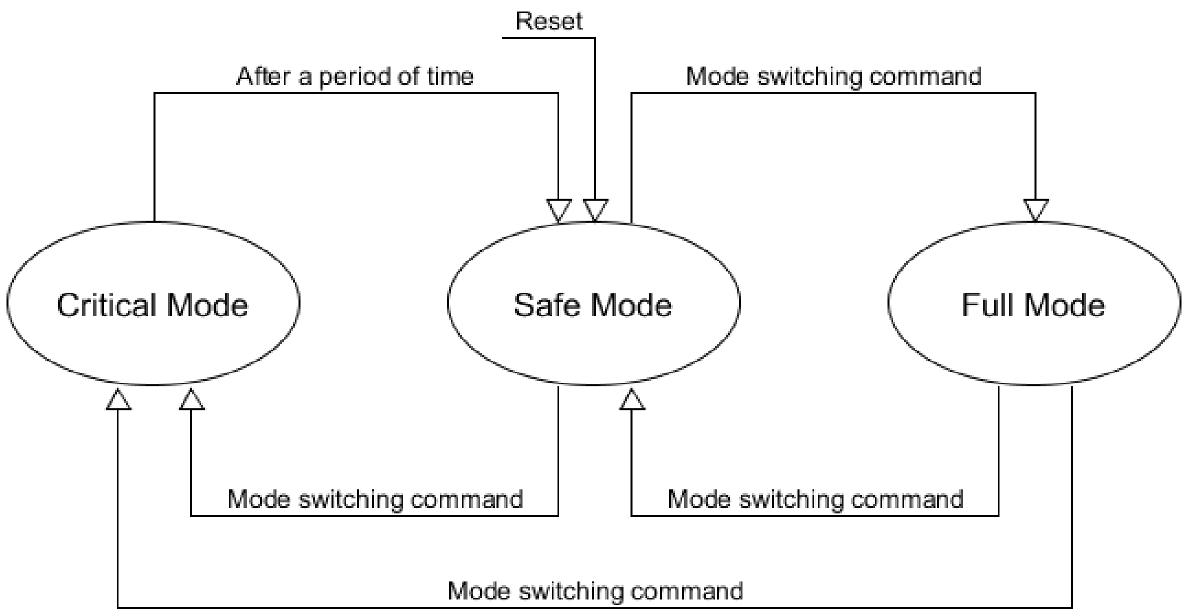


Figure 4.6: System State Switching

Figure 4.6 shows the system state transition diagram. Every time the system resets itself, it will change back to safe mode. Switching to other modes requires issuing the command “set_system_mode”, which will be described later. Switching from full mode to safe mode can only be caused by using this command as well. When in critical mode, the EPS runs on its own without any external loads. It will be switch back to safe mode after a period of time, which is a variable in the configuration settings called time_switch_from_critical_s. If it cannot switch back successfully, the OBC will not be able to pet the software watchdog timer, so the system will power-cycle reset the satellite after the OBC timeout in an emergency attempt to restore correct operation in the satellite.

4.8.2 Output Channel Control and Protection

The output channel controlling task (hereinafter referred to as channel task), with name outputchanCtrl_task, controls the switches for the output channels depending on various conditions.

Function name:	channel_set_group_mask	
Syntax:	void channel_set_group_mask (channel_data_t *Fchannel, channel_data_t *ChannelA, channel_data_t *ChannelB)	
Parameters (in):	Fchannel	Pointer to first element of the channel data structure array (equivalent to pointer to the data structure of channel 1)
	ChannelA	Pointer to the data structure of first channel being grouped
	ChannelB	Pointer to the data structure of second channel being grouped
Return value:	None	

Table 4.15: Set Channel Group Mask Function

Before going into the channel task process, one function of the channel task will be introduced in advance. There will be some conditions where one load uses multiple output channels or uses only one load that depends on another load. Obviously, these output channels cannot be on and off at the same time. Therefore, a special variable is added to the channel data structure, called group_mask. It is a uint32 type variable, and the lower 18 bits can used to represent the relationship of 18 channels. Each channel has its own group_mask, with the corresponding bit marked as 1 and all the other bits marked as 0 at initialization. For example: channel 1 group_mask=0x1, channel 2 group_mask=0x2, channel 3 group_mask=0x4, etc. There is a function called channel_set_group_mask (henceforth referred to as the group function) that can group channels together so that these channels can be on and off together. According to Table 4.15, the function is called to group together ChannelA and ChannelB. With the pointer to the first channel, the function traverses the group_mask of all 18 channels and groups all the related channels together with ChannelA and ChannelB. For example, if channel 1 and channel 2 are initially grouped together, both of their group_mask values should be 0x3. If the group function is called to group channels 2 and 3, it will actually group channels 1, 2 and 3 all together by setting their group_mask to 0x7.

The function that controls the channel switches will check the group_mask of all the channels each time it is called. Thus, it controls all of the grouped channels at the same time.

The channel task first reads raw data from sensor data structures, converts the data into units, and stores the unit data to the channel data structures, like other tasks. Then the task checks the mode of the system and enables/disables output channels depending on the current mode.

The next function monitors the voltage of the battery and controls the channels switches when the voltage reaches the predefined levels, on_level voltage and off_level voltage, of each channel. It also checks the charging status and the current of the battery, as mentioned in 4.5.2. If the battery overcurrent condition happens when discharging, the output channels will be switched off in order of their predefined priority until the battery current drops below the maximum threshold.

After that, the task calls the function which controls the switches based on the voltage of each channel. If the channel voltage exceeds its voltage limit, it will be switched off (so too will its grouped channels).

The latch-up and overcurrent protection for channels is configured by software and controlled by hardware. Each channel current sensor is configured with an overcurrent threshold in the configuration settings during system initialization. The alert output pin of the sensor is connected to a transistor that controls the switch of the channel. When a latch-up occurs, the channel current will exceed the threshold and the alert sensor output will assert and then cause the corresponding channel to trip. The software will switch the channel back to on after a period of time, reset_timeout_ms, which is a variable in configuration settings that can be modified by command.

However, as chips are exposed to increasing total lifetime doses of radiation, the semiconductor current will keep increasing. If the overcurrent threshold is fixed, there will be various problems. If the threshold is set too high, the protection module is not able to detect the SEL and trip the channel in time, which might allow a soft error to develop into a destructive error. If the threshold is set too low, the output channel might be switched off when the operating current become higher than the threshold. Therefore, there is a novel function in this task designed to self-adjust the overcurrent threshold of the current sensor when overcurrent occurs too frequently.

If one channel is tripped again in a user configurable period after recovered from an overcurrent condition, an internal trip counter will increase by 1. After three consecutive times, the overcurrent threshold will be auto-adjusted upward by a configurable amount. Otherwise, the trip counter will be clear to zero. The value of the threshold increment, maxI_increment_mA, is also stored in configuration settings and can be modified by command. With this design, the overcurrent protection can maintain the sensitive current limit near the operating current as well as detect the overcurrent condition caused by SEL.

All the channels are set to be off before the sensors are initialized with configuration settings, so it is safe when the channel task has not started.

The priority of the channel task is 6, which is lower than that of the battery task since the channel task requests data of battery data structure. The channel task therefore gets the lowest priority task among all the tasks, except for the idle task, which has a priority of 0. The frequency of the channel task is set to 10 Hz (100 ms task period).

4.9 Command Interface

For the convenience of diagnostics and configuration for both debugging before launch and operating from the ground station during the mission, a command interface is required for the EPS system. It is designed to be accessed in two ways. One way is through the MCU UART channel using simple serial terminal. The other method is reaching the interface remotely via CSP over the UART or CAN interface. Due to the limitation of time and device, the second method has not been fully implemented and tested. This section will focus on the first way based on a simple serial protocol.

The serial port driver functions of TMS570 are implemented and provided by TI. With the help of driver functions, data is encoded in ASCII and transmitted byte-by-byte. Therefore, the data is converted to an array of uint8 or char type before transmission. The transmission standard is set to fit the popular following settings: 8 bits, no parity, one stop bit, with a 9600 baud rate.

To decrease the length of transmitted data, commands are simplified into letters, where one letter identifies one command. The list of commands will be shown in next section.

4.9.1 Receive Command Task

This task is the only task in the software system that runs (unblocks) on the serial port interrupt and not a time delay interrupt. When the task is created and initialized, it is blocked until a command is received over the serial connection. It will store the received data into an array of uint8 type and then execute the command. After execution of each command, the task will return a value as acknowledgement, so that user can know the result of the command, and then the task blocks itself again waiting for the next command. If this task is not working properly, or not in active, the OBC will not be able to pet the software watchdog timer by command, so the system will power-cycle reset the satellite after OBC timeout. The list of the command return values are shown in Table 4.16.

#	Symbol	Description
0	CMD_EXECUTED	Command executed. This value will be returned after command is successfully executed. *For “reset” and “powercycle_satellite” command, this value will be returned before the command is executed.
1	CMD_INVALID	Invalid command. This value will be returned if the input command does not exist.
2	CMD_BAD_CRC	Failed in CRC checking. Used for flash-related commands only. This value will be returned if CRC checking failed when reading from FLASH.
3	CMD_WRONG_PARA_NUM	Wrong parameter number. This value will be returned if command has wrong number of input parameters (either more or less).
4	CMD_WRONG_PARA_VALUE	Wrong parameter value. This value will be returned if command input

		parameter has wrong value (exceed the length of its type).

Table 4.16: Return Values for the Receive Command Task

4.9.2 List of Commands

Commands are entered by alphabetic code followed by required parameters which are separated by spaces. At the end of each command, an Enter (\r) character is required so that the system knows that command entry is completed.

#	Commands
a	reset
b	get_status
c	set_base_time
d	revert_to_factory_config
e	force_revert_to_factory_config
f	revert_to_reboot_config
g	get_working_config
h	get_hk_all
i	get_hk_channel
j	get_hk_battery
k	get_hk_bc
m	set_all_working_config
n	set_working_config
p	update_factory_config
q	update_reboot_config
r	set_system_mode
s	set_channel
t	get_error_message
u	powercycle_satellite
v	pet_watchdog

A	print_status (display human readable data)
B	print_working_config (display human readable data)
C	print_hk_all (display human readable data)

Table 4.17: List of Commands

Table 4.17 shows the list of all the implemented commands. In this section, only a few commands will be introduced. Detailed information on all the commands can be found in the EPS ICD Software section in the Appendix A.

get_status and print_status are commands that requesting the same data but return the information in different formats. When the system receives either command, it stores all the required data into a status data structure. After converting the data into uint8 array, the get_status command will directly return the data, while print_status arranges the data with predefined text and then sends it to the serial port. The data from get_status are simple integers, such as 1 2 3 4. While the data from print_status are more like version: 1, voltage: 2 V, current: 3 A, power: 4 W. (All the data and text here are fake data.)

Command set_base_time, as mentioned before, updates the time base variable of the system with the current real-world time from the OBC/PC. It is used to timestamp the housekeeping data. Note that some systems don't have millisecond resolution in Unix time, and so the uint16 can be sent with 0.

Command set_system_mode has also been mentioned before. It is a simple command that just sets the running mode of the system.

The set_channel command forces the switch state of the target output channel, even if the channel is not available to be on in the mode when the command is executed.

The powercycle_satellite command power-cycles the satellite by suspending the check active task.

Human operators are also potential to be source of errors during the system operation. If a value out of range are accidentally entered by command, it could easily lead to serious problems. Therefore, a range checking function is created and will be called by the command functions that have input parameters. The range of parameters are defined as constants by software before launch.

Chapter 5: Testing and Evaluation

Testing is the important task of verifying that the developed system meets the requirements. The testing of the EPS embedded system design was divided into three steps: unit testing, functional testing and system testing.

Unit testing is used to verify the implemented software functions. With the specific testing tool, a unit test exercises a "unit" of code in isolation and compares the actual input-output behavior with the expected behavior. Unit tests invoke one or more methods from a function to produce observable results that are verified automatically [52]. These units can be verified to check the behavior of a specific aspect of the software.

Functional testing in this project is used to ensure that the system peripherals and hardware components provide the required functionalities. Since this thesis is mainly focused on the software system of EPS, the functional tests discussed here are all software-related. Hardware testing, such as whether the regulator on the EPS board can supply the defined voltage, will be included in the thesis of PhD student Stefan Damkjar.

System testing in this project runs the software code on the prototype board and tests its correct operation. It is a test for the EPS system but not the CubeSat system. Similar to unit testing, it is tested with some of possible conditions but at hardware level and the results will be compared with expected values. It verifies the combination of software and hardware.

All the tests mentioned above should be completed in order. Moreover, for aerospace equipment, normally there is one more step of testing called qualification testing. Qualification tests are used to ensure that the system can operate in an environment it is designed for and meets the requirements. As the design of the EPS is not in the last phase, components still require to be optimized and the final assembling of the prototype board has not been completed, the qualification testing had to be left as a future work.

5.1 Unit Testing

For C code unit testing, there are two apparently popular test frameworks available online: Ceedling [53] and Cgreen [54].

Test tool	Ceedling	Cgreen
Programming language	C	C, C++
Testing complexity	Medium	Simple
Supporting platform	Ruby’s Rake build system	Unix-based OS

Table 5.1: Comparison of Two Unit Testing Tools

While both tools are designed to support C software developers, Ceedling is promoted to be a better choice for embedded software [53] since it supports mocking hardware interfaces (such as I2C communication). However, Ceedling requires the installation of an external Ruby build system before testing and the implementation of a testbench is more complicated. As part of the AlbertaSat project, the testbench files are likely to be modified and used by other developers in the future. Therefore, complexity of a testbench is an essential consideration. Since other members of the CubeSat project are familiar with Cgreen, that tool was also selected for our testing. The Unix-based OS that we used for unit testing was Ubuntu 18.0. The hardware interfaces are directly tested on the development board RM46.

Testbenches need to be implemented and expected results for the considered states and inputs must be defined before unit testing can begin. A Cgreen testbench is like a C function with no parameters and no return value. To signal that they actually are tests, they are defined as an Ensure macro [54]. The function to test is called in this Cgreen macro and the expected output will be set as well. The sample pseudo code of a testbench of the function channel_set_group_mask is shown below:

```
Ensure(Channels, Set_group_mask_test)
{
        channel_set_group_mask(pchannelD, pchannelD+1, pchannelD+2);
        assert_that(channelD[3].group, is_equal_to(14));     //14 in Dec. = 1110 in Bin.
}

int main(void)
{
        fill the channel data structure with initialization values;
        create test suite;
        add test into the test suite;
        run the test suite and output the results;
}
```

The two arguments of the Ensure macro are the system under test (SUT) and the test name. In the macro, the tested function is called with input parameters. The call to assert_that() is the primary part of an assertion, which has actual output as the first input parameter and is complemented with a constraint, which is the second input parameter. In this case the constraint is is_equal_to(), which compares the actual output integer to the expected one and verifies that they are equal. Then in the main function, a test suite, which is the structure for test cases, is created and the test is added to the suite. After executing the test suite, the test results will be printed out.

For this single test, if the value is correct, the output will be like:

```
Running “main” (1 test) …
   “main”: 1 passes in 1 ms.
Completed “main”: 1 passes in 1 ms.
```

And if the value is incorrect, the output will print out the name and the location of the failure, which is shown below. Noted that channel_test.c is the name of the testbench file.

```
Running "main" (1 test) …
channel_test.c: 21: Failure: set_group_mask_test
        Expected [channelD[3].group] to [equal]     [14]
          actual value:                             [XXX] //XXX is the random
                                                          number to simulate a
                                                          wrong value
          expected value:                           [14]
   "main": 1 failure in 1 ms.
Completed "main": 1 failure in 1 ms.
```

A total of 18 functions were tested with Cgreen unit test. All of them passed the tests with designed conditions. Ideally, the unit testing should be repeated after every time that the code is modified to ensure that the functions are still correct. This is often called regression testing. In practice, the code is implemented in CCS on Windows OS while Cgreen is available on Ubuntu OS, so the unit testing was not done very frequently. The unit testing methodology should be improved in the future.

5.2 Functional Testing

In this project, functional testing is defined to mean testing the functionality and accuracy of the hardware components including the sensors, DAC, and watchdog timer. The controlling software functions of these components are also tested in this step. In this section, only a few parts of the tests are discussed. The complete tests and results can be found in List of Tests and Results on EPS Board in Appendix B.

5.2.1 Test of Sensors

All sensors were first tested by reading and writing complementary values (all zeros and all ones) to their registers to verify the basic read-and-write functionality. Other tests were designed and completed based on this prerequisite.

The INA226 current sensor was connected to a 100-mΩ reference resistor, and the configuration register was set to 0x4527 (sampling number is 16 and sampling time is 1.1ms). To test the accuracy of the sensor, we connected a multimeter to the tested component and then compared the sensor and multimeter measurements with each other. According to the results from Table 5.2, the error rate can be calculated, which is <5% for voltage and <1% for current. Since the sensor voltage is a directly measured value and the current is calculated by sensor internal functions, it is unusual that the current measurement is more accurate than the voltage measurement.

We also tested the maximum current limit of the sensor. It is 0.816 A with the current configuration. Maximum current measurement is limited by the register length of shunt register. The register is an int16 type register that stores a 2's complement number, so the maximum value of shunt register is 0x8000. For our application of the EPS board, the reference resistance should be lower, from 100 mΩ to 5 mΩ, to increase the current limit to 16.32 A.

	Voltage (V)	Current (A)
Multimeter	1.119	0.327
Sensor	1.168	0.329

(a)

	Voltage (V)	Current (A)
Multimeter	1.123	0.307
Sensor	1.172	0.309

(b)

Table 5.2: Accuracy Test of the INA226 (a) Test 1 (b) Test 2

The Max6698 temperature sensor is connected to an 8.66-kΩ external resistor with a NTC RT 01M1002J thermistor and the configuration register is same as the default. To test the accuracy of the sensor, we connected a multimeter to the thermistor and then compared the measurements. According to Table 5.3, the error rate is relatively large, about 10%. The accuracy of temperature sensor is not only determined by the feature of thermistor, but also influence by the lookup table in software. Since the LSB is 1 °C, and battery operation is not very temperature sensitive, 10%

error rate is still acceptable. This error rate should also be considered when configuring temperature threshold values.

Sensor measured temp. (°C)	Multimeter measured temp. (°C)
27	24.9 - 27.2
30	28.2 - 29.3

Table 5.3: Accuracy Test of the MAX6698

5.2.2 Test of Watchdog Timer

To test the watchdog timeout and reset timeout of the watchdog timer MAX16998A, we tried 100-nF and 200-nF capacitors connected to its SWT pin and a 1-uF capacitor connected to the SRT pin. According to the datasheet, we can measure both timeout periods at the same time by keeping the signal to WDI pin low and measuring the waveform of the RESET pin signal. The period when its high equals the watchdog timeout and the period when its low equals the reset timeout. The results are shown in Table 5.4. All of the periods are relatively far from the ideal values. The recommended maximum timeout periods for SWT and SRT are 217.36 ms and 116.09 ms, respectively, while our configurations have much longer periods. As mentioned before, a longer period still works, but is less accurate due to the current leakage. Therefore, the software code has been improved so that the periods can be set to a smaller value, such as 50 ms. The real periods should be much more accurate with the small values, but the test for that accuracy has not been done.

Capacitance (nF)	Ideal period (s)	Real period (s)
100	1	0.8753
200	2	1.6927

(a) Test of Watchdog Timeout Period (SWT pin)

Capacitance (uF)	Ideal period (s)	Real period (s)
1	2.9	2.701

(b) Test of Reset Timeout Period (SRT pin)

Table 5.4: Test of the MAX16998A Timeout Periods

The petting functions of the watchdog timer were also tested, and they all worked correctly. When the WDI pin is petted within the watchdog timeout period, which is 900 ms in our test, and the RESETIN pin is high, the RESET output pin remains high. When the RESETIN pin is set to low, the RESET pin also becomes low and remains low until a reset timeout period after the RESETIN goes back to high.

5.3 System Testing

Since the prototype EPS board was not yet been fully completed (some sensors were temporarily not fully populated, and some newly developed extensions were breadboarded in off-board circuits), some system tests were done by using a same sensor for multiple times. This section only introduces a few tests that verify some of the more innovative functions of this system. The complete tests and results can be found in List of Tests and Results on EPS Board in Appendix B.

5.3.1 EEPROM Process in Initialization Testing

The EEPROM reading and checking function is discussed in Section 4.5.2. For this test, four groups of data were prepared. In first group, all three copies store the normal configuration settings with the correct CRC checksum. In second group, data in the reboot copy was cleared to zeros, while the CRC checksum remained the same. In third group, based on the change in second group, the data in factory copy 1 was also cleared to zeros and CRC checksum is still unchanged. For the last group, data in all three copies were cleared to zeros with the initial CRC checksum retained. Then we started the system for four times, with four groups of data storing in the EEPROM respectively. The results of the test are listed below.

Data group	CRC check result		Copy used for initialization
1	Reboot copy	Passed	Reboot copy
	Factory copy 1	Unchecked	
	Factory copy 2	Unchecked	
2	Reboot copy	Failed	Factory copy 1
	Factory copy 1	Passed	
	Factory copy 2	Unchecked	

3	Reboot copy	Failed	Factory copy 2
	Factory copy 1	Failed	
	Factory copy 2	Passed	
4	Reboot copy	Failed	Factory copy 2
	Factory copy 1	Failed	
	Factory copy 2	Failed	

Table 5.5: EEPROM Reading and Checking Test

According to Table 5.5, the EEPROM reading and checking during initialization was founded to work properly. It reads and checks starting from reboot copy and will move to next copy if CRC check fails. If all three copies fail in CRC check, the system will be forced to use the data in factory copy 2.

5.3.2 MPPT Algorithm Testing

MPPT testing is one of the most important of the tests as well as the most time-consuming test. The details of testing of each component and the comparison of various controlling methods are included in Stefan Damkjar's PhD thesis. In this section, only some final tests of the module are shown.

Only two pieces of PV XTJ Prime solar panel were used for testing. A 230-W Godox LA200Bi LED video light was used as light source to simulate the sunshine. Due to the limitations of the equipment, the designed maximum output current, 4 A, of the EPS board could not be reached. Therefore, the actual test could only verify the functionality of the power conversion module and the MPPT algorithm, and so the values appearing in this section do not represent the final features of the system.

Figure 5.1 shows the performance testing plot of the MPPT algorithm with the UVLO controlling method. The algorithm was run on the target microcontroller on the development board. The test circuit consisted of development board and some breakout boards based on the schematic shown in Figure 3.4. The data are measured by oscilloscope and transmitted in real-time via the serial port.

According to Figure 5.1, the algorithm started from initial DAC output value, 3000, which impliess about 0.42 W of input power. In the first 100 seconds, the DAC output is increasing with a growing step size, and the input power is updated with the DAC changes. When the operating point exceeds the MPP, the input power will decrease. The algorithm then changes the direction of the increment and reduces the step size. Due to the scale, oscillations in the power waveform are not obvious in this plot. We can observe the plot of DAC output instead. With some back-and-forth adjustments, the algorithm reaches the MPP at about 170 s and then keeps fluctuating around that point. Since the frequency of MPPT software task in set to be 50 times slower than the real frequency (5-s period) to satisfy the delay of the oscilloscope, the algorithm should be able to reach MPP much faster for a real application. Comparing the input power and output power (which is not shown in this graph), the efficiency of this method is calculated to be 82.8%.

The FB controlling method was also tested, but without oscilloscope monitoring. The circuit was connected based on the schematic of Figure 3.5. The output of the boost converter was connected to a programmable load set to about 70 Ω, and so a higher MPP can be reached. Under the circumstances, the algorithm can always find the MPP in less than 30 MPPT task periods (which is 30 seconds here). The maximum power was measured to be 1525 mW, with the DAC output in the range of 3327 - 3391 and so the efficiency is about 95%.

In conclusion, both controlling methods were verified to work and are able to reach the MPP. Although the results shown in this section does not compares their accuracy, the FB control method was shown to be the better choice considering the conversion efficiency.

5.3.3 Battery Protection Testing

Battery protection functions are related to varies modules, and so they were tested together.

Temperature controlling

First, the temperature controlling functionality of the heater was verified. In this test, the heater and the thermistor of the temperature sensor were connected onto a piece of metal, which is used to simulate a battery pair (the battery was not available at that time). The heater profile in the heater data structure was set to be “sunshine” and heater_sunshine_temp_on_c and heater_sunshine_temp_off_c in configuration settings were set to 30 °C and 35 °C, respectively. The temperature is monitored by a multimeter and the data were transmitted in real-time. Figure 5.2 plots the changes of the resulting measured temperature over 20 minutes. The temperature curve was found to bounce between 30 and 35, which means that the heater successfully controlled the temperature in a closed loop.

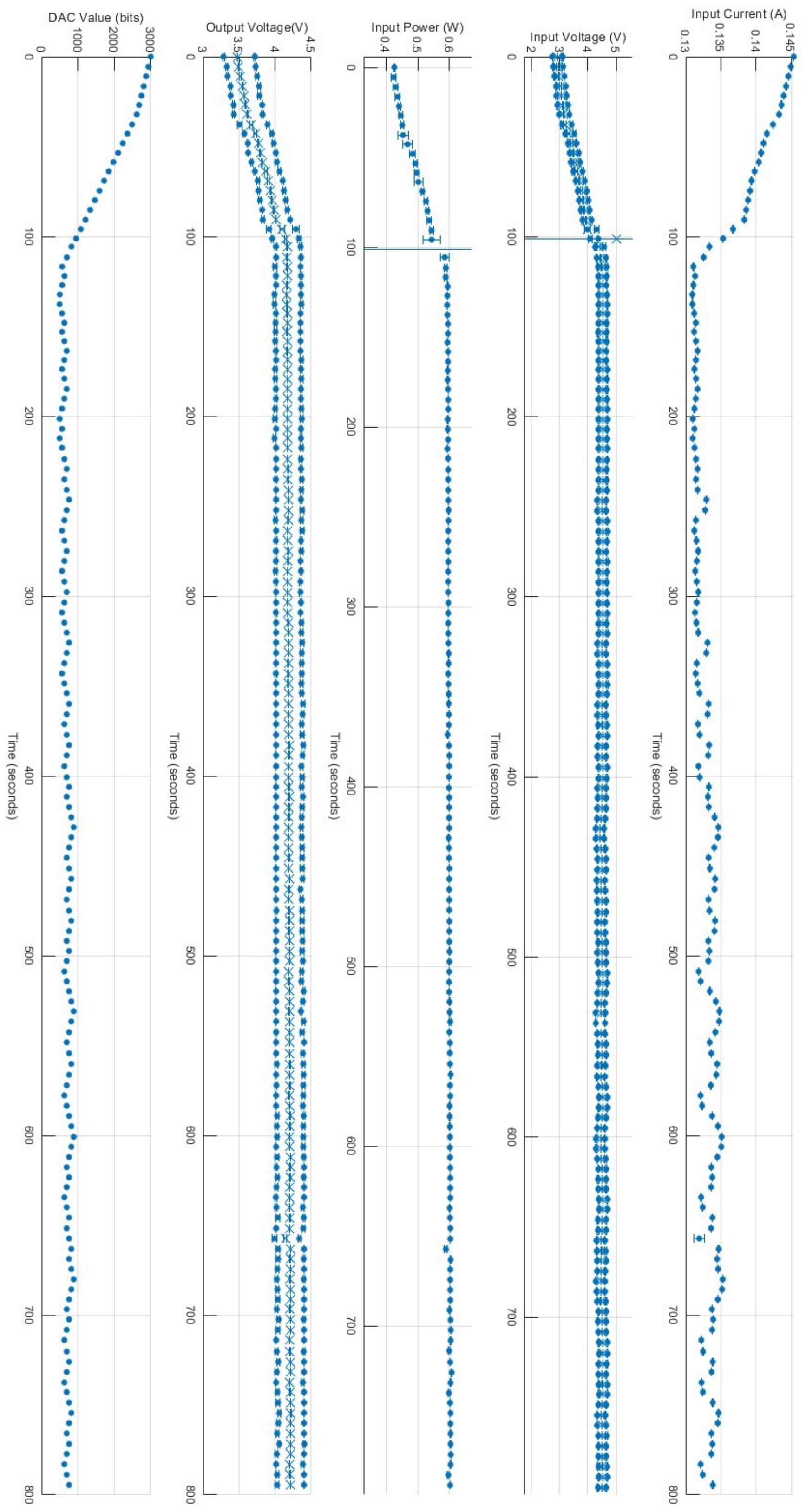


Figure 5.1: MPPT Algorithm Test of the UVLO Method

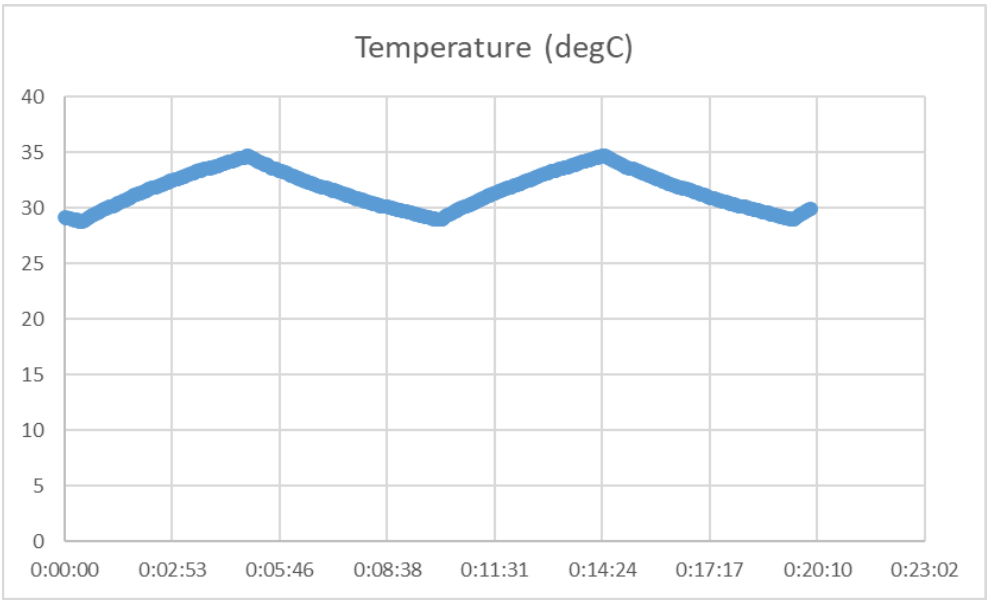


Figure 5.2: Temperature Response of a Simulated Battery

Heater profile switching

The functions of the heater profile were discussed earlier in Section 4.7.3. For this test, we need to set the parameters related to profile updating in the data structure: heater_tumble_threshold_time_s = 3, heater_solar_panel_threshold_power_mW equals 100, heater_orbit_period_s is set to 120 and heater_battery_heat_up_time_s = 10. A test function was created to print out the current heater profile, current time and two internal parameters time_of_first_light_per_orbit and time_light_last_seen.

The LED light was turned on before starting the system. In this condition, the heater profile is in "sunshine". The time_of_first_light_per_orbit parameter equals the time when the light is on for 3 seconds while the time_light_last_seen parameter equals the current time and keeps updating.

Then the LED light was turned off, which leads to the heater profile switching to "eclipse". Both internal parameters stop updating in this profile. When the current time minus the parameter time_of_first_light_per_orbit equals 110 s, which indicates the satellite will be in "sunshine" soon, the heater profile is changed to "sunshine".

Battery charging overcurrent protection

As mentioned before, the overcurrent protection when battery charging is related to the MPPT algorithm. A resistor was used to simulate the battery and another resistor was connected in parallel to decrease the resistance, and so increases in the battery current could also be simulated. The battery charging current threshold was set to 200 mA in this test. A test function was created to print out the current value and the minimum value of DAC output.

We turned on the LED light and started the system. The MPPT algorithm then found the MPP in within 20 MPPT task periods. The battery current was 106 mA and the DAC output was around 3359 when the MPP is reached. Next, we increased the current to 206 mA, which exceeds the threshold. At this point, the minimum DAC output was set to 3487, which is equal to the current DAC output plus a maximum step size. Since the DAC output is limited, the MPP was adjusted to a lower level, which should decrease the charging current of the battery. In this test, the charging current is simulated, so before it is changed back manually, the minimum DAC output kept increasing until it equaled the maximum DAC output (4096). When the current is changed to lower than the threshold, the minimum DAC output was reset to its initial value, which is zero in this case. And the MPPT algorithm also changed back to its normal mode, with the DAC output value being around 3359.

Battery discharging overcurrent protection

The hardware setup before the test is the same as for the previous one: a resistor simulating the battery and another parallel-connected resistor controlled by GPIO signal. This time, the discharging current limit was set to 300 mA while the initial current was 211 mA. Five output channels are set with various priorities, from 0 to 4, indicating the lowest priority to the highest priority, and they were all set to be on at the beginning. A test function was created to print out the off channel and its priority.

When the system was started, no output was printed by the test function since the channels are all switched on. Then we increased the current to 311 mA, which is higher than the threshold. The channel with priority 0 is printed at this moment. This confirmed that the lowest priority channel had been switched off. For the real system, the discharging current should decrease due to the decrease of loads at output channels. In this test, the simulated current remained at 311 mA as long as we don’t change it. Subsequently, more channels were switched off and with their identity and priority printed out. The channels were switched off in the order of their priority, from 1 to 4. When the current is changed back to 211 mA, those channels were switched back on at the same time.

5.3.4 Channel Protection Testing

Grouped channels switching

At the beginning of this test, channel 17 and channel 18 are grouped together by setting their group masks to 0x30000, and so they were both set to be on. A test command was created to set the group mask of channels when system is running. Another test function was created to print out the group mask of channels.

After the system started, we switched off channel 17 using the command "set_channel". Both channel 17 and 18 were observed to be off. Their group masks were still 0x30000 at this moment. Then we used the test command to group up channel 1 and channel 17 by setting their group mask to 0x10001 and 0x30001, respectively. After one channel task cycle, the printed group masks for all three channels became 0x30001, which means all three channels are grouped together. We then used "set_channel" to switch off channel 1 in this condition, and then all three channels were observed to be off.

Overcurrent threshold self-adjustment

In this test, a programmable load is connected to simulate a real loaded channel with the current of the channel being easily controlled by changing the load resistance. The initial channel current was set to 300 mA. Then the maxI_mA and maxI_increment_mA parameters in the channel data structure were set to 400 mA and 100 mA, respectively. Note that for the real system, the increment parameter should be set to about 5% of the initial threshold. We used a relatively large value here for the convenience of observing the test results. The reset_timeout_ms is set to 10 seconds. A test function was created to print out the maxI_mA value of the testing channel.

After starting the system, the load resistance was decreased to increase the current from 300 mA to 501 mA, which is higher than then initial threshold, 400 mA. The channel was then turned back to on after 10 seconds. Then while the load current at 501 mA, this operation repeated for three consecutive times. The maxI_mA was increased to 500 mA. The current at this point was still higher than the threshold, so the maximum limit is observed to increase again to 600 mA after another three cycles.

Chapter 6: Conclusions

6.1 Summary

As mentioned before, as a subsystem of the CubeSat that provides the regulated power source for the whole satellite using energy produced by solar panels, the EPS is one of the most critical systems of the spacecraft. Commercial EPS units are expensive and still have their limitations. This thesis research set out to develop a fully functional embedded software for a CubeSat-compatible Electrical Power Supply module for use on future AlbertaSat projects. And as an open-source project, this design will be made available to be shared with other CubeSat EPS design.

In this thesis, the background and fundamental theory were considered, and existing flight-proven systems were analyzed and discussed. Each module within the EPS software was carefully designed and the potential failures were considered. All the novel features were designed and realized successfully:

Eighteen output channels were implemented, with four hardware-configurable voltage supplies (1.2V, 3.3V, 5V and battery string voltage). Overcurrent protection for channels was designed with a self-adjusting threshold function to accommodate the expected increasing semiconductor current caused by the total lifetime dose of radiation. The overcurrent protection can maintain sensitive overcurrent limits near the operating current to detect smaller latch-up currents.

A low-power battery heater control algorithm is designed and implemented. There is adaptive battery temperature set point for discharge and charge, which saves power when a higher temperature (required for charging in some Li-Ion cells) is not necessary. As a further optimization, a predictive algorithm was designed for when the satellite emerges from eclipse so that the battery can be preheated for anticipated charging. Battery protection is also provided with two separate switches for battery controlling charging and discharging based on battery temperature.

Other important features are implemented and tested successfully as well:

For system resetting functions, redundant configuration settings were provided to store three copies in non-volatile memory EEPROM and the copies were protected with a CRC code. This design greatly increases the reliability of the system in harsh space environment since it can always start with a safe configuration.

For system monitoring, real-time sensor data is stored and updated in RAM frequently. Error messages can be logged and stored in non-volatile memory as well. Finally, the satellite can always be power cycled to recover from a serious error.

For power conversion, FB pin control method is used to control the boost converter duty cycle. Variable-step P&O was implemented as the MPPT algorithm. In the system tests, the MPP always found correctly. The overall efficiency of the boost converter with this MPPT algorithm was found to be about 95%.

For power management, batteries were designed to be protected from all foreseeable error conditions. The system can adjust the temperature within a proper range using the battery heater according to the predictive algorithm. Overcurrent condition in charging and discharging status are avoided by MPPT algorithm and channel control logic, respectively. The voltage is limited by the hardware circuit.

For the power distribution, output channel protection is also implemented to prevent channels from various error conditions. Besides overcurrent error, there are also undervoltage and overvoltage protection. Channels can be controlled in order of priority when battery voltage is low.

Testing and results have shown that each subsystem functions as designed, and that the overall design implements a fully functional system. Since the sensors are not fully populated on the prototype EPS board when the thesis was written (because of the on-going chip shortage), some modules are tested by using a same sensor for multiple times, which does not influence the result of the tests.

Although great care was taken that all components were specified for the LEO operating environment and that all calculations (where applicable) took worst case conditions into account, full environmental testing and qualification of the hardware should still be done when that becomes possible.

6.2 Future Work

There are many possible ways in which the EPS embedded system could be improved and extended in future work:

- TMS570 is a relatively powerful microcontroller, so it is able to run a more complicated but accurate MPPT algorithm. However, the algorithm is also limited by the provided features of the selected boost converter and current sensor. Many components may face a big change if we plan to use a better MPPT algorithm.
- The MCU is currently set to operate in maximum clock frequency, 180 MHz. In the future work, we should test a lower frequency to leave a redundant margin and to save power. The processor workload should also be tested again with the new frequency. Moreover, the TMS570 is relatively overkill since its capacity is not fully utilized, but it is one of few processors with ECC protection on SRAM and Flash memory. The high-clock frequency is unnecessary, and a lot of built-in functionality are disabled. It can be replaced by a less powerful MCU for power saving.
- When the MCU is idle, it should be placed in a low power mode, which has not yet been done. In a simple demonstration without FreeRTOS, we were able to wake the MCU from a doze or snooze mode with an interrupt, but not when running FreeRTOS, so the "tick-less" operation in FreeRTOS requires more debugging for this MCU.
- In the original design, EPS board is able to interface with CSP through CAN protocol. Although it could not be completed due to the time limit, it is not a complicated improvement.
- The vendor-provided I2C driver uses polling, wasting CPU cycles. A driver using interrupts and preferably DMA should be explored.
- The current sensor overcurrent alert may be triggered by a high in-rush current. We may need to add a setting for the maximum current for each channel for the initial in-rush current as well as the operating current.
- Both Flash and SRAM are protected by ECC. When there is an uncorrectable error in Flash detected by ECC, the MCU will abort. To avoid a reset caused by an uncorrectable 2-bit error when reading SRAM, a possible improvement is to add a low-priority task to scrub the memory word by word (read a word in memory with ECC checking, correct

any 1-bit error and then write it back). In this case, the error can be corrected in time, to prevent correctable 1-bit errors from turning into uncorrectable multiple-bit errors.

- The built-in ECC protection for EEPROM is now disabled. As future work, the configuration data stored in EEPROM should also be protected by the ECC, but to use the redundant copies, we must first have an error handler for an uncorrectable (2+ bit) ECC error.

References

[1] Liddle, J.D., Holt, A.P., Jason, S.J., O'Donnell, K.A. and Stevens, E.J., 2020. Space science with CubeSats and nanosatellites. Nature Astronomy, 4(11), pp.1026-1030.

[2] TX2-MC-128 Ver. 1.00 Ex-Alta 2 Mechanical Systems CDR Document. Retrieved 2022.

[3] AlbertaSat. 2022. The Experimental Albertan #2 Satellite. [online]. Retrieved 2022. Available at: <https://albertasat.ca/ex-alta-2/>.

[4] Villela, T., Costa, C.A., Brandão, A.M., Bueno, F.T. and Leonardi, R., 2019. Towards the thousandth CubeSat: A statistical overview. International Journal of Aerospace Engineering, 2019.

[5] Langer, M. and Bouwmeester, J., 2016. Reliability of CubeSats-statistical data, developers' beliefs and the way forward. 30th Annual AIAA/USU Conference on Small Satellites.

[6] Nwankwo, V.U., Jibiri, N.N. and Kio, M.T., 2020. The impact of space radiation environment on satellites operation in near-Earth space. Satellites Missions and Technologies for Geosciences.

[7] Cheremetiev, K., 2020. Design of reliable Electrical Power System for Foresail-1 Small Satellite.

[8] Gupta, V., 2017. Analysis of single event radiation effects and fault mechanisms in SRAM, FRAM and NAND Flash: application to the MTCube nanosatellite project (Doctoral dissertation, Université Montpellier).

[9] Murr, L.E. and Kinard, W.H., 1993. Effects of low earth orbit. American Scientist, 81(2), pp.152-165.

[10] Vincent, R.F., 2018. Orbits. In PHE 354 within Royal Military College of Canada.

[11] Blanks, H.S., 1980. The temperature dependence of component failure rate. Microelectronics Reliability, 20(3), pp.297-307.

[12] Sheard, B.C.V., 2015. An Electrical Power System for CubeSats.

[13] Plante, J. and Lee, B., 2005. Environmental conditions for space flight hardware: a survey.

[14] The CubeSat Program, Cal Poly SLO, CubeSat Design Specification, rev. 13, 2014.

[15] Molton, B.L., 2013. KySat-2 Electrical Power System Design and Analysis.

[16] Parida, B., Iniyan, S. and Goic, R., 2011. A review of solar photovoltaic technologies. Renewable and sustainable energy reviews, 15(3), pp.1625-1636.

[17] Kopp, G. and Lean, J.L., 2011. A new, lower value of total solar irradiance: Evidence and climate significance. Geophysical Research Letters, 38(1).

[18] Ibrahim, O., Yahaya, N.Z., Saad, N. and Umar, M.W., 2015, October. Matlab/Simulink model of solar PV array with perturb and observe MPPT for maximising PV array efficiency. In 2015 IEEE conference on energy conversion (CENCON) (pp. 254-258). IEEE.

[19] salingberbagi. 2010. Equivalent circuit of a solar cell [web log]. Retrieved 2022. Available at: < http://sunpowercell.blogspot.com/2010/09/equivalent-circuit-of-solar-cell.html >.

[20] Spectrolab, 29.5% NeXt Triple Junction (XTJ) Solar Cells, 2012.

[21] GomSpace, NanoPower P-series Datasheet P31u / P31us V9.0, 2014.

[22] NanoAvionics, EPS Electrical Power System Datasheet, rev. 3, 2021.

[23] Damkjar, S., ATLAS Electrical Power Supply Interface Control Document, Ver.1, 2021.

[24] Texas Instruments, INA226 High-Side or Low-Side Measurement, Bi-Directional Current and Power Monitor with I2C Compatible Interface, revised 2015.

[25] Maxim Integrated, MAX6698 7-Channel Precision Remote-Diode, Thermistor, and Local Temperature Monitor, rev. 3, 2007.

[26] Texas Instruments, TPS61088 10-A Fully-Integrated Synchronous Boost, revised 2021.

[27] Monolithic Power Systems, MP3432 Fully Integrated, Synchronous, 30W Boost Converter with Pass-through Mode and Programmable Switching Current Limit, rev. 1.0, 2019.

[28] Faubert, P., High Efficiency Solar MPPT Battery Charger Using LT8611 and AD5245 [web log]. Available at: < https://www.analog.com/en/technical-articles/high-efficiency-solar-mppt-battery-charger-using-lt8611-and-ad5245.html>

[29] Analog Devices, AD5304/AD5314/AD5324 2.5 V to 5.5 V, 500 μA, Quad Voltage Output 8-/10-/12-Bit DACs in 10-Lead Packages, rev. J, 2019.

[30] Linear Technology, LT1782 Micropower, Over-The-Top SOT-23, Rail-to-Rail Input and Output Op Amp, revised 2010.

[31] Burr-Brown Products, TLV3011 TLV3012 Nanopower, 1.8V, SOT23 Comparator with Voltage Reference, revised 2004.

[32] Panasonic, NCR-18650B Lithium-ion Rechargeable battery.

[33] Keil, P., Schuster, S., Lüders, C.V., Hesse, H., Arunachala, A. and Jossen, A., 2015, December. Lifetime analyses of lithium-Ion EV Batteries. In 3rd Electromobility Challenging Issues conference (ECI), Singapore, 1st–4th December.

[34] Texas Instruments, TPS4030x 3-V to 20-V Input, Voltage Mode, Synchronous Buck Controller, revised 2018.

[35] Texas Instruments, INA381 26-V, High-Speed, Current Sense Amplifier With Integrated Comparator, revised 2019.

[36] Maxim Integrated, MAX16997/MAX16998 High-Voltage Watchdog Timers with Adjustable Timeout Delay, rev.5, 2019.

[37] Texas Instruments, TMS570LS12x/11x 16/32-Bit RISC Flash Microcontroller Technical Reference Manual, revised 2018.

[38] Texas Instruments, TMS570LS1224 16- and 32-Bit RISC Flash Microcontroller, revised 2015.

[39] Embedded Markets Study, Integrating IoT and Advanced Technology Designs, Application Development & Processing Environments, 2019.
Available at: < https://www.embedded.com/wp-content/uploads/2019/11/
EETimes_Embedded_2019_Embedded_Markets_Study.pdf > (accessed in August 2022).

[40] Amazon Web Service, FreeRTOS Kernel Developer Docs [web log].
Available at: < https://www.freertos.org/features.html > (accessed in August 2022).

[41] Labrosse, J.J., µC/OS-II The Real-Time Kernel User's Manual, revised 2015.

[42] Xiao, L. and Liu, K., 2016, Analysis and Comparison of uC/OS and FreeRTOS Dynamic Memory Management Mechanism, Software Engineering, 19(5), pp.21-23.

[43] Ungurean, I., 2020. Timing comparison of the real-time operating systems for small microcontrollers. Symmetry, 12(4), p.592.

[44] Texas Instruments, TI FEE Driver User Guide, ver. 1.11, revised 2016.

[45] Shi-yi, C. and Yu-bai, L., 2006, June. Error correcting cyclic redundancy checks based on confidence declaration. In 2006 6th International Conference on ITS Telecommunications (pp. 511-514). IEEE.

[46] Martin, C., 2020. Choosing an Optimal CRC Polynomial [web log]. Retrieved 2022. Available at: < http://blog.martincowen.me.uk/choosing-an-optimal-crc-polynomial.html >

[47] Texas Instruments, Initialization of Hercules™ ARM® Cortex™-R4F Microcontrollers, revised 2013.

[48] Hauser, E., 2018. UNIX Time, UTC, and datetime: Jussivity, prolepsis, and incorrigibility in modern timekeeping. Proceedings of the Association for Information Science and Technology, 55(1), pp.161-170.

[49] Sarvi, M. and Azadian, A., 2022. A comprehensive review and classified comparison of MPPT algorithms in PV systems. Energy Systems, 13(2), pp.281-320.

[50] Motahhir, S., El Hammoumi, A. and El Ghzizal, A., 2020. The most used MPPT algorithms: Review and the suitable low-cost embedded board for each algorithm. Journal of cleaner production, 246, p.118983.

[51] Verma, D., Nema, S., Shandilya, A.M. and Dash, S.K., 2016. Maximum power point tracking (MPPT) techniques: Recapitulation in solar photovoltaic systems. Renewable and Sustainable Energy Reviews, 54, pp.1018-1034.

[52] Olan, M., 2003. Unit testing: test early, test often. Journal of Computing Sciences in Colleges, 19(2), pp.319-328.

[53] VanderVoord, M., Karlesky, Mike. and Williams, G., 2021. Ceedling [GitHub]. Retrieved 2022. Available at: < https://github.com/ThrowTheSwitch/Ceedling/blob/master/docs/Ceedling-Packet.md >

[54] Baker, M.L. and Freitas, J.J. Cgreen : Unit Tests, Stubbing and Mocking for C and C++, V1.6.0, 2022. Available at: < https://cgreen-devs.github.io/cgreen/cgreen-guide-en.html#_building_cgreen_test_suites >

Appendix A: EPS Software ICD

The EPS software is an independent network node designed for the CAN and serial communication.

There are two ways of transmitting commands to EPS. One is to directly send through the serial port of EPS which is designed for debugging. The other method is to send commands through the CAN interface. This is used by the CSP protocol of OBC. The second method is not fully implemented and tested.

Command Interface

List of Commands

#	Commands
a	reset
b	get_status
c	set_base_time
d	revert_to_factory_config
e	force_revert_to_factory_config
f	revert_to_reboot_config
g	get_working_config
h	get_hk_all
i	get_hk_channel
j	get_hk_battery
k	get_hk_bc
m	set_all_working_config

n	set_working_config
p	update_factory_config
q	update_reboot_config
r	set_system_mode
s	set_channel
t	get_error_message
u	powercycle_satellite
v	pet_watchdog
A	print_status (display human readable data)
B	print_working_config (display human readable data)
C	print_hk_all (display human readable data)
Available in Debugging mode	
Y	get_sensor_data
Z	set_sensor_data

Commands Description

- Commands are entered by alphabetic code followed by any parameters which are separated by spaces.

List of Status Return Values

The return values will be used for all the commands if not noted.

#	Symbol	Description
0	CMD_EXECUTED	Command executed. This value will be returned after command is successfully executed. *For "reset" and "powercycle_satellite" command, this value will be returned before the command is executed.
1	CMD_INVALID	Invalid command. This value will be returned if the input command does not exist.
2	CMD_BAD_CRC	Failed in CRC checking. Used for flash-related commands only. This value will be returned if CRC checking failed when reading from FLASH.
3	CMD_WRONG_PARA_NUM	Wrong parameter number. This value will be returned if command has wrong number of input parameters (either more or less).
4	CMD_WRONG_PARA_VALUE	Wrong parameter value. This value will be returned if command input parameter exceeds its pre-defined range.

Details for Each Command

1. Reset

Name:	reset	
Code:	a	
Parameter(s):	None	
Return value(s):	uint8	Status return value (see the list of status return values)
Description:	Warm reset the MCU. It is similar to power-on reset from black start except that a power domain PD1 for some control modules and some error signaling registers won't be reinitialized.	

Example:	case 1: a↵
	- 0
	case 2: F↵
	- 1
	case 3: a 1↵
	- 3	

2. Get key status of the system

Name:	get_status	
Code:	b	
Parameter(s):	None	
Return value(s):	uint8	Status return value (see the list of status return values)
	system_status_t	Data structure of system status
Description:	Get the key information of the system, including number of reset, version of current configuration, system runtime, etc.	
Example:	case 1: b↵ - 0 1 2 3 4 … //All the values in status data structure.	

3. Set base time

Name:	set_base_time	
Code:	c	
Parameter(s):	uint32	Unix time's second.
	uint16	Unix time's millisecond.
Return value(s):	uint8	Status return value (see the list of status return values)
Description:	Update the base time variable with current time from OBC/PC. It is used for timestamp of housekeeping data.	
Example:	case 1: c 123456 123↵ or c,123456,123↵ - 0 case 2: c 123456↵ - 3 case 3: c 123456 123456↵ //Second parameter exceeds its range. - 4	

4. Revert configuration to factory settings

Name:	revert_to_factory_config	
Code:	d	
Parameter(s):	uint8	Number of factory copy (1 or 2)
Return value(s):	uint8	Status return value (see the list of status return values)
Description:	Read from factory settings with CRC checking. If the CRC is good, copy factory settings to working copy. If the CRC is bad, do nothing.	

Example:	case 1: d 1↵
	- 0 //CRC correct.
	case 2: d 2↵
	- 2 //CRC wrong. Does not revert.	

5. Force revert configuration to factory settings

Name:	force_revert_to_factory_config	
Code:	e	
Parameter(s):	uint8	Number of factory copy (1 or 2)
Return value(s):	uint8	Status return value (see the list of status return values)
Description:	Copy factory settings to working copy without CRC checking.	
Example:	case 1: e↵ - 0 //CRC correct.	

6. Revert configuration to flashed settings

Name:	revert_to_reboot_config	
Code:	f	
Parameter(s):	None	
Return value(s):	uint8	Status return value (see the list of status return values)
Description:	Read from reboot settings with CRC checking. If the CRC is good, copy reboot settings to working copy. If the CRC is bad, do nothing.	

Example:	case 1: f↵
	- 0 //CRC correct.
	case 2: f↵
	- 2 //CRC wrong. Does not revert.	

7. Get configuration of working copy

Name:	get_working_config	
Code:	g	
Parameter(s):	None	
Return value(s):	uint8	Status return value (see the list of status return values)
	system_config_t	Data structure of configuration

Description:	Get working copy of configuration settings.
Example:	case 1: g↵ - 0 1 2 3 4 5 … //All the values in configuration data structure

8. Get all housekeeping data

Name:	get_hk_all	
Code:	h	
Parameter(s):	None	
Return value(s):	uint8	Status return value (see the list of status return values)
	channel_data_t	Data structure of output channels
	battery_data_t	Data structure of battery
	mppt_data_t	Data structure of boost converters
Description:	Get all the housekeeping data.	
Example:	case 1: h↵ - 0 1 2 3 4 5 … //All the values in channel data structures 1 2 3 4 5 … //All the values in battery data structures 1 2 3 4 5 … //All the values in mppt data structures	

9. Get housekeeping data of output channels

Name:	get_hk_channel	
Code:	i	
Parameter(s):	None	
Return value(s):	uint8	Status return value (see the list of status return values)
	channel_data_t	Data structure of output channels
Description:	Get housekeeping data of output channels	

Example:	case 1: i↵ - 0 1 2 3 4 5 … //All the values in channel data structures

10. Get housekeeping data of battery

Name:	get_hk_battery	
Code:	j	
Parameter(s):	None	
Return value(s):	uint8	Status return value (see the list of status return values)
	battery_data_t	Data structure of battery
Description:	Get housekeeping data of battery	
Example:	case 1: j↵ - 0 1 2 3 4 5 … //All the values in battery data structures	

11. Get housekeeping data of solar panel

Name:	get_hk_bc	
Code:	k	
Parameter(s):	None	
Return value(s):	uint8	Status return value (see the list of status return values)
	mppt_data_t	Data structure of mppt
Description:	Get housekeeping data of boost converter	
Example:	case 1: k↵ - 0 1 2 3 4 5 … //All the values in mppt data structures	

12. Set all the configurations of working copy

Name:	set_all_working_config	
Code:	m	
Parameter(s):	system_config_t	Data structure of configuration
Return value(s):	uint8	Status return value (see the list of status return values)
Description:	Set all the values in the working copy of configuration settings	
Example:	case 1: m 1 2 3 4 5 …↵ // Enter all the parameters of configuration data structure - 0	

13. Set a configuration value of working copy

Name:	set_working_config	
Code:	n	
Parameter(s):	uint8	Order number of the value in the data structure
	uint8/uint16/uint32	Actual value
Return value(s):	uint8	Status return value (see the list of status return values)
Description:	Set a value in the working copy of channel configurable settings	
Example:	case 1: n 1 123↵ - 0	

14. Set all the configuration of working copy to factory copy

Name:	update_factory_config	
Code:	p	
Parameter(s):	system_config_t	Data structure of configuration
	uint8	Number of factory copy (1 or 2)
Return value(s):	uint8	Status return value (see the list of status return

		values)
Description:	Set all the values in the factory copy of configuration settings (with CRC)	
Example:	case 1: p 1 2 3 4 5 … 1↵ // Enter all the parameters of configuration data structure followed by factory copy number - 0	

15. Update the configuration of working copy to reboot copy

Name:	update_reboot_config	
Code:	q	
Parameter(s):	None	
Return value(s):	uint8	Status return value (see the list of status return values)
Description:	Update working copy of configurable settings to reboot copy in flash	
Example:	case 1: q↵ - 0	

16. Set the mode of system

Name:	set_system_mode	
Code:	r	
Parameter(s):	uint8	Mode of the system
Return value(s):	uint8	Status return value (see the list of status return values)
Description:	Set the running mode of the system Mode: 0 - Critical mode (All the loads are OFF) 1 - Safe mode (Only OBC and UHF loads are available to be ON) 2 - Full mode (All the loads are available to be ON)	
Example:	case 1: r 2↵	

	- 0

17. Set the switch of output channel

Name:	set_channel	
Code:	s	
Parameter(s):	uint8	# of the channel
	uint8	Switch of the channel
Return value(s):	uint8	Status return value (see the list of status return values)
Description:	Set the switch of an output channel Switch: 0 - OFF 1 - ON	
Example:	case 1: s 1 0↵ - 0	

18. Get error message

Name:	get_error_message	
Code:	t	
Parameter(s):	None	
Return value(s):	uint8	Status return value (see the list of status return values)
	errMsg_t	Data structure of the error message
Description:	Get all the error message from the error log	
Example:	case 1: t↵ - 0 1 1 12345 12345 2 2 54321 54321 ...	

19. Powercycle the whole satellite

Name:	powercycle_satellite	
Code:	u	
Parameter(s):	None	
Return value(s):	uint8	Status return value (see the list of status return values)
Description:	Stop petting the watchdog timer to powercycle the whole satellite (power-on reset).	
Example:	case 1: u↵ - 0	

20. Pet the software watchdog

Name:	pet_watchdog	
Code:	v	
Parameter(s):	uint8	Source of petting
Return value(s):	uint8	Status return value (see the list of status return values)
Description:	OBC and the ground station should periodically pet the software watchdog timer to proof the activity of the communication. Mode: 0 – From OBC (period: 1 minute) 1 – From ground station (period: 1 week)	
Example:	case 1: v 0↵ - 0	

21. Print key status of the system

Name:	print_status	
Code:	A	
Parameter(s):	None	
Return value(s):	uint8	Status return value (see the list of status return values)
	system_status_t	Data structure of system status

Description:	Print the key information of the system, including number of reset, version of current configuration, system runtime, etc.
Example:	case 1: A↵ - 0 Number of reset time:1 Version of the current configuration: 0 System runtime: 1d 2h 3min 4s 5ms … //Print all the values in status data structure.

22. Print configuration of working copy

Name:	print_working_config	
Code:	B	
Parameter(s):	None	
Return value(s):	uint8	Status return value (see the list of status return values)
	system_config_t	Data structure of configuration
Description:	Print working copy of configuration settings.	
Example:	case 1: B↵ - 0 Battery minimum charging temp: 0 C Battery minimum charging temp: 20 C … //All the values in configuration data structure	

23. Print all housekeeping data

Name:	print_hk_all	
Code:	C	
Parameter(s):	None	
Return value(s):	uint8	Status return value (see the list of status return values)

	channel_data_t	Data structure of output channels
	battery_data_t	Data structure of battery
	mppt_data_t	Data structure of mppt
Description:	Print all the housekeeping data.	
Example:	case 1: C↵ - 0 Channel 1: ON, 1000mA, 5000mV, group mask-0x0001 Channel 2: OFF, 0mA, 0mV, group mask-0x0002 … //All the channels and their values Battery 1: Charging, 1000mA, 5000mV, 10C Battery 2: Discharging only, 1000mA, 5000mV, 0C … //All the batteries and their values Solar panel 1: 500mA, 4000mV … //All the values in mppt data structures	

*** Debugging mode commands**

24. Get register value of sensor

Name:	get_sensor_data	
Code:	Y	
Parameter(s):	uint8	i2c address
	uint16	register pointer
Return value(s):	uint8	Status return value (see the list of status return values)
	uint8/uint16	Value in the register
Description:	Get the value in a sensor register	
Example:	case 1: Y 0x46 0x01↵ - 0 0x4127	

25. Set register value of sensor

Name:	set_sensor_data	
Code:	Z	
Parameter(s):	uint8	i2c address
	uint16	register pointer
	uint8/uint16	Value of the register
Return value(s):	uint8	Status return value (see the list of status return values)
Description:	Set the a value in sensor register	
Example:	case 1: Z 0x46 0x01 0x4127↵ - 0	

Configurable Data Format

The configurable data is based in structures as specified below:

1) Configuration Data Structure

```
typedef struct{
    uint8_t batt_charging_temp_min_c;          //C. battery minimum charging temperature
    uint8_t batt_charging_temp_max_c;          //C. battery maximum charging temperature
    uint8_t batt_discharging_temp_min_c;       //C. battery minimum discharging temperature
    uint8_t batt_discharging_temp_max_c;       //C. battery maximum discharging temperature
    uint8_t heater_sunshine_temp_on_c;         //C. heater switch on temperature threshold in
                                                 sunshine profile
    uint8_t heater_sunshine_temp_off_c;        //C. heater switch off temperature threshold in
                                                 sunshine profile
    uint8_t heater_eclipse_temp_on_c;          //C. heater switch on temperature threshold in
                                                 eclipse profile
    uint8_t heater_eclipse_temp_off_c;         //C. heater switch off temperature threshold in
                                                 eclipse profile
    uint16_t overcurrent_protection_alert_mA[NUM_OF_INA226_OVERCURRENT_PROTECTION];
                                               //mA. overcurrent alert for ina226 of overcurrent
                                                 protection module
    uint16_t current_monitor_alert_mA[NUM_OF_INA226_MONITOR];
                                               //mA. overcurrent alert for ina226 of current
                                                 monitor module
    uint16_t overcurrent_protection_Rshunt[NUM_OF_INA226_OVERCURRENT_PROTECTION];
                                               //mΩ. Shunt resistance for overcurrent module
    uint16_t current_monitor_Rshunt[NUM_OF_INA226_MONITOR];
```

```
                                            //mΩ. Shunt resistance for current monitor
                                              module
    uint16_t battery_protection_Rshunt[NUM_OF_INA226_BATTERY];
                                            //mΩ. Shunt resistance for battery protection
                                              module
    uint16_t channel_monitor_Rshunt[NUM_OF_INA226_CHANNEL];
                                            //mΩ. Shunt resistance for channel monitor
                                              module
    uint16_t power_conversion_Rshunt[NUM_OF_INA3221];
                                            //mΩ. Shunt resistance for boost converter
    uint16_t batt_charging_current_limit_mA;
                                            //mA. battery charging current
    uint16_t batt_discharging_current_limit_mA;
                                            //mA. battery discharging current
    uint16_t heater_tumble_threshold_time_s;
                                            //sec. cubesate tumble threshold time
    uint16_t heater_solar_panel_threshold_power_mW;
                                            //watt. solar panel minimum power threshold
    uint16_t heater_delay_time_s;           //sec. the delay time since last time exiting eclipse
                                              to switch battery heater profile to the charge
                                              profile
    uint16_t dac_init;                      //initial output value of DACs
    uint16_t dac_stepsize_init;             //initial stepsize of DACs
    channel_config_t chan_config_data[NUM_OF_CHANNELS];
                                            //array of channel configurable data structures
}system_config_t;
```

2) Configurable Data Structure of Output Channels

```
typedef struct

{

    uint8_t priority;                          //priority of channel

    uint16_t onlevel_mV;                       //mV. battery voltage level to turn the channel on

    uint16_t offlevel_mV;                      //mV. battery voltage level to turn the channel off

    uint16_t maxI_mA;                          //mA. overcurrent alert threshold

    uint16_t maxV_mV;                          //mV. overvoltage limit

    uint16_t minV_mV;                          //mV. undervoltage limit

    uint16_t maxI_increment_mA;                //mA. increment of overcurrent threshold

    uint16_t reset_timeout_ms                  //ms. OFF time before the channel is turned back to on from an

                                                 overcurrent condition

    uint32_t group_mask;                       //group mask channels. 1 bit for each channel. If grouped with a

                                                 channel, that bit is set to 1

}channel_config_t;
```

Housekeeping Data Format

The housekeeping data is based in structures as specified below:

1) Battery Data Structure

```
typedef struct

{

    uint8_t num;                    //# of battery pair. Starting from 1

    uint8_t sw[2];                  //Switches. SW1 is charging switch and SW2 is discharging switch

                                      1:ON, 0:OFF

    uint8_t status;                 //Status of battery (depending on the direction of current)

                                      1:charging, 0:discharging

    uint8_t temp;                   //Voltage of thermistor

                                      -20C:10110111; 0C:10011110; 100C:00001111

    uint16_t current;               //mA. Battery current

    uint16_t voltage;               //mV. Battery voltage

}battery_data_t;
```

2) Heater Data Structure

```
typedef struct

{

    uint8_t num;                    //# of heater. Starting from 1.

    uint8_t sw;                     //switch of heater. 0:off, 1:on

    uint8_t temp;                   //Voltage of thermistor

    uint8_t profile;                //The profile of the heater. 0: in eclipse, 1: in sunshine

    uint8_t profile_counter;        //A counter used by profile updating function
    uint32_t init_time;                      //sec. Initial time of profile judgment.
                                               Used by profile updating function
}heater_data_t;
```

3) Output Channel Data Structure

```
typedef struct
{
    uint8_t num;                             //# of channel
    uint8_t priority;                        //priority of channel
    uint8_t mode;                            //mode that indicating whether the channel should be on or off
    uint8_t sw;                              //switch of channel. 0:off, 1:on
    uint8_t resume;                          //Resume statement. It is set to 0 when initialized.
                                               When resume is set to 1, switch on the channel in next period
    uint8_t trip_counter;                    //the variable that counts the consecutive trip time of the channel
    uint8_t trip_counter_reset_timer;        //the timer that used to clear the trip counter
    uint16_t current;                        //mA
    uint16_t voltage;                        //mV
    uint32_t group_mask;                     //group mask channels. 1 bit for each channel.
                                               If grouped with a channel, that bit is set to 1
}channel_data_t;
```

System Status Data Format

```
typedef struct

{

    uint8_t num_of_reset;              //number of the reset time

    uint8_t config_ver;                //version of the current configuration setting

    uint16_t runtime_ms;               //ms. millisecond of system runtime

    uint16_t current_time_ms;          //ms. millisecond of current time

    uint32_t runtime_s;                //s. second of system runtime

    uint32_t current_time_s;           //s. second of current time

}system_status_t;
```

Appendix B: List of Tests and Results on EPS Board

*Parameters with underscore are just testing values, not the final value used on EPS board

Functional Testing

Functions	Designed Test	Results
Current sensor INA226 * The reference resistor used is 100 mΩ. * The configuration used is 0x4527 (Sampling number is 16 and sampling time is 1.1ms)	Test registers with read & write functions - Read the original values in all registers and compare with power-on reset values (according to the datasheet). - Write to all programmable registers with 0x0. - Read from these registers and check values. - Write to these registers again with 0xFFFF. - Read from these registers and check values.	Reading and writing functions are working well. All bits of all the programmable registers are tested with complement values and they work well.
	Test the alert function - Set the alert enable register to <u>0x8001</u> (overcurrent alert, and alert pin auto reset when the fault is cleared). - Set the <u>current threshold</u> to 310 mA. - Start with 200 mA current. Monitor the alert pin and	Alert function is working well. Alert pin asserted when the overcurrent condition happened.

	increase the measured current to exceed the threshold (0.311 A).	
	Test the accuracy of the sensor - Connect a multimeter to the tested component. - Compare the multimeter measured values and sensor measured values.	1. The voltage error rate < 5% and current error rate < 1% Test 1: [| Voltage (V) | Current (A)] [Multimeter | 1.119 | 0.327] [Sensor | 1.168 | 0.329] Test 2: [| Voltage (V) | Current (A)] [Multimeter | 1.123 | 0.307] [Sensor | 1.172 | 0.309] 2. Maximum current limit (with 100 mΩ reference resistor) is 0.816A. * This is limited by the shunt register (maximum 0x8000).
Temperature sensor MAX6698 *The thermistor used is NTC RT 01M1002J. *The external resistor used is 8.66 kΩ *The configuration used is same as default value (0x0)	- Read the original values in used registers (only test the registers that will be used by EPS board) and compare with power-on reset values (according to the datasheet). - Write to all programmable registers with 0x0. - Read from these registers and check values. - Write to these registers with 0xFFFF.	Reading and writing functions are working well. All bits of all the programmable registers are tested with complement values and they work well.

Test 1:

	Voltage (V)	Current (A)
Multimeter	1.119	0.327
Sensor	1.168	0.329

Test 2:

	Voltage (V)	Current (A)
Multimeter	1.123	0.307
Sensor	1.172	0.309

	- Read from these registers and check values.	
	Test the accuracy of the sensor - Connect the thermistor of the sensor with a multimeter. - Compare the multimeter measured values and sensor measured values.	The error rate < ±10% (see table below)
Digital-to-Analog converter AD5324 Use 1.2V reference voltage	Test the controlling of the DAC - Connect a multimeter at output channel 1 of DAC. - Sweep channel 1 back and forth (0-4095). - Check the output voltage at this channel.	The output voltage at channel 1 increases and decreases linearly between 0 - 1.2V.
Watchdog timer MAX16998A	Test the watchdog timeout and reset timeout - Connect a 100 nF (and 200nF) capacitor to the SWT pin to set the watchdog timeout period t_{WD} (about 1s (and 2s) according to the datasheet). - Connect a 1 uF capacitor to the SRT pin to set the reset timeout period t_{RST} (about 2.9s according to the datasheet). - Connect a GPIO pin to RESETIN pin to control the voltage by digital signal. The	1. Watchdog timeout: - Ideal period 1s (see table below) - Ideal period 2s (see table below)

Sensor measured temp. (C)	Multimeter measured temp. (C)
27	24.9 - 27.2
30	28.2 - 29.3

- Ideal period 1s

Test number	Measured period (s)
1	0.875
2	0.875
3	0.876

- Ideal period 2s

Test number	Measured period (s)
1	1.692

signal is set to high (3.3V) at the beginning.

- Stop petting the WDI (keeping WDI low) and check the waveform of the reset pin. The period when it’s high equals watchdog timeout and the period when it’s low equals reset timeout.

- Pet the WDI pin with 1s period and check if the reset pin remains high.

- Set the GPIO signal to low (0V) then back to high and check the waveform of reset pin

2	1.693
3	1.693

2. Reset timeout:

Test number	Measured period (s)
1	2.619
2	2.733
3	2.751

3. The reset pin remains high if the WDI pin is petted properly.

The reset pin is low when GPIO is set to low, and change back to high after a reset period when the GPIO is set to high.

System Testing

Reset & Initialization Module

Functions	Designed Test	Results
Read & write in EEPROM when reset	Test read & write functions to EEPROM - Write an array of uint8 (0-99) data to block 1 of EEPROM. - Read from block 1 of EEPROM. - Check the memory address of EEPROM (starting from 0xF0200000) block 1 by CCS tool.	Reading and writing functions are working well. Read values are the same as write values. The values are stored properly in the EEPROM according to the memory browser of the CCS.
	Test the initialization functions related to EEPROM 1. - Write SRAM copy with proper CRC code to boot copy in flash. - Read flashed reboot copy (with CRC checking). 2. - Write incorrect SRAM copy (all zeros) with correct CRC code to boot copy in flash. - Read flashed reboot copy (with CRC checking). - See if it reads factory copy 1 (with CRC checking). 3. - Write incorrect SRAM copy (all zeros) CRC code to reboot copy and factory copy 1 in flash. - …	Function reading from the reboot copy is working well. Functions reading from other copies when the CRC check fails are also working well.

	- See if it reads factory copy 2 (with CRC checking). 4. - Write incorrect SRAM copy (all zeros) CRC code to all the copies in flash. - … - See if it reads factory copy 2 without CRC checking when factory copy 2 CRC fails.	

Watchdog Module

Functions	Designed Test	Results
Overcurrent Protection	Test overcurrent condition function - Set threshold overcurrent_protection_alert_mA (310 mA) to an ina226 sensor. - Give it a current (311 mA) exceeding the threshold and check if the watchdog reset pin asserts.	Overcurrent protection function is working well. The watchdog timer reset pin asserts when overcurrent happens.
Check active task	Test the delay of the first petting when system is reset - Connect an oscilloscope to the power supply of the board and the WDI pin of the watchdog timer. - Power on the system and record the result.	The delay from the power-on to the first petting signal is ~110 ms.

	Test the reaction when a task is inactive. (WDI fault) - Enter a testing command to suspend one task (getHK task) - Check if the watchdog output pin asserts.	The WDI output is working well. When the microcontroller fails to pet the watchdog timer for 3 consecutive timeout periods. Reset pin also asserts for one reset period when WDI fault happens.

Housekeeping Module

Functions	Designed Test	Results
Store the housekeeping data with timestamp	Print out the housekeeping data - Create test functions to print out the housekeeping data of all tested sensors periodically. - Run the housekeeping task. - Check the printed data.	Housekeeping task collects data properly with the timestamp.
Minimum period of the housekeeping task	Measure the time reading from all the sensors - Read all four required registers from one ina226 for 31 times (to simulate 31 ina226) and two required registers from one max6698 - Measure the time period	Time period: 65-66ms *If we set the task period to be 80ms, the real period is 79.2ms.

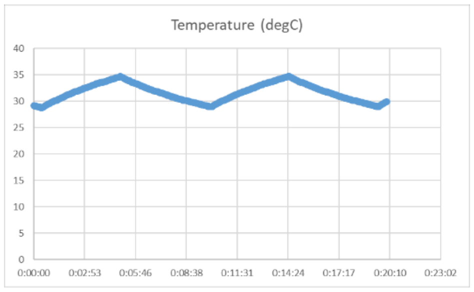
Power Conversion Module

Functions	Designed Test	Results
FB controlling	Control the FB pin of the boost converter - Connect the circuit according to the schematic. - Sweep the DAC output back and forth (0 - 4095) and check if the input of the converter has a proper IV curve.	The FB controlling method is working well. The waveform of input of boost converter is a perfect IV curve.
MPPT algorithm *Tested with only two solar panels. *A 230W LED video light is used as light source (Godox LA200Bi)	Test the MPPT algorithm as well as the Hunt algorithm - Connect a programmable load to the boost converter and set the resistance (~70 Ω). - Turn on the LED light and run the task. - Wait and see if the boost converter reaches the MPP.	The algorithm can always find the MPP in less than 30 task periods. (DAC output at 3327 - 3391, input power is 1525mW) According to the change of step size of DAC output, the hunt algorithm also works well.
	Test the charging overcurrent protection of the battery - Set the batt_charging_current_limit_mA (200mA) to the data structure of the battery. - Use a resistor to simulate the battery. And another resistor will be parallel connected to it to decrease resistance when a GPIO pin is set to high, so that the battery current increases. - Turn on the LED light and run the task to charge the battery. Wait until it reaches the MPP (I=106mA at this point). - Increase the battery current (set GPIO pin high)	The charging overcurrent protection function works well. It will adjust the output value of DAC to decrease the battery current. When the current is changed back, it can also change the algorithm back to normal.

	to let it exceed the threshold (206mA). - Check if the MPP is adjusted to a lower point so that the battery current decreases (battery current will not change in this test since it is not connected to the boost converter. we are just simulating this current). - Change the current back (set GPIO pin low. for the real system, current should decrease when MPP adjusts) and check if the MPP is changed back.	

Battery Protection Module

Functions	Designed Test	Results
Battery switch	Measure the feature of the battery switch (Transistor) - Use a multimeter to measure transistor gate voltage. - Use a multimeter to measure leakage current when the transistor is off.	1. Transistor gate voltage: ON: 7.89 - 7.90V OFF: 0 - 0.1mV 2. Leakage current: 17.53 - 17.55uA

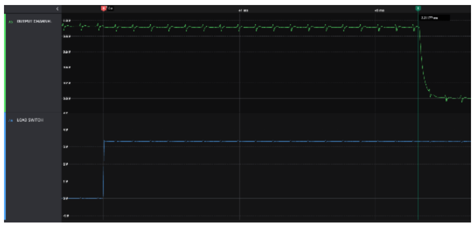
	Test the overtemperature protection when the battery is charging - Set batt_charging_temp_max_c (40C) and batt_charging_temp_min_c (30C) in the battery data structure and use a load resistor to simulate the battery. - Heat/cool the thermistor of the temperature sensor to let the temperature be higher/lower than the threshold temperatures (29C/41C). - Monitor the charging switch of the battery and see if it is ON/OFF properly.	The overtemperature protection function works well. The switch is ON when the temperature is in the range (30 - 40C) and is OFF when out of the range. *Over-temperature response: ~858 ms ± getHK task period (The measured response may be larger than the real value since the exact time of temperature crossing the threshold is hard to measure)
Heater switch	Test if the heater can keep the temperature in a close loop - Set the heater profile to be in sunshine and set heater_sunshine_temp_on_c (30C) and heater_sunshine_temp_off_c (35C) to the temperature sensor. - Connect the heater and the thermistor of the temperature sensor to a piece of metal to simulate the battery. - Run the task and monitor the temperature of the metal.	The heater switch functions work well. When the metal temperature is below 30 C, the heater will be turned on and when the temp. is above 35 C it will be turned off. In this case, the metal temperature can be kept in the range of 30 - 35C. 

	Test the profile switching functions of the heater - Set the parameters related to profile updating of the data structure: heater_tumble_threshold_time_s = 3, heater_solar_panel_threshold_power_mW = 100, heater_orbit_period_s = 120, heater_battery_heat_up_time_s = 10. - Create a test function to print out the current heater profile, current time and two internal parameters time_of_first_light_per_orbit and time_light_last_seen. - Turn on the LED light and start the task. - Check the printed value. - Turn off the LED light and check the printed value again.	The profile switching functions work well. When the LED light is on, the heater profile is in sunshine. The time_of_first_light_per_orbit equals the time when the light is on for 3 seconds. The time_light_last_seen parameter equals the current time. When the LED light is off, the heater profile is in eclipse. Both internal parameters stop updating. When the current time minus time_of_first_light_per_orbit equals 110s, the heater profile is changed back to sunshine.

Output Channel Control Module

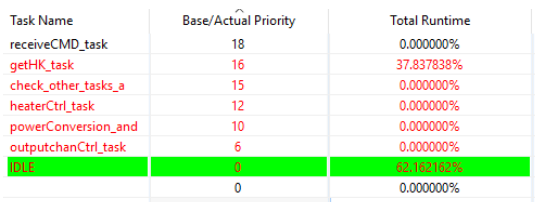
Functions	Designed Test	Results
Channel switch * Load resistors are used to simulate real loads.	Test the switch depending on the battery voltage - Set onlevel_mV (3300mV) and offlevel_mV (1200mV) to channel data structure. - Change the battery voltage to be higher/lower than the threshold voltage. - See if the channel switch is on/off properly.	Switch functions work well. When the battery voltage is higher than 3.3V, the channel is turned on. When the battery voltage is lower than 1.2V, the channel is turned off.

	Test the discharging overcurrent protection of battery - Set the batt_discharging_current_limit_mA (300mA) to the data structure of the battery. - Use a resistor to simulate the battery. And when a GPIO pin is set to high, another resistor will be parallel connected to it to decrease resistance, so that the battery current increases. - Run the task to discharge the battery. - Increase the battery current (set GPIO pin high) to let it exceed the threshold (311mA). - Print out the output channels that are switched off and their priority until all the channels are off. (Since only a few channels are available when testing, we monitored the switch pins of the channels instead of channel outputs)	The discharging overcurrent protection of the battery works well. The output channels are switched off in order. The lowest priority channel will be off first and the highest priority channel will be off at last.
	Test the switch depending on the channel voltage - Set maxV_mV (3300mV) and minV_mV (1200mV) to channel data structure. - Change the channel voltage to be higher/lower than the threshold voltage. - See if the channel switch is on/off properly.	Switch functions work well. When the channel voltage is higher than 3.3V, the channel is turned on. When the channel voltage is lower than 1.2V, the channel is turned off.

	Test the switch depending on the channel current - Set maxI_mA (300mA) to the current sensor. - When a GPIO pin is set to high, another resistor will be parallel connected to the load resistor to decrease resistance, so that the channel current increases. - Increase the battery current (set GPIO pin high) to let it exceed the threshold (311mA). - See if the channel switch is on/off properly.	Switch functions work well. When the channel current exceeds 300 mA, the channel is turned off. * This overcurrent protection is triggered by hardware. The overcurrent response time is ~ 2.3ms 
	Test the group function of channels - Set the group mask of two channels (channel 17 and 18) to group them together. - Use command to turn on/off channel 17. - Check if channel 18 is on/off as well.	Group function works well. Channel 18 is changed with channel 17 ON/OFF.
Channel limit adjustment * A programmable load is used to simulate real load.	Test the overcurrent threshold adjustment function - Set maxI_mA (400mA) maxI_increment_mA (100mA) and to the channel data structure. - Create test functions that print out the maxI_mA parameter. - Decrease the load resistance at the channel to set the current to be	The overcurrent adjustment function works well. When the overcurrent condition happens for 3 times in a short period, the threshold parameter will be increased until it is higher than the channel current. (600mA)

	501mA. - Check the printed value and see if it is increased.	

Others

Functions	Designed Test	Results
Low power mode Main Oscillator frequency: 16 MHz	Get the runtime information of tasks - Install and configure the analysis tool - Run the system and check the runtime statistic	 (see task runtime table below) The idle task occupies about 62.1% while the getHK task takes 37.8%. (In future work, an interrupt driven I2C driver will be explored. DMA will also be investigated.) All the other tasks execute so fast that their runtimes are ignored compared to these two tasks.
	Measure the power consumption of the system in LPM (power supply V=1.2V) * Since there are still some unsolved problems in LPM functions, the power consumption measured in this test is from fully doze / fully snooze, which means the system is not switching between LPM and normal mode. - Create the test commands to control the system go into LPM. - Run the system in normal mode	Measured Results (see table below) Calculated Results (according to task runtime statistics) (see table below)

Task Name	Base/Actual Priority	Total Runtime
receiveCMD_task	18	0.000000%
getHK_task	16	37.837838%
check_other_tasks_a	15	0.000000%
heaterCtrl_task	12	0.000000%
powerConversion_and	10	0.000000%
outputchanCtrl_task	6	0.000000%
IDLE	0	62.162162%
	0	0.000000%

Measured Results

Mode	Current (mA)	Relative Power Percentage
Normal	153.5 ± 0.5	100%
Fully Doze	6.9 ± 0.3	4.5%
Fully Snooze	16.0 ± 0.2	10.4%

Calculated Results (according to task runtime statistics)

Mode during Idle	Power (mW)	Relative Power Percentage
Busy wait	184.2	100%

	and measure current. - Use command to let the system go into doze mode and measure the current. - Power-cycle the board to reset the system. - Use command to let the system go into snooze mode and measure the current.	Doze | 74.95 | 40.69% Snooze | 81.74 | 44.38%